SIEMENS

Ганс Бергер

Автоматизация посредством STEP 7 с использованием STL и SCL и программируемых контроллеров SIMATIC S7-300/400

Издание 2001

Указания по технике безопасности

Данное руководство содержит указания, которые вы должны соблюдать для обеспечения собственной безопасности, а также защиты от повреждений продукта и связанного с ним оборудования. Эти замечания выделены предупреждающим треугольником и представлены, в соответствии с уровнем опасности следующим образом:

Опасность
указывает, что если не будут приняты надлежащие меры предосторожности, то это **приведет** к гибели людей, тяжким телесным повреждениям или существенному имущественному ущербу.

Предупреждение
указывает, что при отсутствии надлежащих мер предосторожности это **может привести** к гибели людей, тяжким телесным повреждениям или к существенному имущественному ущербу.

Осторожно
указывает, что возможны легкие телесные повреждения и нанесение небольшого имущественного ущерба при непринятии надлежащих мер предосторожности.

Осторожно
указывает, что возможно повреждение имущества, если не будут приняты надлежащие меры безопасности.

Замечание
привлекает ваше внимание к особо важной информации о продукте, обращении с ним или к соответствующей части документации.

Квалифицированный персонал

К монтажу и работе на этом оборудовании должен допускаться только **квалифицированный персонал**. Квалифицированный персонал – это люди, которые имеют право вводить в действие, заземлять и маркировать электрические цепи, оборудование и системы в соответствии со стандартами техники безопасности.

Надлежащее использование

Примите во внимание следующее:

Предупреждение
Это устройство и его компоненты могут использоваться только для целей, описанных в каталоге или технической документации, и в соединении только с теми устройствами или компонентами других производителей, которые были одобрены или рекомендованы фирмой Siemens.
Этот продукт может правильно и надежно функционировать только в том случае, если он правильно транспортируется, хранится, устанавливается и монтируется, а также эксплуатируется и обслуживается в соответствии с рекомендациями.

Товарные знаки

SIMATIC®, SIMATIC HMI® и SIMATIC NET® - это зарегистрированные товарные знаки SIEMENS AG.

Некоторые другие обозначения, использованные в этих документах, также являются зарегистрированными товарными знаками; права собственности могут быть нарушены, если они используются третьей стороной для своих собственных целей.

Siemens AG
Департамент автоматизации и приводов
Промышленные системы автоматизации
П\я 4848, D- 90327, Нюрнберг

Отказ от ответственности

Мы проверили содержание этого руководства на соответствие с описанным аппаратным и программным обеспечением. Так как отклонения не могут быть полностью исключены, то мы не можем гарантировать полного соответствия. Однако данные, приведенные в этом руководстве, регулярно пересматриваются, и все необходимые исправления вносятся в последующие издания. Мы будем благодарны за предложения по улучшению содержания.

ПРЕДИСЛОВИЕ

Новая система автоматизации SIMATIC объединяет отдельные частные решения системной автоматизации на основе однородной архитектуры в единое целое от аппаратуры "полевого" уровня непосредственно до управления процессом. Это достигается с помощью интегрированных в систему средств конфигурирования и программирования, с помощью управления данными в системе коммуникаций с программируемыми контроллерами (SIMATIC S7), специализированными компьютерами (SIMATIC M7) и системами управления (SIMATIC C7). С помощью программируемых контроллеров трех выпускаемых серий перекрываются все области их применения при решении задач автоматизации процессов в целом и в производственной сфере в частности. При этом изделия серии S7-200 используются как компактные контроллеры ("микро-PLC"), изделия серий S7-300 и S7-400 используются как модульные функционально расширяемые контроллеры для применения в системах низкой и высокой производительности.

Система STEP 7, представляющая собой дальнейшее развитие STEP 5, является программным обеспечением для программирования в новой системе SIMATIC. Windows 95/98 Microsoft или Windows NT Microsoft были выбраны в качестве операционных систем, чтобы пользователь STEP 7 мог в полной мере использовать знакомый ему интерфейс пользователя для стандартных ПК (оконная система, работа с манипулятором "мышь").

Для программирования блоков STEP 7 предназначены языки программирования, соответствующие международному стандарту DIN EN 6.1131-3: STL ("statement list" - список мнемоник, Assembler-подобный язык), LAD ("ladder diagram" - "контактный план", представление в виде логических схем), FBD ("function block diagram" - "функциональный план", язык функциональных блок-схем) и поставляемый по отдельному заказу пакет SCL ("Structured Control Language" – "структурированнный язык управления", Pascal-подобный язык высокого уровня). Кроме того по специальным заказам могут быть также поставлены дополнительные пакеты ПО, предоставляющие следующие языки программирования: S7-GRAPH (для графической разработки программ систем автоматизации SIMATIC в виде последовательности шагов и переходов между ними), S7-HiGraph (для графической разработки программ систем автоматизации SIMATIC в виде графа состояний системы и переходов между ними) и CFC ("continuous function chart" - план соединений программных блоков; при этом проектирование на CFC похоже на проектирование с FBD). Пользователю предоставляется полное право выбора из этого набора различных методов представления для описания функций при решении его задачи управления.

Широкие возможности адаптации в представлении задачи управления, которую необходимо решить, значительно упрощают работу в STEP 7.

Эта книга содержит описание языка программирования STL для S7-300/400. В первом разделе представлены обзор систем автоматизации S7-300/400 и изложены основы работы со STEP 7. Следующий раздел адресован начинающим пользователям STEP 7 или пользователям, переходящим к STEP 7 от работы с системами управления на базе контакторов и реле. Здесь описаны базовые функции для дискретного управления с помощью языка программирования STL и показано, как с помощью двоичных функций преобразуются значения сигналов. Здесь представлены основы двоичных вычислений, работа компаратора, преобразование типов данных. Используя STL, Вы сможете обрабатывать управляющую программу (управлять ходом выполнения программы) и разрабатывать структурные программы. Вы сможете создать циклически выполняемую основную программу, Вы также сможете использовать управляемые событиями подпрограммы, такие как подпрограммы, управляемые поведением контроллера при запуске, а также подпрограммы обработки ошибок или проявлений неисправности.

Один раздел книги посвящен описанию языка программирования SCL. Язык SCL особенно подходит для программирования сложных алгоритмов или для задач управления данными, и это сближает SCL с языками программирования высокого уровня. Блочная структура STEP 7 позволяет создавать SIMATIC S7-программы из блоков, написанных на различных языках.
Данная книга включает описание программы для преобразования программ STEP 5 в программы STEP 7, а также краткий общий обзор системных функций и набора функций для языков программирования STL и SCL.

В этой книге представлен пакет программного обеспечения STEP 7 версии 5.1, а также поставляемый по специальному заказу пакет S7-SCL версии 5.1.

Erlangen, март 2001

Ганс Бергер
(Hans Berger)

КРАТКИЙ ОБЗОР СОДЕРЖАНИЯ КНИГИ

Краткий обзор программируемых логических контроллеров серий S7-300/400	Функции PLC в сравнении с системами управления на базе реле	Числа, управление содержанием аккумуляторов	Управление ходом выполнения программы, функции блоков
Введение	**Базовые функции**	**Функции цифровой обработки**	**Управление выполнением программы**
1 SIMATIC S7-300/400 программируемые контроллеры Структура PLC (аппаратные компоненты S7-300/400); области памяти; распределенная периферия (PROFIBUS DP); коммуникации (подсети); адреса модулей; адреса областей памяти	**4 Операторы двоичной логики** Функции AND (И), OR (ИЛИ), Exclusive OR (Исключающее ИЛИ); вложенные функции	**9 Функции сравнения** Сравнение в соответствии с типом данных INT, DINT и REAL	**15 Биты состояния** "Двоичные" флаги, "цифровые" флаги; механизм EN/ENO
	5 Функции памяти Функции назначения, установки и сброса; проверки прихода фронта сигнала; пример системы управления конвейера	**10 Арифметические функции** Четыре функции для типов данных INT, DINT и REAL; прибавление констант; декремент/инкремент	**16 Функции перехода** Безусловный переход, переходы по условию, проверяемому в RLO, BR, "цифровых" флагах; распределенный и циклический переход
2 Средства программирования STEP 7 Редактирование проектов; конфигурирование станций; кнфигурирование сетей; Symbol Editor (редактор имен); редакторы STL/ SCL; интерактивный режим; тестирование программ	**6 Функции перемещения данных** Функции загрузки Load функции Transfer; функции аккумулятора системные функции передачи данных	**11 Математические функции** Тригонометрические; Arc-функции; степенные функции; логарифмы	**17 Главное реле управления (MCR)** MCR–зависимость, MCR-область, MCR-зона
	7 Функции таймера Запуск таймеров SIMATIC пяти разных типов; IEC-таймеры	**12 Функции преобразования** Преобразование типа данных; дополнения	**18 Функции блоков** Вызов блока, завершение обработки блока; Временные и статические локальные данные
3 S7-программа Выполнение программы; тип блока; блоки кодов STL/ SCL; блоки DB; адресация переменных; представление констант; типы данных (обзор)	**8 Функции счетчика** SIMATIC-счетчики; прямой/обратный счет; сброс/установка; IEC-счетчики	**13 Функции сдвига** Сдвиг и циклический сдвиг	**19 Параметры блоков** Формальные и фактические параметры; объявление и назначение; передача параметров блоку
		14 Логика для типа Word Функции AND (И), OR (ИЛИ), Exclusive OR (Исключающее ИЛИ)	

Обработка программы пользователя	Обработка сложных переменных, косвенная адресация	Описание языка программирования SCL	Преобразование S5/S7 библиотеки блоков, обзоры
Обработка программы	**Работа с переменными**	**Язык SCL**	**Приложения**
20 Основная программа Структура программы; управление циклом сканирования; (время отклика, стартовая информация; фоновая обработка); функции программы; коммуникации посредством распределенной периферии и глобальных данных; коммуникации SFC и SFB	**24 Типы данных** Структура типов данных; объявление и использование простых и сложных типов данных; программирование пользовательских типов данных UDT	**27 Введение, элементы языка** Адресация, операторы, выражения, назначения **28 Управляющие операторы** IF, CASE, FOR, WHILE, REPEAT, CONTINUE, EXIT, GOTO, RETURN	**32 Программа S5/S7-преобразования** Подготовка к преобразованию; преобразование S5-программ; последующая обработка
21 Управление прерываниями Аппаратные прерывания; прерывания по времени таймера; прерывания по времени суток; прерывания многопроцессорной обработки; прерывания, управляемые событиями	**25 Косвенная адресация** Указатель на область; указатель на DB; указатель типа ANY; косвенная адресация посредством памяти и регистра (внутризонная и межзонная адресация); использование адресных регистров	**29 Вызов блока SCL** Программирование и вызов блоков SCL; значение функции; переменная OK; механизм EN/ENO	**33 Библиотеки блоков** Организационные блоки; системные функциональные блоки; IEC-функциональные блоки; блоки S5-S7 преобразования; блоки TI-S7 преобразования; блоки с функциями ПИД-управления; коммуникационные блоки
22 Обзор использования STL "Холодный", "теплый" и полный рестарт; режимы STOP, HOLD, сброс памяти; параметризация модулей	**26 Прямой доступ к переменным** Загрузка адреса переменной; сохранение переменных в памяти; сохранение данных при передаче параметров; "переменная" указатель ANY; описание примера фрейма сообщения	**30 SCL-функции** Функции таймера; функции счетчика; математические функции; функции сдвига и циклического сдвига; функции преобразования	**34 Обзор использования STL** Базовые функции; функции цифровой обработки; управление выполнением программы
23 Обработка ошибок Синхронные ошибки; асинхронные ошибки; системная диагностика		**28 Управляющие операторы** Функции преобразования и сравнения; STRING-функции; Date/TOD-функции; численные функции	**35 Обзор SCL-мнемоник и функций** Операторы; команды; вызовы блоков; стандартные функции

КРАТКИЙ ОБЗОР СОДЕРЖАНИЯ ДИСКЕТЫ С ПРИМЕРАМИ ПРОГРАММ

Настоящая книга содержит много иллюстраций, демонстрирующих использование языков программирования STL и SCL. Все представленные в книге программы Вы можете найти на дискете, прилагаемой к книге. Программы расположены в двух библиотеках – STL_BOOK и SCL_BOOK. После разархивирования эти библиотеки занимают на жестком диске приблизительно 2,7 или 1,6 Мбайт (в зависимости от используемой файловой системы на Вашем ПК или PG).
Библиотека STL_BOOK содержит восемь программ, которые иллюстрируют STL-метод представления. Два насыщенных деталями примера показывают программирование функций, функциональных блоков и частных случаев их практического применения (пример: конвейерная линия [Conveyor Example]), а также управление данными (пример фрейма сообщения [Message Frame Example]). Все примеры представлены исходными файлами и содержат описания и комментарии.

Библиотека STL_BOOK

Базовые функции Примеры для STL-представления	**Выполнение программы** Примеры вызова SFC
FB 104 Глава 4: Двоичные логические операции FB 105 Глава 5: Функции памяти FB 106 Глава 6: Функции пересылки FB 107 Глава 7: Функции таймера FB 108 Глава 8: Функции счетчика	FB 120 Глава 20: Основная программа FB 121 Глава 21: Обработка прерываний FB 122 Глава 22: Характеристики перезапуска FB 123 Глава 23: Обработка ошибок
Цифровые (Digital) функции Примеры для STL-представления	**Работа с переменными** Примеры работы с типами и переменными
FB 109 Глава 9: Функции сравнения FB 110 Глава 10: Арифметические функции FB 111 Глава 11: Математические функции FB 112 Глава 12: Функции преобразования FB 113 Глава 13: Функции сдвига FB 114 Глава 14: Логика для типов Word	FB 124 Глава 24: Типы данных FB 125 Глава 25: Косвенная адресация FB 126 Глава 26: Прямой доступ к переменным FB 101 Простые типы данных FB 102 Сложные типы данных FB 103 Типы параметров
Управление ходом программы Примеры для STL-представления	**Пример конвейера** Примеры базовых функций и частные примеры
FB 115 Глава 15: Биты состояния FB 116 Глава 16: Функции перехода FB 117 Глава 17: Главное управляющее реле FB 118 Глава 18: Функции блоков FB 119 Глава 19: Параметры блоков Программирование исходного файла блока (Глава 3)	FC 11 Система управления конвейером FC 12 Управление счетчиком FB 20 Загрузка конвейера FB 21 Лента конвейера FB 22 Подсчет деталей
Пример фрейма сообщения Примеры обработки данных	**Общие примеры**
UDT 51 Структура данных, заголовок UDT 52 Структура данных, фрейм сообщения FB 51 Создание фрейма сообщения FB 52 Сохранение фрейма сообщения FC 61 Управление часами FC 62 Генерация контрольной суммы FC 63 Преобразование данных	FC 41 Мониторинг диапазона FC 42 Определение граничного значения FC 43 Нахождение оптимального решения FC 44 Проверка фронта сигнала DOUBLE WORD FC 45 Преобразование числа формата с плавающей запятой S5 в формат REAL S7 FC 46 Преобразование числа формата REAL S7 в формат с плавающей запятой S5 FC 47 Копирование из области данных (указатель ANY)

Библиотека SCL_BOOK содержит пять программ на языке SCL с использованием SCL-функций. Программы с примерами, аналогичными рассмотренным выше для STL-представления и имеющими те же названия: пример "конвейерная линия" [Conveyor Example] и пример фрейма сообщения [Message Frame Example], из библиотеки SCL_BOOK соответственно предложены в формате SCL-представления. Программа "General Examples" ("Общие примеры") содержит SCL-функции для обработки сложных типов данных, сохранения данных и специально для программистов на SCL – мнемоники для программирования простых STL-функций в SCL-программах.

Библиотека SCL_BOOK

Элементы языка Примеры для SCL-представления (глава 27)	**SCL-функции** Примеры для SCL-представления (глава 30)
FC 271 Пример ограничителя OB 1 Основная программа для примера ограничителя FB 271 Операторы, выражения, присвоение FB 272 Косвенная адресация	FB 301 Функции таймера FB 302 Функции счетчика FB 303 Функции преобразования FB 304 Математические функции FB 305 Сдвиг и ротация
Операторы управления Примеры для SCL-представления (глава 28)	**Работа с переменными** Примеры для SCL-представления (глава 31)
FB 281 Оператор IF FB 282 Оператор CASE FB 283 Оператор FOR FB 284 Оператор WHILE FB 285 Оператор REPEAT	FB 311 Функции преобразования FB 312 Функции сравнения FB 313 Функции для String FB 314 Функции для Date/TOD FB 315 Функции для чисел
Вызов SCL-блоков Примеры для SCL-представления (глава 29)	**Общие примеры**
FC 291 FC с значением функции FC 292 FC без значения функции FB 291 FB блок FB 292 Примеры вызовов FC и FB блоков FC 293 FC блок для примера EN/ENO FB 293 FB блок для примера EN/ENO FB 294 Вызовы для примеров EN/ENO	FC 61 DT_TO_STRING FC 62 DT_TO_DATE FC 63 DT_TO_TOD FB 61 Длина переменной FB 62 Контрольная сумма FB 63 Кольцевой буфер FB 64 FIFO регистр STL функции для программирования на SCL
Пример управления конвейером Примеры базовых функций и частные примеры	**Пример фрейма сообщения** Примеры управления данными
FC 11 Управление конвейером FC 12 Управление счетчиком FB 20 Загрузка конвейера FB 21 Конвейер FB 22 Подсчет деталей	UDT 51 Структура данных, заголовок UDT 52 Структура данных, фрейм сообщения FB 51 Создание фрейма сообщения FB 52 Сохранение фрейма сообщения FC 61 Управление часами

Для того, чтобы опробовать указанные программы в действии, создайте проект в соответствии с конфигурацией Вашего оборудования, после чего скопируйте программу, содержащую таблицу символов, в проект. Теперь Вы можете вызывать программы-примеры, адаптировать их к Вашим целям и тестировать их в интерактивном режиме.

Если у Вас нет полной версии пакета STEP 7 или STEP 7Mini, Вы сможете ознакомиться с программами-примерами, используя прилагаемый к книге компакт-диск с демонстрационной версией пакета STEP 7 (см. последнюю страницу приложения).

Автоматизация с применением STEP 7: схемы применения

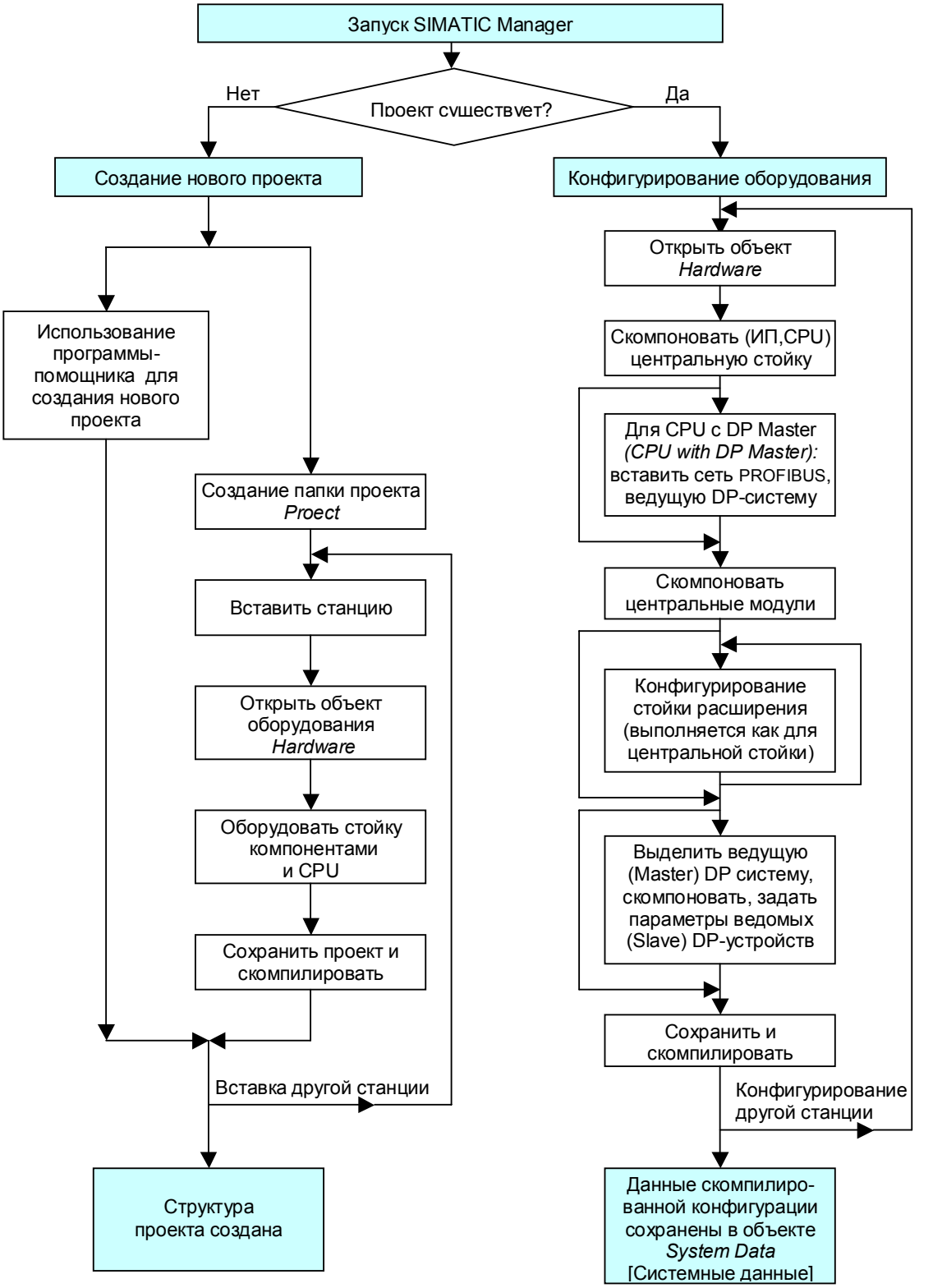
На схемах, представленных на следующих трех страницах, показаны общие процедуры (алгоритмы) использования пакета для программирования в STEP 7.

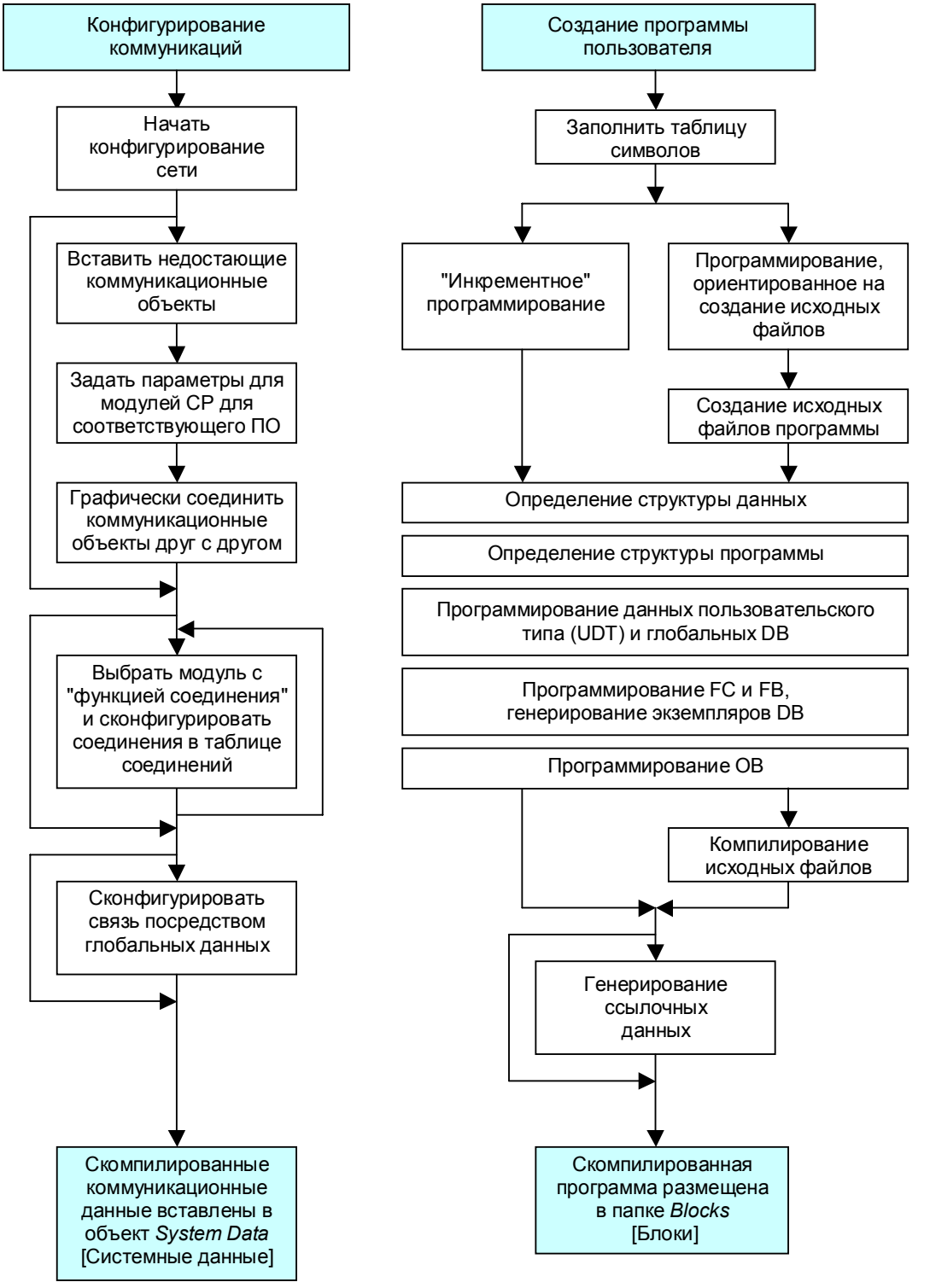
Запустите SIMATIC Manager, создайте новый или откройте существующий проект. Все данные для задачи автоматизации хранятся в форме объектов в проекте. Когда Вы формируете проект, Вы создаете папки (в терминах системы Windows) для группирования данных с помощью установки требуемых станций, по крайней мере с CPU. Кроме того создаются также папки для программ пользователя. Вы можете создать папку для своих программ непосредственно в папке проекта.

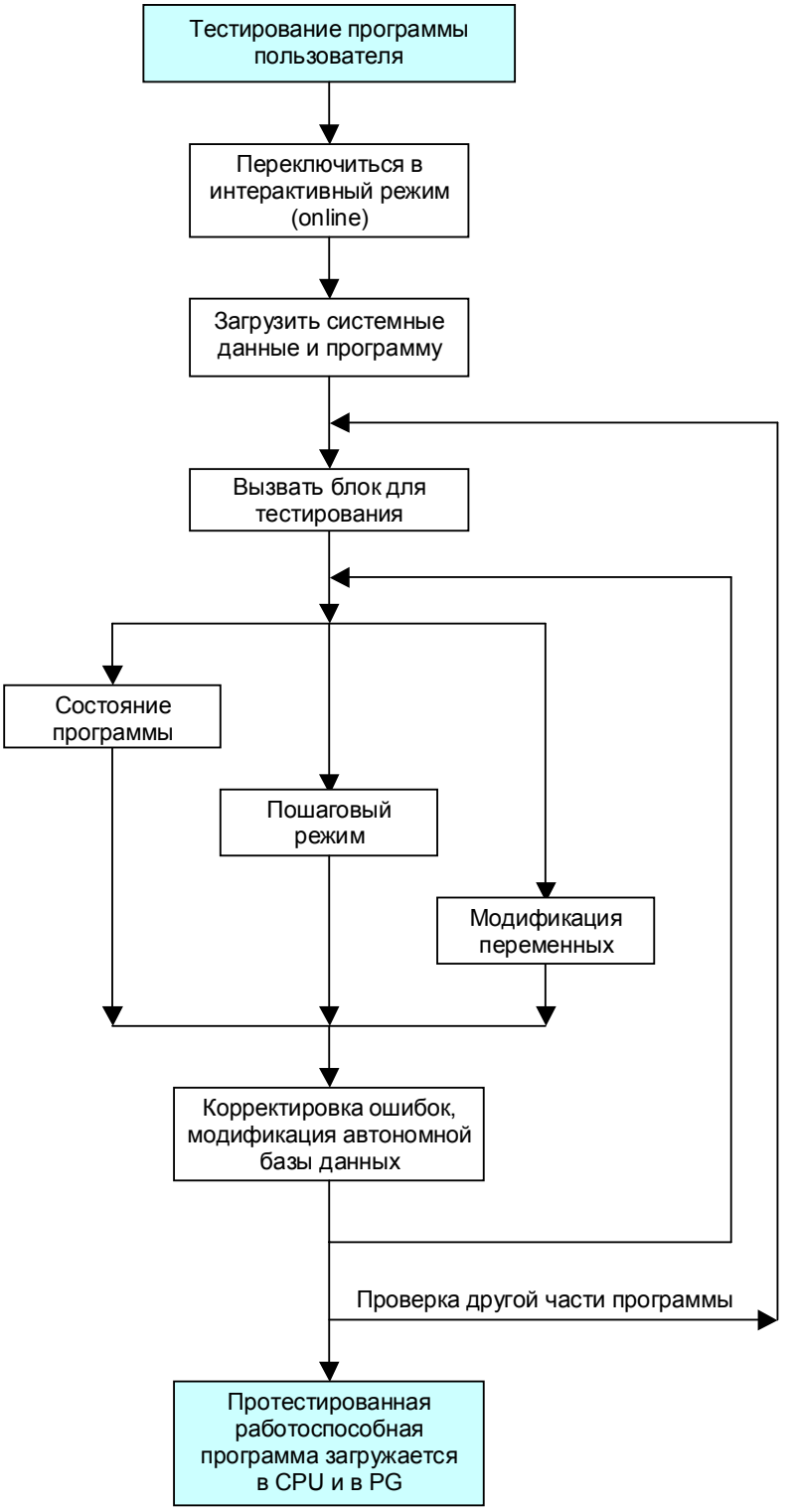
На следующем этапе Вы должны сконфигурировать Ваше оборудование и, если это необходимо, коммуникации. После этого необходимо создать программу и протестировать ее.

Порядок создания данных автоматизации не является строго фиксированным. Необходимо только придерживаться следующих правил:
если Вы желаете обрабатывать объекты (данные), они должны существовать;
если Вы желаете вставить (добавить) объекты в проект, соответствующие папки (путь к объектам) должны быть доступны.

Вы можете прервать разработку проекта в любой момент и продолжить ее вновь с любой точки, вновь запустив SIMATIC Manager.


Запуск SIMATIC Manager
Проект существует?
Нет
Да
Создание нового проекта
Конфигурирование оборудования
Использование программы-помощника для создания нового проекта
Создание папки проекта *Proect*
Вставить станцию
Открыть объект оборудования *Hardware*
Оборудовать стойку компонентами и CPU
Сохранить проект и скомпилировать
Вставка другой станции
Структура проекта создана
Открыть объект *Hardware*
Скомпоновать (ИП,CPU) центральную стойку
Для CPU с DP Master *(CPU with DP Master):* вставить сеть PROFIBUS, ведущую DP-систему
Скомпоновать центральные модули
Конфигурирование стойки расширения (выполняется как для центральной стойки)
Выделить ведущую (Master) DP систему, скомпоновать, задать параметры ведомых (Slave) DP-устройств
Сохранить и скомпилировать
Конфигурирование другой станции
Данные скомпилированной конфигурации сохранены в объекте *System Data* [Системные данные]


Конфигурирование коммуникаций
Начать конфигурирование сети
Вставить недостающие коммуникационные объекты
Задать параметры для модулей CP для соответствующего ПО
Графически соединить коммуникационные объекты друг с другом
Выбрать модуль с "функцией соединения" и сконфигурировать соединения в таблице соединений
Сконфигурировать связь посредством глобальных данных
Скомпилированные коммуникационные данные вставлены в объект *System Data* [Системные данные]
Создание программы пользователя
Заполнить таблицу символов
"Инкрементное" программирование
Программирование, ориентированное на создание исходных файлов
Создание исходных файлов программы
Определение структуры данных
Определение структуры программы
Программирование данных пользовательского типа (UDT) и глобальных DB
Программирование FC и FB, генерирование экземпляров DB
Программирование OB
Компилирование исходных файлов
Генерирование ссылочных данных
Скомпилированная программа размещена в папке *Blocks* [Блоки]


Тестирование программы пользователя
Переключиться в интерактивный режим (online)
Загрузить системные данные и программу
Вызвать блок для тестирования
Состояние программы
Пошаговый режим
Модификация переменных
Корректировка ошибок, модификация автономной базы данных
Проверка другой части программы
Протестированная работоспособная программа загружается в CPU и в PG

СОДЕРЖАНИЕ

Введение

В данной части книги читателю предлагается обзор изделий SIMATIC S7-300/400.

Программируемый контроллер SIMATIC S7-300/400 имеет модульную конструкцию. Модули, из которых составляется требуемая конфигурация контроллера, могут быть центральными (располагаться по соседству с CPU) или распределенными. В системах SIMATIC S7 распределенные входы/выходы (I/O) являются составной частью системы. CPU, имеющий различные области памяти, составляет основу оборудования системы для обработки программ пользователя. Загрузочная память (load memory) целиком содержит пользовательскую программу: части программы, выполняемые в любое заданное время (исполняемый модуль программы), находятся в рабочей памяти (work memory), обеспечивающей малое время доступа к данным, что предопределяет высокую скорость обработки программы.

STEP 7 – это программное обеспечение для программирования S7-300/400. Для организации работы по конфигурированию, программированию и тестированию программной части системы автоматического управления процессами служит утилита SIMATIC Manager. SIMATIC Manager – это приложение, работающее под управлением Windows 95/98/NT и содержащее все функции, необходимые для создания проекта. При необходимости SIMATIC Manager инициирует запуск других утилит, например, для конфигурирования станций, для инициализации модулей или для написания и тестирования программ.

Пользователь должен изложить свое программное решение для автоматизированной системы, используя языки программирования STEP 7. Программа SIMATIC S7 является структурированной программой, что означает, что она состоит из блоков, обладающих определенными функциями, соответствующими их положению в сетевой и иерархической структуре системы. Различные классы приоритетов позволяют располагать в определенном порядке прерывания исполняемой программы пользователя. STEP 7 работает с переменными различных типов, начиная с переменных двоичного типа (BOOL), с переменных численных форматов (INT или REAL) и заканчивая сложными типами, такими как массивы или структуры (комбинации переменных различных типов в форме единой переменной).

Первая глава книги содержит краткий обзор оборудования для программируемых контроллеров S7-300/400. Вторая глава книги содержит краткий обзор программного обеспечения STEP 7 для программирования. Описание строится на основе набора функций для STEP 7 версии 5.1.

Глава 3 "Программа SIMATIC S7" представляет собой введение в курс по наиболее важным элементам S7-программы и показывает способы программирования отдельных блоков программы на языках программирования STL и SCL. Функции и операторы языков STL и SCL описаны в последующих главах книги. Все описания сопровождаются пояснениями с использованием кратких примеров.

1 SIMATIC S7-300/400 программируемый контроллер
Структура программируемого контроллера;
распределенная периферия (I/O);
коммуникации;
адресация модулей;
области данных.

2 Программное обеспечение STEP 7 для программирования
SIMATIC Manager;
обработка проекта;
конфигурирование станций;
конфигурирование сети;
создание программ (таблица символов, редакторы программ);
включение интерактивного режима;
тестирование программы.

3 Программа SIMATIC S7
Обработка программы с классами приоритетов;
блоки программы;
адресация переменных;
программирование блоков с использованием STL и SCL;
переменные и константы;
типы данных (краткий обзор).

1 Программируемый контроллер SIMATIC S7-300/400

1.1 Структура программируемого контроллера

1.1.1 Компоненты

Программируемый контроллер SIMATIC S7-300/400 имеет модульную конструкцию и включает в себя следующие компоненты:

- Стойки (Rack):
  стойки используются для размещения в них модулей и для соединения последних друг с другом.
- Источник питания (PS – "power supply"):
  источник питания обеспечивает внутренние напряжения питания.
- Центральный процессор (CPU – "central processing unit"):
  центральный процессор используется для размещения и обработки программы пользователя.
- Интерфейсные модули (IM – "interface module"):
  интерфейсные модули используются для соединения стоек друг с другом.
- Сигнальные модули (SM – "signal module"):
  сигнальные модули используются для преобразования сигналов, поступающих от процесса, во внутренние сигналы для последующей обработки или в дискретные или аналоговые сигналы для управления приводами.
- Функциональные модули (FM – "function module"):
  функциональные модули не зависят от CPU, используются для выполнения сложных или зависящих от времени процессов.
- Коммуникационные процессоры (CP – "communication processor"):
  коммуникационные процессоры используются для связи с подсетями.
- Подсети:
  подсети используются для связи программируемых контроллеров друг с другом или с другими устройствами.

Программируемый контроллер (или станция) может состоять из нескольких стоек, которые связываются друг с другом посредством шины. Источник питания, CPU и I/O модули (модули SM, FM и CP) включаются в центральную стойку. Если для I/O модулей недостаточно места или необходимо часть или все I/O модули разместить вне центральной стойки, то в таких случаях используют дополнительные стойки – стойки расширения, которые соединяются с центральной стойкой посредством интерфейсных модулей (см. рис. 1). Также возможно подключение к станции распределенных входов/выходов (см. раздел 1.2, "Распределенные I/O").

Для связи модулей друг с другом в стойках служат две шины: шина входов/выходов (I/O или P-шина) и коммуникационная шина (или K-шина).

I/O-шина предназначена для высокоскоростного обмена входными и выходными сигналами, а коммуникационная шина обеспечивает обмен между модулями большими порциями данных. Коммуникационная шина соединяет CPU и интерфейс программатора (MPI) с функциональными модулями и коммуникационными процессорами.

1.1.2 Станция S7-300

Централизованная конфигурация

Программируемый контроллер S7-300 позволяет включить в центральную монтажную стойку до 8 входных/выходных модулей. Если такая однорядная конфигурация контроллера не является достаточной, то возможны два варианта расширения конфигурации при использовании CPU 314 или более мощных процессоров:

- или вариант двухрядной конфигурации, имеющей центральную стойку и одну стойку расширения (при использовании интерфейсных модулей IM 365 и с расстоянием до одного метра между стойками);
- или вариант конфигурации, состоящей максимально из 4 рядов, т.е. кроме центральной стойки, имеющей до 3 стоек расширения (при использовании интерфейсных модулей IM 360 и IM 361 и с расстоянием до десяти метров между стойками).

Вы можете задействовать максимум восемь модулей в стойке. Число модулей может быть ограничено также максимально допустимым током потребления на одну стойку, который составляет 1.2 A (для CPU 312 IFM максимально допустимый ток потребления составляет 0.8 A).

Модули связаны между собой внутренней шиной стойки, обеспечивающей функции P- и K-шин.

Локальный сегмент шины

Особую возможность при конфигурировании предоставляет использование прикладного модуля FM 356 из семейства компьютеров для автоматизации M7-300. Модуль FM-356 позволяет "разбить" интерфейсную шину модулей контроллера, чтобы получить контроль над оставшимися в "отсеченном сегменте шины" модулями для автономного управления ими. В данном случае ограничивающим фактором также являются такие параметры, как число модулей и суммарная потребляемая ими мощность.

Внешние условия для изделий SIMATIC

Модули SIMATIC S7-300 допускают использование в жестких внешних условиях. Они имеют расширенный температурный рабочий диапазон: (-25…+60)°C, повышенную вибрационную и ударную стойкость, соответствующие стандарту IEC 68 часть 2-6; удовлетворяют требованиям по влагостойкости, устойчивости к образованию конденсата и инея согласно IEC 721-3-3 Class 3 K5, также как и требованиям стандарта для ж/д транспорта по EN 50155 (в скором будущем). Остальные характеристики стандартны.

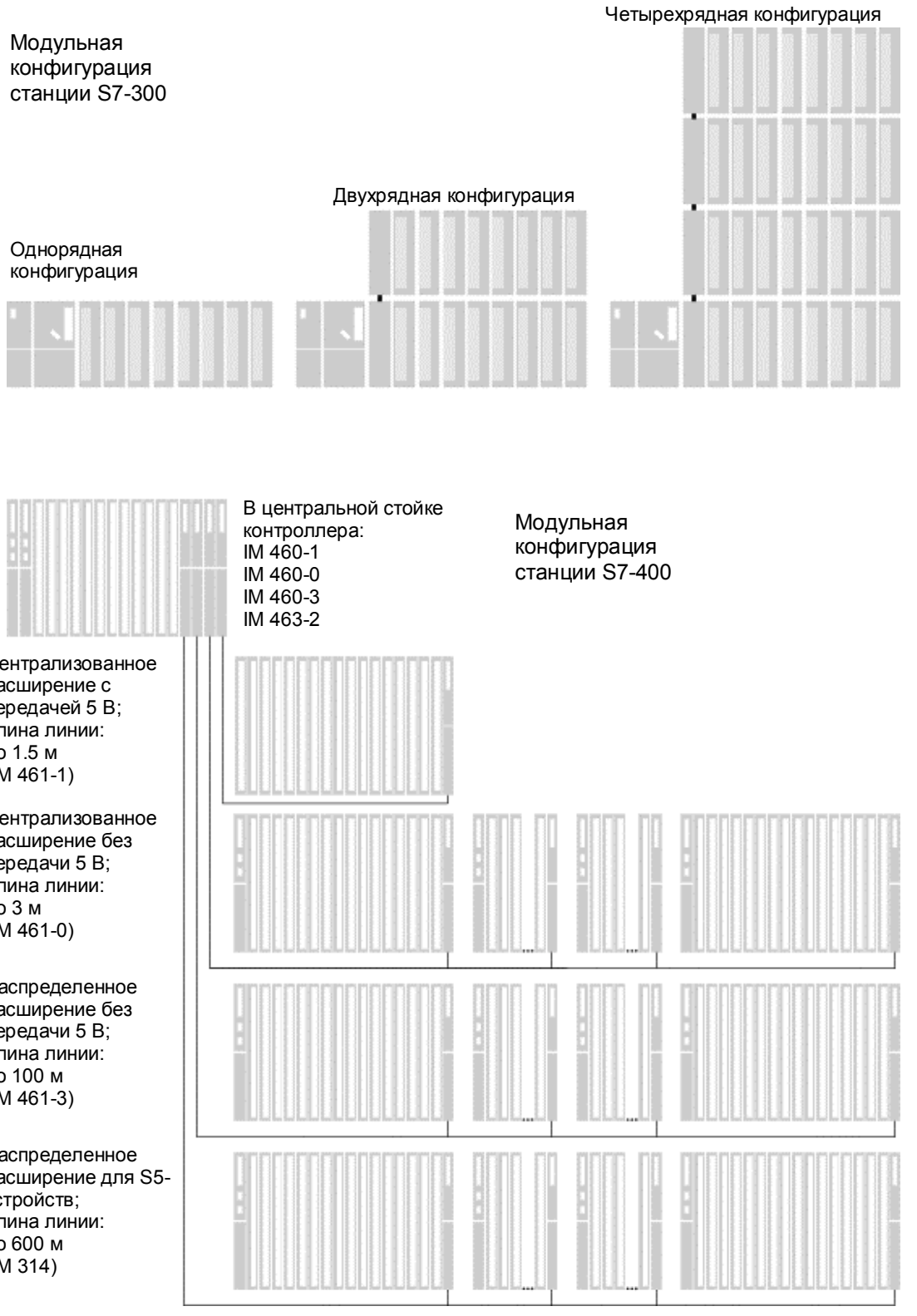


Рис. 1.1 Конфигурация аппаратной части для S7-300/400

1.1.3 Станция S7-400

Централизованная конфигурация

Программируемый контроллер S7-400 имеет следующий состав: центральная монтажная стойка на 18 или 9 слотов (соответственно UR1 или UR2), модуль блока питания и модуль CPU, которые могут сами занимать от одного до нескольких слотов (посадочных мест в монтажной стойке). Интерфейсные модули IM 460-1 и IM 461-1 позволяют использовать одну стойку расширения с подачей в нее от центральной стойки 5-вольтового питающего напряжения с длиной линии до 1.5 метров между стойками. Кроме того, с помощью интерфейсных модулей IM 460-0 и IM 461-0 могут быть использованы от одной до 4 стоек расширения с длиной шины связи с центральной монтажной стойкой до 3 метров. И наконец, с помощью интерфейсных модулей IM 460-3 и IM 461-3 могут быть использованы от одной до 4 стоек расширения с длиной шины связи с центральной монтажной стойкой до 100 метров.

Максимальное количество подсоединяемых к центральной монтажной стойке стоек расширения составляет 21 единицу. Для распознавания стоек расширения необходимо их количество задавать с помощью кодирующего переключателя на приемном интерфейсном модуле.

Внутренняя шина состоит из параллельной P- и последовательной K-шин. Стойки расширения ER1 и ER2 (соответственно на 18 и 9 слотов) предназначены для "простых" сигнальных модулей, которые не могут генерировать аппаратные прерывания, не имеют 24-вольтового питающего напряжения по P-шине, не имеют резервного питания и не имеют связи по K-шине. K-шина используется в стойках UR1, UR2 и CR2 или в случае применения их в качестве центральных монтажных стоек, или в случае применения их в качестве монтажных стоек расширения с номерами от 1 до 6.

Присоединение сегментированной стойки

Особую возможность при конфигурировании предоставляет использование сегментированной монтажной стойки CR2. Сегментированная монтажная стойка CR2 позволяет использовать два центральных процессора с общим для них модулем питания. При этом оба CPU могут сохранять свою функциональную обособленность, так как имеют отдельные P-шины со своими собственными сигнальными модулями, и могут в то же время обмениваться данными посредством K-шины.

Мультипроцессорный режим

В программируемом контроллере S7-400 возможно организовать мультипроцессорный режим с участием нескольких (до четырех единиц) специальных CPU. При этом в такой станции каждый модуль назначается только одному из CPU (соответственно с его адресацией и прерываниями). За более детальной информацией обратитесь к разделам 20.3.6 "Мультипроцессорный режим" и 21.6 "Прерывание мультипроцессорного режима".

Модули SIMATIC S5

Интерфейсный модуль IM 463-2 позволяет подключать к программируемому контроллеру S7-400 устройства расширения SIMATIC S5 (EG 183U, EG 185U, EG 186U, ER 701-2 и ER 701-3), а также позволяет централизованное расширение устройств расширения. Интерфейсный модуль IM 314 в устройствах расширения SIMATIC S5 используется для обеспечения функций связи. Вы можете использовать все аналоговые и дискретные модули, допустимые в этих устройствах расширения. В контроллере S7-400 могут использоваться от одного до четырех интерфейсных модулей IM 463-2; а к каждому из интерфейсных модулей IM 463-2, таким образом, могут быть подключены от одного до четырех устройств расширения S5 в распределенной конфигурации.

Резервирование на основе программного обеспечения

Используя стандартные компоненты SIMATIC S7-300/400, Вы можете создать резервированную на основе программного обеспечения систему с ведущей станцией управления процессом и резервной станцией, которая принимает на себя управление процессом в случае выхода из строя ведущей станции.

Устойчивость к сбоям системы управления с резервированием на основе программного обеспечения подходит для так называемых "медленных процессов", так как переключение управления на резервную станцию может потребовать нескольких секунд, в зависимости от конфигурации программируемых контроллеров. Сигналы, поступающие от процесса, "замораживаются" на время перехода управления к резервной станции. Резервная станция будет продолжать работу по управлению процессом, используя последние корректные данные, полученные ведущей станцией.

Резервирование входных/выходных модулей обеспечивается с помощью распределенной периферии (I/O) (ET 200M с интерфейсным модулем IM 153-3 для резервирования PROFIBUS-DP). В системе управления может быть сконфигурировано заказное (опционное) программное обеспечение для резервирования ("Software Redundancy").

Отказоустойчивый контроллер SIMATIC S7-400H

SIMATIC S7-400H – это отказоустойчивый программируемый контроллер с резервированной конфигурацией, имеющей две центральные стойки, каждая с H CPU и с модулем синхронизации для сравнения данных с волоконно-оптическим кабелем. Оба регулятора работают в режиме "горячего резервирования"; в случае отказа неповрежденный контроллер в одиночку продолжает выполнять функции управления после плавного автоматического переключения на резервный режим работы.

Входы/выходы могут обеспечивать обычный нормальный доступ (одноканальная, односторонняя конфигурация) или расширенный доступ (одноканальная переключаемая конфигурация с ET 200M). Связь осуществляется по простой или по резервной шине.

Программа пользователя точно такая же, как и для не резервированного контроллера; функции резервирования выполняются исключительно аппаратно и не заметны для пользователя. Для конфигурирования системы требуется заказное (опционное) программное обеспечение "S7-400H".

1.1.4 Области памяти CPU

Память пользователя

На рисунке 1.2 показаны области памяти CPU, имеющие значение для Вашей программы. Программа пользователя собственно располагается в двух областях, а именно в загрузочной памяти (load memory) и в рабочей памяти (work memory).

Загрузочная память (load memory) конструктивно может быть частью CPU или может быть в виде встраиваемого отдельного модуля памяти. Вводимая пользователем программа, включая данные конфигурирования, располагается в загрузочной памяти (load memory) и в оперативной памяти.

Рабочая память (work memory) конструктивно является частью CPU и представляет собой быструю RAM-память. В оперативной памяти содержатся релевантные части программы пользователя: собственно код программы и данные пользователя. Здесь "релевантность" означает, что в эту память загружается код, описывающий существующие объекты, но это не предполагает обязательность вызова отдельных блоков этого кода для обработки.

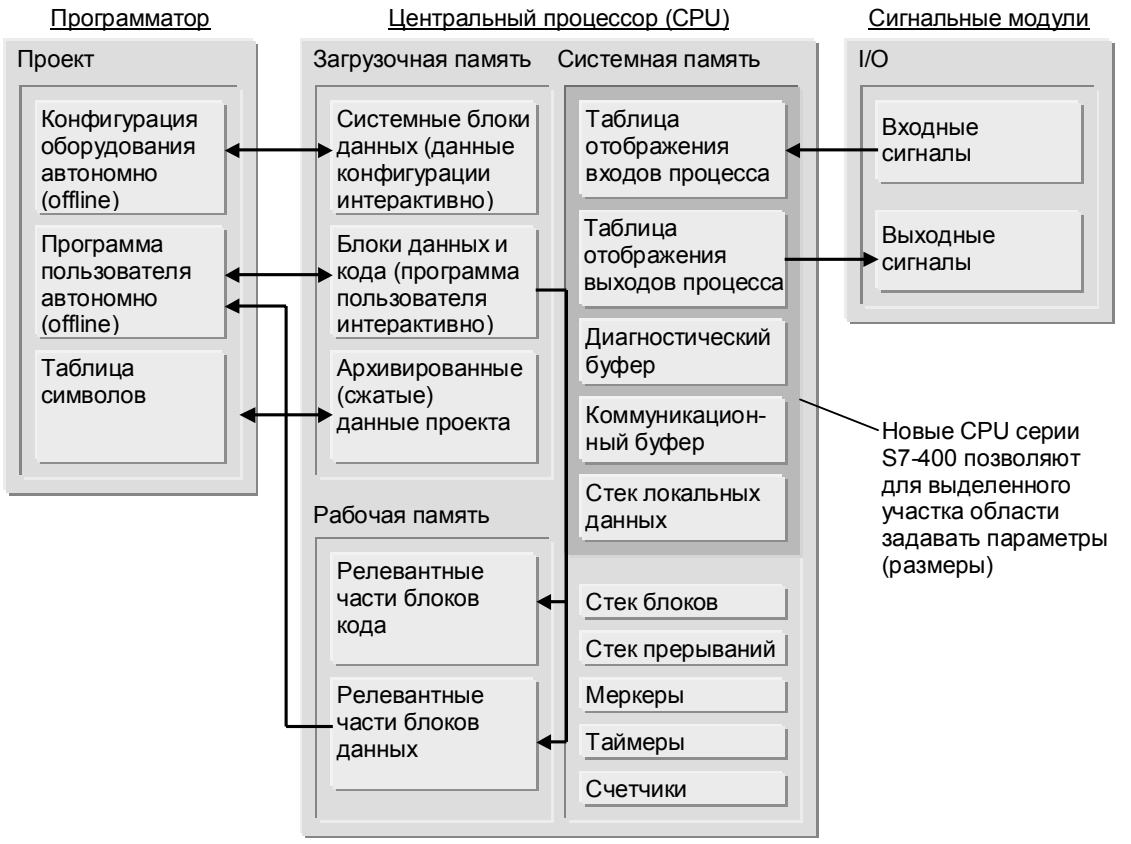


Рис. 1.2 Области памяти CPU

Из программатора программа пользователя целиком, включая данные конфигурации, пересылается в загрузочную память (load memory). Операционная система CPU копирует "релевантные" (см. выше) части программного кода и данных в рабочую память (work memory). Когда программа считывается программатором из CPU, блоки выбираются из загрузочной памяти (load memory) с текущими значениями адресов данных из рабочей памяти (work memory) (для получения более подробной информации см. разделы 2.6.4 "Загрузка программы пользователя в CPU" и 2.6.5 "Обработка блоков").

Если загрузочная память (load memory) построена на основе RAM-памяти, то необходимо использовать дополнительную батарею для резервирования питания, чтобы обеспечивать сохранность программы пользователя в случае отказа штатной системы питания CPU. Если загрузочная память (load memory) выполнена на основе встроенной EEPROM-памяти или внешнего модуля EPROM флэш-памяти, то CPU может использоваться без дополнительного резервирования питания батареей.

Загрузочная память (load memory) в CPU 3xxIFM состоит из RAM и EEPROM компонентов. Вы можете загрузить Вашу программу в RAM-область для ее тестирования, а затем протестированную программу можете посредством команд меню сохранить во внутренней EEPROM-памяти, где она не будет зависеть от отказов блока питания.

Загрузочная память (load memory) в CPU для S7-300 (за исключением CPU 318) состоит из встроенной RAM-памяти, которая может целиком вмещать программу. При этом Вы можете использовать модуль EPROM флэш-памяти в качестве носителя для данных и программ пользователя или в качестве памяти, защищенной от сбоев питания.

В CPU для S7-300 текущие значения из областей памяти пользователя (блоки данных) и системной памяти (меркеры, таймеры, счетчики) могут содержаться в энергонезависимой форме. Таким образом пользователь может сохранять свои данные без применения резервной батареи в условиях возможных перебоев электропитания.

Загрузочная встроенная RAM-память в CPU для S7-400 предназначена для маленьких программ или для модифицирования отдельных блоков. Если полная программа по объему больше, чем встроенная загрузочная память (load memory), то Вам потребуется модуль RAM-памяти для тестирования программы. Вы можете использовать модуль EPROM флэш-памяти в качестве носителя для данных или программы пользователя с бесперебойным питанием.

В новых CPU для S7-400 рабочая память (work memory) может наращиваться с помощью установки дополнительных модулей.

Начиная с STEP 7 V5.1, используя соответствующие CPU для S7-400, Вы можете сохранять все данные проекта в виде архивированного сжатого файла в загрузочной памяти (load memory) CPU (см. раздел 2.2.2 "Управление, перекомпоновка и архивирование").

1.1.5 Модуль памяти

Существуют два типа модулей памяти: модули RAM-памяти и модули EPROM флэш-памяти.

Если необходимо просто увеличить загрузочную память (load memory), то используйте для этого модуль RAM-памяти (например, в CPU для S7-400). Модуль RAM-памяти позволит Вам целиком обновлять программу пользователя в интерактивном режиме. При этом необходимо помнить, что модули RAM-памяти теряют всю записанную в них информацию при вынимании их из слота.

Если Вам необходимо защитить от стирания из-за возможного сбоя в цепях питания пользовательскую программу, включая данные конфигурации и параметры модулей, то используйте модуль EPROM флэш-памяти. При использовании такого модуля памяти загружайте целиком в него программу в автономном режиме, установив модуль EPROM флэш-памяти в программатор. При использовании соответствующего CPU Вы в интерактивном режиме сможете загружать в модуль свою программу пользователя, установив модуль EPROM флэш-памяти непосредственно в такой CPU.

1.1.6 Системная память

Системная память содержит адреса (переменные), к которым Вы обращаетесь в своей программе. Все адреса объединяются в области (адресное пространство), содержащие определенное, зависящее от конкретного CPU, число адресов. Адреса эти могут, например, принадлежать входам, используемым для опроса состояния сигналов от кнопок или конечных переключателей, или выходам, используемым для управления контакторами (реле) или лампами.

Системная память CPU содержит следующие адресные области:

- Входы (I):

  входы формируют "отображение процесса по входам" дискретных входных модулей.

- Выходы (Q):

  выходы формируют "отображение процесса по выходам" дискретных выходных модулей.

- Меркеры (M):

  меркеры хранят информацию, доступную из любой точки программы.

- Таймеры (T):

  таймеры хранят информацию, определяющую параметры времени для функций ожидания и мониторинга.

- Счетчики (C):

  счетчики хранят информацию для функции прямого и обратного счета.

- Временные локальные данные (L)

  временные локальные данные используются в качестве динамических промежуточных буферов при обработке блоков. Временные локальные данные располагаются в L-стеке, который динамически занимается и высвобождается CPU при выполнении программы.

Буквы, заключенные в скобки в выше приведенных пунктах перечисления адресных областей, представляют собой аббревиатуры, используемую в мнемониках программы при различных способах задания адресов. Вы можете также назначать символ для каждой переменной и затем использовать этот символ вместо идентификатора адреса.

В системной памяти также содержатся буферы для коммуникационных заданий и системных сообщений (буфер диагностики). Размеры этих буферов данных, также как и размеры областей хранения отображения процесса по входу и выходу, в новых центральных процессорах для S7-400 может определять пользователь.

1.2 Распределенные I/O (входы/выходы)

Сеть PROFIBUS-DP обеспечивает стандартный интерфейс для передачи преимущественно двоичных данных процесса между "интерфейсным модулем" в центральном программируемом контроллере и приборами полевого уровня. Этот "интерфейсный модуль" называется "ведущим DP-устройством" (DP-master), а приборы полевого уровня называются "ведомыми DP-устройствами" (DP-slave). Распределенные входы/выходы (I/O) относятся к модулям, подключенным посредством PROFIBUS-DP к ведущему модулю PROFIBUS. PROFIBUS-DP совместим со стандартом EN 50170 и является независимым от производителей стандартом для подключения стандартных ведомых DP-устройств.

За дополнительной информацией обратитесь к разделу 1.3.2 "Подсети".

Ведущее DP-устройство и все управляемые им ведомые DP-устройства образуют "систему ведущего DP-устройства" (DP-master system). В одном сегменте сети может быть до 32 станций и максимально до 127 станций может быть в сети всего. Ведущему DP-устройству соответствует определенное число, управляемых им ведомых DP-устройств. Пользователь может также подключать к сети PROFIBUS-DP программатор, также как и, например, устройства для обеспечения человеко-машинного интерфейса, устройства ET 200 или ведомые DP-устройства SIMATIC S5.

1.2.1 Система ведущего DP-устройства

Система с одним ведущим DP-устройством (mono master system)

Сеть PROFIBUS-DP обычно используется как система с одним ведущим DP-устройством ("mono master system"); в такой сети одно ведущее DP-устройство управляет несколькими ведомыми DP-устройствами. В этом режиме ведущее DP-устройство является единственным ведущим устройством на шине, за исключением случаев, когда возможно временное подключение программатора в качестве устройства диагностики и обслуживания. Ведущее DP-устройство и все управляемые им ведомые DP-устройства образуют "систему ведущего DP-устройства" ("DP-master system") (см. ниже рис. 1.3).

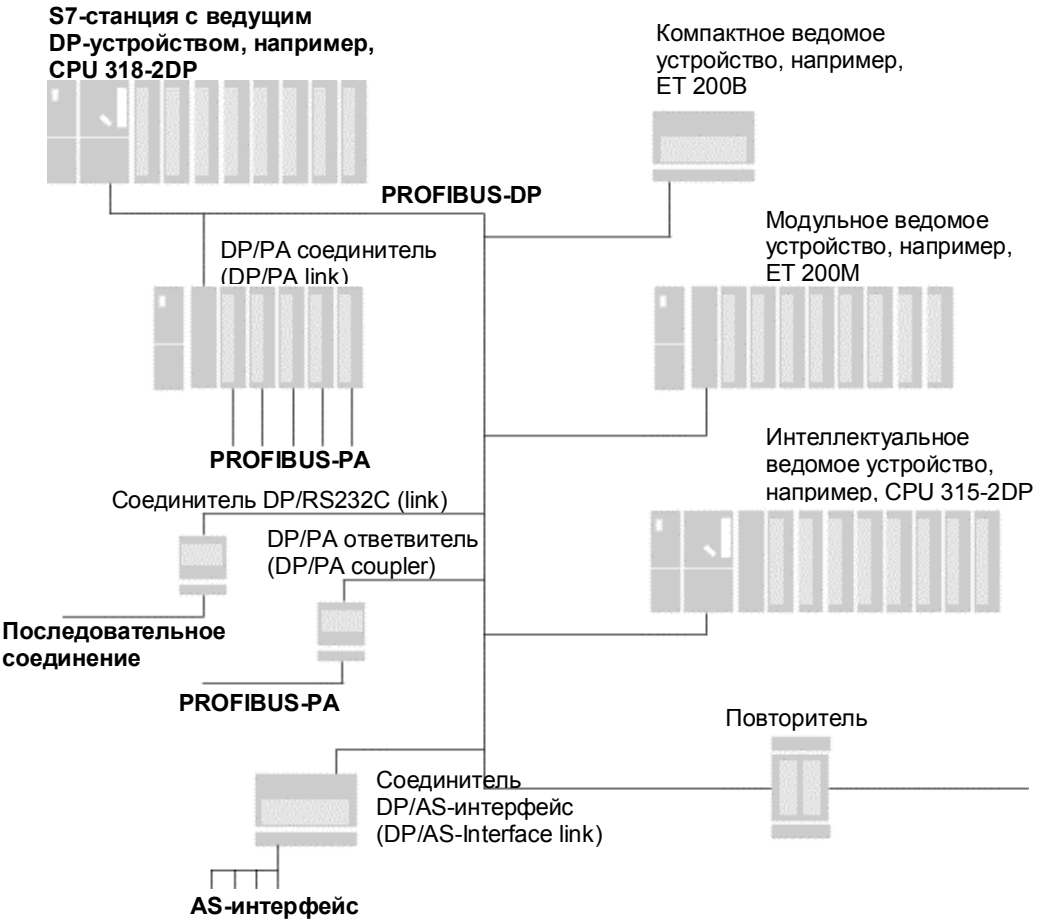


Рис. 1.3 Компоненты системы ведущего DP-устройства (DP-master system)

Система с несколькими ведущими DP-устройствами (multi master system)

Вы можете также установить в одной сети PROFIBUS-DP несколько ведущих DP-устройств ("multi master system"). Тем не менее, это приводит к уменьшению времени отклика в такой сети для каждого сегмента с ведущим DP-устройством; это важно, так как при этом режиме, когда одно ведущее DP-устройство инициализирует "свои" ведомые DP-устройства, другое ведущее DP-устройство не имеет доступа к "своим" ведомым DP-устройствам при попытке их инициализировать и т. д.

Несколько систем ведущих DP-устройств в каждой станции

Вы можете уменьшить время отклика, если в системе ведущего DP-устройства будет уменьшено число ведомых DP-устройств. Так как в одной S7-станции возможно установить несколько ведущих DP-устройств, Вы можете распределить ведомые DP-устройства всей станции между их системами. В мультипроцессорном режиме каждый CPU имеет свою собственную "систему ведущего DP-устройства".

1.2.2 Ведущее DP-устройство (DP Master)

Ведущее DP-устройство (DP Master) является активным узлом сети PROFIBUS. Это ведущее устройство циклически обменивается данными со "своими" ведомыми DP-устройствами. Ведущим DP-устройством может быть:

- CPU с встроенным интерфейсом ведущего DP-устройства или с вставленным интерфейсным модулем (например, CPU 315-2DP, CPU 417).
- Интерфейсный модуль в соединении с CPU (например, IM 467).
- Коммуникационный процессор CP в соединении с CPU (например, CP 342-5, CP 443-5).

Существуют "ведущие DP-устройства 1 класса", предназначенные для обмена данными при обработке процесса, и "ведущие DP-устройства 2 класса", предназначенные для обслуживания и диагностики (например, программаторы).

1.2.3 Ведомые DP-устройства (DP Slaves)

Ведомые DP-устройства (DP Slaves) являются пассивными узлами сети PROFIBUS. В SIMATIC S7 различают следующие ведомые DP-устройства:

- Компактные ведомые устройства, которые ведут себя как отдельные модули по отношению к ведущему DP-устройству.
- Модульные ведомые устройства, состоящие из нескольких (под)модулей.
- Интеллектуальные ведомые устройства, содержащие программу управления для собственных подчиненных модулей.

Компактные ведомые PROFIBUS DP-устройства

Примерами компактных ведомых DP-устройств могут послужить следующие устройства:

ET 200B (в версии для дискретных входных/выходных модулей или аналоговых входных/выходных модулей, имеющий степень защиты IP 20 и максимальную скорость передачи данных, равную 12 Мбит/с);

ET 200C (имеющий конструкцию для условий эксплуатации, отвечающих стандарту IP 66/67, имеющий варианты исполнения для дискретных входных/выходных модулей и аналоговых входных/выходных модулей, имеющий максимальную скорость передачи данных, равную 1,5 Мбит/с или 12 Мбит/с);

ET 200L-SC (дискретно-модульной конструкции с возможностью свободного комбинирования дискретных входных/выходных модулей и аналоговых входных/выходных модулей, имеющий степень защиты IP 20 и максимальную скорость передачи данных, равную 1,5 Мбит/с);

Шинные шлюзы, такие как соединитель DP/AS-I (DP/AS-I Link), ведут себя как компактные ведомые DP-устройства в сети PROFIBUS-DP.

Модульные ведомые PROFIBUS DP-устройства

Примером модульных ведомых DP-устройств может служить устройство ET 200M. Его конструкция аналогична конструкции станции S7-300, имеет профильную шину DIN, модуль блока питания, интерфейсный модуль IM 153 на месте CPU и до 8 сигнальных модулей (SM) или функциональных модулей (FM). Скорость передачи данных составляет от 9,6 кбит/с до 12 Мбит/с).

ET 200M может также иметь в своем составе *активные шинные модули*, если ведущим DP-устройством является станция S7-400. Это означает, что входные/выходные модули S7-300 могут быть добавлены в стойку или удалены из нее в то время, когда она работает с включенным питанием. При этом остающиеся в стойке модули продолжают работать. И при этом при установке модулей в такой стойке не накладывается больше требование плотного их расположения, т.е. не запрещается пропускать слоты при установке модулей в монтажную стойку.

ET 200M может быть также использован с интерфейсным модулем IM 153-3 как ведомое DP-устройство в *резервной шине.* Интерфейсный модуль IM 153-3 имеет два соединителя: один - для подключения к ведущему DP-устройству в ведущей (основной) станции и второй - для подключения к ведущему DP-устройству в резервной станции.

Интеллектуальные ведомые PROFIBUS DP-устройства

Примером интеллектуальных (программируемых) ведомых DP-устройств может послужить станция S7-300, в которой задействован CPU с DP-интерфейсом, который может быть переключен в режим ведомого (slave) устройства (как например, CPU 315-2DP), а также станция S7-300 с коммуникационным процессором CP 342-5 в режиме ведомого (slave) устройства.

Как интеллектуальное ведомое DP-устройство может работать также ET 200X с базовым модулем BM 147/CPU. Он включает в себя базовый модуль и до 7 модулей расширения. В качестве базовых модулей Вы можете использовать "пассивные" базовые модули с дискретными входами или выходами или же Вы можете использовать "интеллектуальный" базовый модуль BM 147/CPU, способный обрабатывать S7-программу пользователя. Модули расширения могут быть представлены дискретными входными/выходными модулями и аналоговыми входными/выходными модулями, а также модулями нагрузок (load feeders), служащими для подключения и защиты любых трехфазных нагрузок для переменного тока напряжением до 400 В мощностью до 5,5 кВт.

Базовые модули обеспечивают передачу данных со скоростями от 9,6 кбит/с до 12 Мбит/с.

1.2.4 Подключение к PROFIBUS-PA

PROFIBUS-PA

PROFIBUS-PA ("Process Automation" [автоматизация процесса]) – это шинная система для организации процессов как в взрывоопасных зонах (или в так называемых Ex-зонах, например, в таких областях, как химическая промышленность), так и в взрывобезопасных зонах (например, в пищевой отрасли).

Протокол для PROFIBUS-PA опирается на стандарт EN 50170, том 2 (PROFIBUS-DPA); способ передачи данных отвечает стандарту IEC 1158-2.

Существуют два возможных способа соединения PROFIBUS-DP и PROFIBUS-PA:

- DP/PA ответвитель (DP/PA coupler), применяемый в случае, когда сеть PROFIBUS-DP может работать со скоростью передачи данных, равной 45,45 кбит/с.
- DP/PA соединитель(DP/PA link), обеспечивающий согласование разных скоростей обмена в PROFIBUS-DP и PROFIBUS-PA.

DP/PA ответвитель (DP/PA coupler)

DP/PA ответвитель (DP/PA coupler) позволяет подключать PA-приборы полевого уровня к сети PROFIBUS-DP. В сети PROFIBUS-DP DP/PA ответвитель имеет статус ведомого DP-устройства со скоростью обмена данными, равной 45,45 кбит/с. К одному DP/PA ответвителю могут быть подключены до 31 PA-прибора полевого уровня. Такая совокупность "полевых" приборов образует сегмент PROFIBUS-PA со скоростью обмена данными, равной 31,25 кбит/с. Взятые все вместе сегменты PROFIBUS-PA образуют шинную систему PROFIBUS-PA общего использования (shared).

DP/PA ответвитель может быть в двух вариантах исполнения:

- DP/PA ответвитель не-Ex версии с выходным током до 400 мА и
- DP/PA ответвитель Ex версии с выходным током до 100 мА.

DP/PA соединитель (DP/PA link)

DP/PA соединитель (DP/PA link) позволяет подключать PA-приборы полевого уровня к сети PROFIBUS-DP и обеспечивать скорость обмена данными от 9,6 кбит/с до 12 Мбит/с. DP/PA соединитель имеет в своем составе интерфейсный модуль IM 157 и до 5 единиц DP/PA-ответвителей, связанных между собой посредством шинных соединителей (коннекторов) SIMATIC S7. Шинная система, состоящая из сегментов PROFIBUS-PA, образует ведомое PROFIBUS-PD устройство. К одному DP/PA соединителю Вы можете подключить до 31 PA-прибора полевого уровня.

SIMATIC DPM

SIMATIC DPM (Process Device Manager [менеджер настройки приборов автоматики], ранее: "SIPROM") – это независимая от поставщика утилита, служащая для задания параметров, отладки, запуска и диагностики интеллектуальных приборов полевого уровня с возможностью работы с протоколами PROFIBUS-PA или HART (Highway Addressable Remote Transducers [протокол обмена с удаленными адресуемыми датчиками-преобразователями]). Для задания параметров датчикам-преобразователям используется программное средство DDL (Device Description Language [язык параметризации приборов]).

Вы можете работать с SIMATIC DPM как в "автономном" режиме, предназначенном для работы под Windows 9x/NT, или как с составной частью системы STEP 7.

1.2.5 Подключение к AS-интерфейсу

AS-интерфейс

AS-интерфейс ("Actuator-Sensor Interface" ("AS-i") [интерфейс привод-датчик]) – это сетевая система для обмена данными с оборудованием процесса нижнего уровня в системе управления. Ведущее устройство AS-i может управлять группой, включающей до 31 единиц ведомых устройств AS-i. Управление обеспечивается по двухпроводной AS-i-линии, по которой передаются как питающее напряжение, так и информационные сигналы. Ведомыми устройствами AS-i могут быть приводы или датчики с шинной организацией или AS-i модули, к которым подключено до 8 двоичных ("normal" - "нормальных") датчиков или приводов.

Сегмент AS-i может иметь длину до 100 м, однако длина сегмента может быть увеличена вдвое при применении повторителя (при этом ведомые устройства AS-i и источники питания должны присутствовать на обоих концах) или при применении расширителя (при этом ведомые устройства AS-i и источник питания должны быть только на линии, идущей от ведущего устройства AS-i).

Ведущее устройство AS-i (AS-i master)

Ведущее устройство AS-i (AS-i master) обновляет свои данные и данные всех подключенных к нему ведомых устройств AS-i на интервале времени, не превышающем 5 мс. Вы можете подключать AS-i шину непосредственно к SIMATIC S7 с помощью коммуникационного процессора CP 342-2 или к сети PROFIBUS-DP с помощью использования DP/AS-интерфейсного соединителя (см. рис. 1.4).

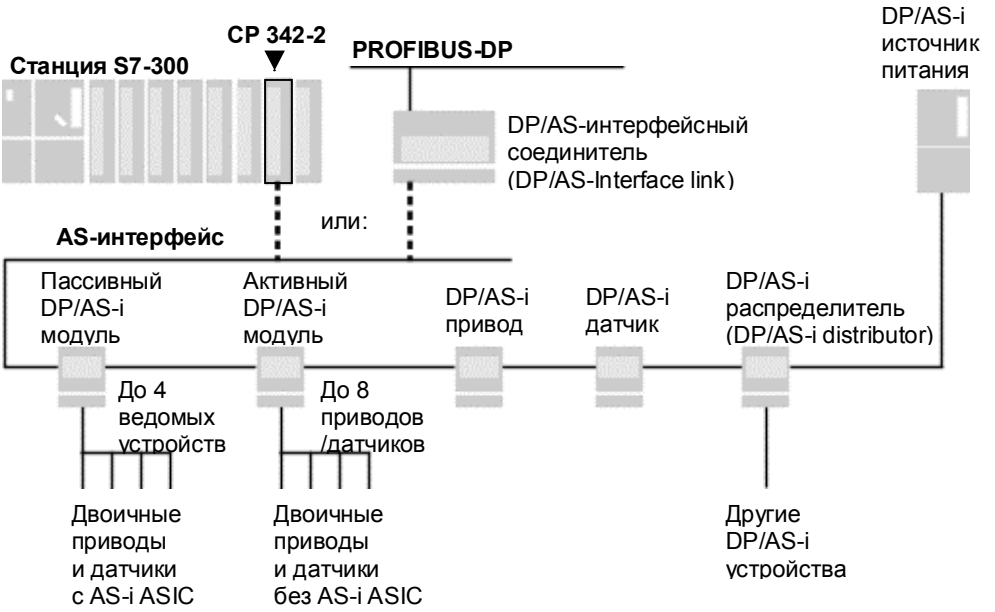


Рис. 1.4 Подключение к SIMATIC S7 шинной системы AS-i

Коммуникационный процессор **CP 342-2** в качестве ведущего устройства AS-i может быть использован в станции S7-300 или в станции ET 200M. Он поддерживает два рабочих режима:

В *стандартном режиме* CP 342-2 ведет себя как входной/выходной модуль. Он занимает 16 входных байтов и 16 выходных байтов в аналоговом адресном пространстве (начиная с 256). Ведомые устройства AS-i параметризуются данными в CP, заданными по умолчанию.

В *расширенном режиме* реализуется полный набор функциональных возможностей ведущего устройства AS-i. Если используется блок FC, то вызовы ведущего устройства могут выполняться из программы пользователя в добавление к возможностям стандартного режима (передача параметров во время работы, проверка заданной/фактической конфигурации, тестирование и диагностика).

DP/AS-интерфейсный соединитель (DP/AS-Interface link) обеспечивает подключение AS-i приводов и AS-i датчиков к сети PROFIBUS-DP. В сети PROFIBUS-DP соединитель имеет статус модульного ведомого DP-устройства, а в сети AS-интерфейс он имеет статус ведущего AS-i устройства, которое может контролировать до 31 ведомого AS-i устройства. При максимальном числе ведомых AS-i устройств DP/AS-интерфейсный соединитель занимает 16 входных байтов и 16 выходных байтов. Скорость передачи - до 12 Мбит/с.

DP/AS-интерфейсный соединитель может выполняться в двух вариантах: для жестких условий эксплуатации (версия 65) со степенью защиты IP 66/67 и (версия 20) со степенью защиты IP 20 с возможностью установки дополнительного командного интерфейса, при котором и для входов и для выходов диапазон адресов возрастает до 20 байт.

1.2.6 Подключение к последовательному интерфейсу

Соединитель **PROFIBUS-DP/RS 232C** (**PROFIBUS-DP/RS 232C link)** является конвертором для соединения интерфейса RS 232C (V.24) и PROFIBUS-DP. Посредством соединителя DP/RS 232C устройства с интерфейсом RS 232C могут быть подключены к PROFIBUS-DP. Соединитель DP/RS 232C поддерживает протоколы 3964R и ASCII.

Соединитель DP/RS 232C обеспечивает подключение приборов способом "точка к точке". Данные передаются с сохранением консистентности в обоих направлениях. В фрейме передается до 224 байт данных пользователя.

Скорость передачи данных по PROFIBUS-DP достигает 12 Мбит/сек; RS 232C обеспечивает скорость передачи данных до 38,4 кбит/с без контроля по четности, с проверкой на четность или нечетность, 8 битов данных плюс 1 стоп-бит.

1.3 Коммуникации (Communications)

Коммуникации обеспечивают обмен данными между программируемыми модулями - это встроенный компонент SIMATIC S7. Почти все коммуникационные функции управляются операционной системой. Обмен данными может быть организован без какого-либо дополнительного оборудования посредством только одного соединительного кабеля между двумя CPU. При использовании модулей CP можно создавать мощные сети и с легкостью подключать к ним системы сторонних (кроме SIEMENS) производителей оборудования.

SIMATIC NET - более широкое понятие, включающее в себя понятие коммуникаций SIMATIC. SIMATIC NET представляет собой информационный обмен между программируемыми контроллерами, а также между программируемыми контроллерами и устройствами HMI (человеко-машинный интерфейс). С SIMATIC могут быть реализованы различные варианты функций связи в зависимости от поставленной задачи.

1.3.1 Введение

На рисунке 1.5 показаны наиболее важные объекты связи. Может возникнуть задача реализации обмена данными между станциями SIMATIC или аппаратами сторонних (отличных от SIEMENS) производителей. В этом случае необходимы модули с функцией связи. С помощью SIMATIC S7 все CPU обеспечиваются MPI интерфейсом, с помощью которого они могут связываться друг с другом. Кроме того, для связи могут быть применены коммуникационные процессоры (CP), выполняющие обмен данными с высокой пропускной способностью и с различными протоколами обмена.

Модули могут быть связаны сетью. Сеть - это аппаратное соединение между узлами связи (коммуникационными узлами).

Обмен данными происходит посредством "соединения" в соответствии со специальным планом обработки данных ("служба обмена"), который основывается на специальной процедуре ("протокол"). Например, S7-соединение является стандартом для S7-модулей с функциями связи.

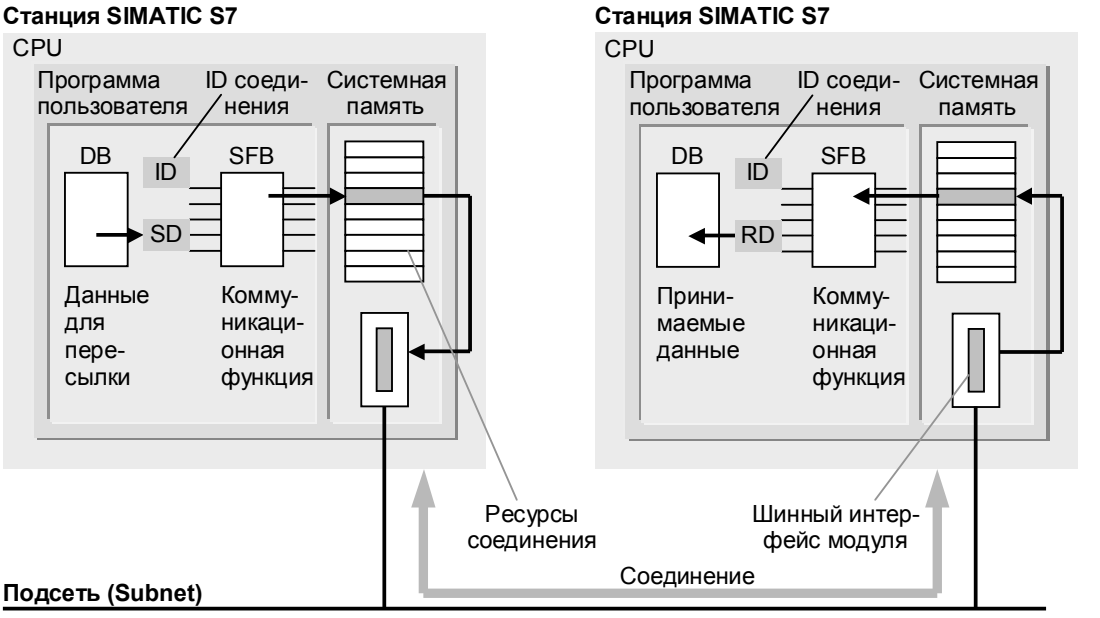


Рис. 1.5 Обмен данными между двумя станциями SIMATIC S7

Сеть

Сеть - это соединение между несколькими устройствами с целью их связи друг с другом. Она состоит из одной или нескольких идентичных или разных подсетей, связанных друг с другом.

Подсеть

В подсети все коммуникационные узлы связаны с помощью аппаратных соединений, обладающих одинаковыми физическими характеристиками и параметрами передачи, такими как скорость передачи; кроме того, обмен данными в подсети происходит в соответствии с единой процедурой передачи данных. В системе SIMATIC применяются несколько типов подсетей: MPI, PROFIBUS, Industrial Ethernet и PTP ("point-to-point" [соединение "точка к точке"]).

Служба обмена (communications service)

Служба обмена (communications service) определяет, как происходит обмен данными между коммуникационными узлами, и как эти данные обрабатываются. Служба обмена базируется на протоколе обмена, который помимо всего прочего описывает процедуру координации работы между коммуникационными узлами.

В SIMATIC различают следующие разновидности организации службы обмена: через функции S7, PROFIBUS-DP, PROFIBUS-FMS, PROFIBUS-FDL (SDA), ISO transport, ISO-on-TSP и связь через глобальные данные.

Соединение (connection)

Соединение определяет коммуникационные отношения между двумя узлами связи (коммуникационными узлами). Это есть логическое назначение двух узлов для выполнения специфических коммуникационных функций (службы обмена) и, кроме того, содержит специальные характеристики, такие как тип соединения (динамический, статический) и каким образом оно устанавливается.

В SIMATIC различают следующие разновидности соединений: S7-соединение, S7-соединение (отказоустойчивое), "point-to-point" [соединение "точка к точке"], FMS- и FDL-соединение, "ISO transport"-соединение, "ISO-on-TSP"- и TSP-соединение, UDP-соединение и E-mail-соединение.

Коммуникационные функции (communications functions)

Коммуникационные функции играют роль интерфейса между программой пользователя и службой обмена подсети. Используемые для внутренних соединений в SIMATIC S7 коммуникационные функции встроены в операционную систему CPU и вызываются с помощью системных блоков. Загружаемые блоки позволяют создавать соединение с устройствами сторонних производителей (кроме Siemens) с помощью коммуникационных процессоров.

Краткий обзор коммуникационных объектов

В таблице 1.1 показано соответствие между подсетями, службами обмена данными и модулями с функцией связи.

1.3.2 Подсети

Подсети - это часть средств связи с одинаковыми физическими характеристиками и одинаковой процедурой обработки данных. Подсети являются центральными объектами в системе связи для утилиты SIMATIC Manager.

Подсети отличаются своими рабочими характеристиками:

- MPI
  экономичный способ создания сетей для небольшого количества устройств SIMATIC с обменом малыми количествами данных.
- PROFIBUS
  высокоскоростной обмен малыми и средними объемами данных; используется прежде всего для работы с системами распределенных входов/выходов.
- Industrial Ethernet
  связь между компьютерами и PLC для высокоскоростного обмена большими объемами данных.
- PTP ("Точка к точке")
  последовательная связь между двумя коммуникационными партнерами по специальным протоколам.

Таблица 1.1 Коммуникационные объекты

Подсеть	Модули	Служба обмена данными	Конфигурация, интерфейс
MPI	Все CPU	Связь через глобальные данные (GD)	GD-таблица
		Связь между станциями посредством SFC	Вызовы SFC
		Связь посредством SFB (только при активной S7-400)	Таблица соединений, вызовы SFB
PROFIBUS	CPU с ведущим DP-устройством	PROFIBUS-DP (ведущее или ведомое устройство)	Конфигурация оборудования, входы/ выходы, вызовы SFC
		Связь внутри станции посредством SFC	Вызовы SFC
	IM 467	PROFIBUS-DP (ведущее или ведомое устройство)	Конфигурация оборудования, входы/ выходы, вызовы SFC
		Связь внутри станции посредством SFC	Вызовы SFC
	CP 342-5 CP 443-5 Extended (расширенный)	PROFIBUS-FDL, PROFIBUS-DP (ведущее или ведомое устройство)	NCM, таблица соединений, SEND / RECEIVE
		Связь внутри станции посредством SFC	Вызовы SFC
		Связь посредством SFB (только при активной S7-400)	Таблица соединений, вызовы SFB
	CP 343-5 CP 443-5 Basic (базовый)	PROFIBUS-FMS, PROFIBUS-FDL	NCM, FMS-интерфейс таблица соединений, SEND / RECEIVE
		Связь внутри станции посредством SFC	Вызовы SFC
		Связь посредством SFB (только при активной S7-400)	Таблица соединений, вызовы SFB
Industrial Ethernet	CP 343-1 CP 443-1	Транспортные протоколы ISO и TCP / IP	NCM, таблица соединений, SEND / RECEIVE
		Связь посредством SFB (только при активной S7-400)	Таблица соединений, вызовы SFB
	CP 343-1 IT CP 443-1 IT	Транспортные протоколы ISO и TCP / IP IT-связь	NCM, таблица соединений, SEND / RECEIVE
		Связь посредством SFB (только при активной S7-400)	Таблица соединений, вызовы SFB

NCM - программное обеспечение для конфигурирования CP; NCM может применяться для PROFIBUS и для Industrial Ethernet.

Посредством STEP 7 V.5 Вы можете использовать программатор для получения доступа к станциям SIMATIC S7 с помощью подсетей, чтобы, например, задать параметры или изменить программу. Шлюзы (переходы) между подсетями должны быть расположены в станции S7 с возможностью программирования.

MPI

Каждый CPU имеет интерфейс для многоточечного подключения ("multipoint interface", MPI ["многоточечный интерфейс"]). Он позволяет создать подсеть для обмена данными между CPU, PG, устройствами HMI (человеко-машинный интерфейс) согласно оригинальному протоколу обмена Siemens.

Линии передачи MPI могут иметь два типа исполнения: экранированный кабель "витая пара" или пластмассовый оптико-волоконный кабель. Длина кабеля в шинном сегменте может достигать 50 м. При этом максимальная длина может быть увеличена до 1100 м в случае применения повторителей RS485 или может даже превышать 100 км в случае применения модулей оптической связи (optical link modul). Скорость передачи данных обычно составляет 187,5 кбит/с.

Максимальное число узлов составляет 32 единицы. Каждый узел имеет доступ к шине в течение определенного отрезка времени и может в это время посылать фреймы данных. По прошествии этого промежутка времени узел передает право доступа к шине следующему узлу (процедура доступа "token passing" [передача маркера или "токена"]).

Посредством сети MPI может быть организован обмен данными между CPU с помощью установления одного из следующих типов связи: связи через глобальные данные, связи между станциями посредством SFC или связи посредством SFB. При этом не требуется применять дополнительные модули.

PROFIBUS

PROFIBUS ("**PRO**cess **FI**eld**BUS**") используется как "шина полевого уровня для автоматизации". PROFIBUS является общим стандартом, совместимым с EN 50170, для связывания в единую сеть устройств полевого уровня.

Линии передачи PROFIBUS могут иметь следующие типы исполнения: экранированный кабель "витая пара" и стеклянный или пластмассовый оптико-волоконный кабель. Максимальная длина кабеля в шинном сегменте зависит от скорости передачи данных; она может достигать 100 м при наибольшей скорости передачи (12 Мбит/с) и может достигать 1000 м при наименьшей скорости передачи (9,6 кбит/с). Длина сети может наращиваться в случае применения повторителей или модулей оптической связи (optical link modul).

Максимальное число узлов составляет 127 единицы. Различают активные и пассивные узлы. Активные узлы имеют доступ к шине в течение определенного отрезка времени и могут в это время посылать фреймы данных. По прошествии этого промежутка времени активный узел передает право доступа к шине следующему узлу (процедура доступа "token passing" [передача "токена"]). Если пассивные узлы (slaves) были назначены активному узлу (master), последний будет выполнять обмен данными с назначенными ему пассивными узлами, пока имеет доступ к шине. Пассивные узлы не получают доступа к шине.

Вы можете осуществлять связь с распределенной периферией посредством сети PROFIBUS; при этом используется соответствующая служба обмена PROFIBUS-DP. Вы можете использовать или CPU со встроенным или вставляемым ведущим DP-устройством или использовать подходящий коммуникационный процессор. В сетях PROFIBUS можно также использовать связь внутри станции посредством SFC или связь посредством SFB.

При использовании соответствующих CP возможен обмен данными посредством служб PROFIBUS-FMS и PROFIBUS-FDL. Как интерфейс для программы пользователя используются загружаемые блоки (FMS-интерфейс или SEND/RESEIVE-интерфейс).

Industrial Ethernet

Industrial Ethernet - это подсеть для обмена данными между компьютерами и программируемыми контроллерами преимущественно в промышленности в соответствии с международным стандартом IEEE 802.3.

Физически линии передачи Industrial Ethernet могут быть в виде коаксиального кабеля с двойным экранированием, в виде кабеля "витая пара" ("industrial") или в виде стеклянного оптико-волоконного кабеля. Длина электрокабеля в сети может достигать 1,5 км, тогда как длина кабеля оптической связи достигает 4,5 км. Скорость передачи данных составляет 10 Мбит/с.

Максимальное число узлов сети Industrial Ethernet может превышать 1000 единиц. Каждый узел, получающий доступ к шине, прежде всего проверяет, не посылает ли данные в это же время другой узел. Если другой узел использует в текущий момент шину, то узел, получающий доступ к шине, ожидает в течение случайным образом выбранного промежутка времени, после чего совершает новую попытку доступа к шине (процедура доступа "CSMA/CD"). Все узлы сети имеют равные права доступа.

Посредством сети Industrial Ethernet может быть также организован обмен данными с помощью установления одного из следующих типов связи: связи через S7-функции или связи посредством SFB. Если использовать для сети Industrial Ethernet соответствующие CP, то тогда есть возможность использовать связь ISO transport или ISO-on-TCP, а также использовать интерфейс SEND/RESEIVE.

Point-to-point

Интерфейс для подключения "точка к точке" ("Point-to-point", PTP) позволяет создать подсеть для обмена данными последовательной связи. Соединение "point-to-point" легко конфигурируется и обеспечивает управление как подсеть с помощью SIMATIC Manager.

Интерфейсные разъемы соединяются посредством электрокабеля. В качестве интерфейсов могут использоваться RS 232C (V.24), 20мА (TTY) и RS 422/485. Скорость передачи данных для интерфейса 20 мА составляет от 300 бит/с до 19,2 кбит/с, а для интерфейсов RS 232C и RS 422/485 - 76,8 кбит/с. Максимальная длина кабеля зависит от физического интерфейса и от скорости передачи данных; она может достигать 10 м для RS 232C, 1000 м для интерфейса 20 мА при скорости передачи 9,6 кбит/с и 1200 м для интерфейса RS 422/485 при скорости передачи 19,2 кбит/с.

3964 (R), RK 512, драйверы для принтеров и ASCII драйвер могут использоваться как протоколы (процедуры), а также последние версии пользовательских протоколов. Отдельные приложения могут потребовать использования особых драйверов.

AS-интерфейс

AS-интерфейс ("AS-Interface", AS-i) позволяет создать подсеть для обмена данными в соответствии со спецификацией IEC TG 178 для AS-интерфейса с соответствующим образом сконструированными двоичными датчиками и приводами. AS-интерфейс не рассматривается как подсеть в SIMATIC Manager; только ведущее устройство AS-i (AS-I master) может быть сконфигурировано процедурами конфигурирования аппаратной части и сети.

Линии передачи AS-Interface представляют собой неэкранированный кабель "витая пара", по которому одновременно обеспечиваются передача данных и электропитание датчиков и приводов (такая сеть требует наличия отдельного блока питания). Максимальная длина кабеля в случае применения повторителей может достигать 300 м. Скорость передачи при этом составляет 167 кбит/с.

Ведущее устройство AS-I (master) может управлять максимум 31 ведомым устройством (slave) в цикле сканирования и обеспечивая определенное время отклика.

1.3.3 Службы обмена (communications services)

Обменом данными в подсетях управляют так называемые службы обмена, тип которых определяется типом соединения. Эти службы используются преимущественно для целей, изложенных ниже:

S7-функции - это главная служба обмена в SIMATIC. S7-функции интегрированы в операционную систему CPU, и обеспечивают связь (коммуникации) между центральными процессорами, устройствами HMI и программаторами.

Ниже представлен краткий обзор их функций:

- Функции для программатора (PG):
  тестирование, запуск и сервисные функции; в PG они используются, например, для выполнения функции мониторинга переменных "monitor variables" или для чтения буфера диагностики или для запуска программ пользователя.
- Функции для человеко-машинного интерфейса (HMI):
  используется подключенными панелями оператора (OP), например, для выполнения функции чтения/записи переменных.
- SFB-коммуникации (SFB-communications):
  управляемые событиями функции для обмена большими объемами данных; запускаются вызовом SFB в программе пользователя с функциями модификации и мониторинга; статические, для сконфигурированных соединений.
- SFC-коммуникации (SFC-communications):
  управляемые событиями функции для обмена данными объемом до 76 байт за передачу; запускаются вызовом SFC в программе пользователя с функциями модификации и мониторинга; динамические, для несконфигурированных соединений.

S7-функции могут выполняться в подсетях MPI, PROFIBUS и Industrial Ethernet.

Связь через глобальные данные (Global data communications) позволяет осуществлять обмен небольшими объемами данных между несколькими CPU без дополнительного усложнения программы пользователя. Передача данных может выполняться циклически или запускаться событиями.

Связь через глобальные данные как процедура носит характер "вещания" (распространения данных); получение данных не квитируется. Состояние соединения подтверждается.

Связь через глобальные данные возможна только с MPI-шиной и K-шиной.

С **PROFIBUS-DP** осуществляется обмен данными между ведущим и ведомыми устройствами через распределенную периферию. Связь имеет "прозрачный режим" и отвечает стандарту EN 50170 том 2. С помощью данной службы обмена может быть организован доступ к ведомым устройствам, отвечающим стандартам SIMATIC S7 и прочим стандартам в подсетях PROFIBUS.

С **PROFIBUS-FMS** (Fieldbus Message Specification ["Спецификация сообщений в шине полевого уровня"]) осуществляется передача структурированных переменных (FMS-переменных) в соответствии со стандартом EN 50170 том 2. Данные коммуникации осуществляются исключительно для статических соединений в подсетях PROFIBUS.

С **PROFIBUS-FDL** (Fieldbus Data Link ["Связь через данные в шине полевого уровня"]) осуществляется передача данных с функцией SDA (Send Data with Acknowledge ["Передача данных с квитированием"]) в соответствии со стандартом EN 50170 том 2. Данные коммуникации осуществляются для статических соединений. В подсетях PROFIBUS данная служба обмена обеспечивает, например, обмен данными с контроллером SIMATIC S5.

С **ISO transport** осуществляется обмен данными в соответствии со стандартом ISO 8073 Class 4. Данные коммуникации осуществляются для статических соединений. С помощью ISO transport может быть организован, например, обмен данными с контроллером SIMATIC S5 в подсетях Industrial Ethernet.

Служба обмена **ISO-on-TSP** соответствует стандарту TCP/IP с расширением RFC 1006. Данные коммуникации осуществляются для статических соединений в подсетях Industrial Ethernet.

1.3.4 Соединения (connections)

Соединения могут быть статическими или динамическими - это зависит от выбранной службы обмена данными. Динамические соединения не конфигурируются; их установление или ликвидация определяются событиями ("Communications via non-configured connections" - "коммуникации посредством несконфигурированных соединений"). Может быть установлено только одно несконфигурированное соединение с коммуникационным партнером.

Статические соединения конфигурируются с помощью таблицы соединений (connection table). Они устанавливаются при запуске программы и остаются на все время выполнения программы ("Communications via configured connections"- "коммуникации посредством сконфигурированных соединений"). Может быть установлено несколько сконфигурированных соединений параллельно с одним коммуникационным партнером. Вы должны выбрать "Connection type" ("Тип соединения") для выбора требуемой службы обмена при конфигурировании сети (см. раздел 2.4 "Конфигурирование сети").

Вам не нужно конфигурировать соединения с помощью утилиты конфигурирования сети для служб обмена посредством глобальных данных (GD) и PROFIBUS-DP или для SFC-коммуникаций (SFC-communications) в случае обмена через S7-функции. Для обмена через GD Вы должны определить коммуникационных партнеров в таблице GD; в случае PROFIBUS-DP или SFC-коммуникаций партнеры определяются посредством адресации узлов.

Ресурсы соединения (Connection resources)

Каждое соединение требует от коммуникационного партнера-участника определенных ресурсов для конечного пункта соединения или "транзитного" пункта в модуле CP. Если, например, функции S7 выполняются посредством MPI-интерфейса CPU, соединения назначаются в CPU; такие же функции посредством MPI-интерфейса CP занимают (используют) одно соединение в CP и одно соединение в CPU.

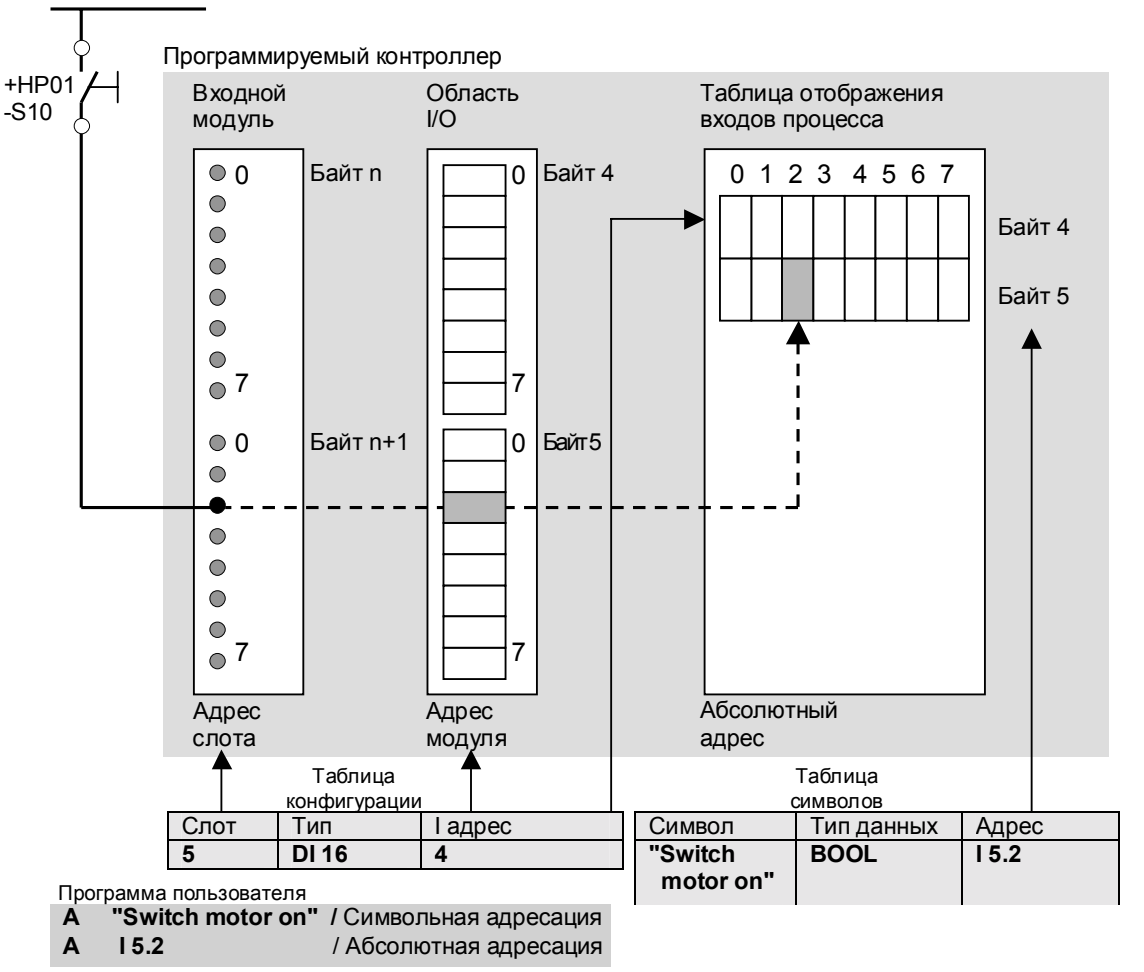
Каждый CPU имеет специальный номер для возможного соединения. Одно соединение резервируется для PG и одно соединение для OP (эти соединения не могут быть использованы для других целей).

Ресурсы соединений также требуются временно для "несконфигурированных соединений" (SFC-коммуникации).

1.4 Адресация модулей

1.4.1 Путь прохождения сигнала

При монтаже установки нужно трассировать сигналы в PLC (см. рис. 1.6).



Входной сигнал, например, сигнал от кнопки +HP01-S10, включающей мотор ("Switch motor on" - "включение мотора"), приходит во входной модуль, где он поступает на специальный терминал. Этот терминал имеет "адрес", называемый I/O-адрес (например, байт 5, бит 2).

Перед каждым началом выполнения программы CPU автоматически копирует значение сигнала в таблицу входов для сохранения "образа процесса по входу", где это значение будет доступно по входному ("input") адресу для дальнейшей обработки (например, I 5.2). При этом выражение "I 5.2" является абсолютным адресом.

Вы можете задать в таблице символов имя этому входу в виде алфавитно-цифрового символьного имени, соответствующего абсолютному адресу входного сигнала (такое, как "Switch motor on"). При этом выражение "Switch motor on" является символьным адресом.

1.4.2 Адрес слота

Каждый слот имеет фиксированный адрес в программируемом контроллере (в S7-станции). Этот адрес слота состоит из номера монтажной стойки и номера слота. Таким образом каждый модуль может быть однозначно описан указанием адреса слота ("географический адрес").

Если модуль содержит интерфейсные платы, каждая из них также описывается заданием адреса подмодуля. Таким образом, каждый дискретный или аналоговый сигнал и каждое последовательное соединение в системе имеет свой собственный уникальный адрес.

Соответственно, распределенные I/O модули также имеют "географические адреса", в данном случае включающие в себя номер системы ведущего DP-устройства и номер станции вместо номера стойки.

Вы должны использовать утилиту для конфигурирования оборудования "Hardware Configuration" для того, чтобы спланировать конфигурацию аппаратной части S7-станции, как места физического расположения модулей. Эта утилита позволяет также установить начальные адреса модулей и задать для модулей параметры (см. раздел 2.3 "Конфигурирование станций").

1.4.3 Начальный адрес модуля

Кроме адресов слотов, позволяющих определить отдельный слот, каждый модуль имеет начальный адрес, определяющий позицию в области логических адресов (область I/O-адресов). Адресное пространство входов/выходов начинается с адреса 0 и заканчивается некоторым значением, соответствующим верхней границе, которая определяется типом CPU.
Начальный адрес определяет, как адресуются входные/выходные сигналы в программе программируемого контроллера (S7-станции). Этот адрес слота состоит из номера монтажной стойки и номера слота. Таким образом каждый модуль может быть однозначно описан указанием адреса слота ("географический адрес"). В случае дискретных модулей отдельные сигналы (биты) собираются в группы по 8 (т.е. в байты). Эти байты имеют соответствующие адреса 0, 1, 2 и 3; адресация байтов начинается с начального адреса модуля. Например, в дискретном модуле с четырьмя байтами и с начальным адресов модуля 8 отдельные байты имеют адреса 8, 9, 10 и 11 соответственно.

В случае аналоговых модулей каждый из аналоговых сигналов (в виде напряжения или тока), называемых "каналами" ("channel"), занимает 2 байта. Аналоговые модули, в зависимости от конструкции имеющие 2, 4, 8 и 16 каналов, соответственно занимают 4, 8, 16 или 32 байта адресного пространства.

При включении питания (если не было предустановок) CPU устанавливает начальный адрес каждого модуля, зависящий от типа модуля, от слота и стойки. Этот начальный адрес соответствует (относительный адрес) байту 0. Вы можете видеть этот адрес в таблице конфигурации.

В системах S7-3xx с интегрированным DP-интерфейсом, S7-318 и S7-400 Вы можете изменять этот адрес. Для этого Вы должны выбрать опцию назначения начальных адресов модулей внутри разрешенного адресного пространства. Вы также можете выбрать опцию для назначения разных начальных адресов для входов и выходов для смешанных дискретных или аналоговых модулей.

Как и централизованные модули, модули распределенной периферии (станции) резервируют соответствующие номера байтов в области I/O-адресов. При этом адреса централизованных модулей и распределенных I/O не должны перекрываться.

Соответствующим образом построенные ведомые DP-устройства могут быть параметризированы таким образом, что особые номера байтов обеспечивают консистентность (логическую связанность) данных при их пересылке. Этим ведомым DP-устройствам соответствует I/O адрес одного байта, которым они адресуются при использовании системных функций SFC 14 DPRD_DAT и SFC 15 DPWR_DAT.

Дискретные модули обычно адресуются в таблицах отображения процесса таким образом, что состояния их сигналов могут автоматически обновляться и к ним обеспечивается доступ в области адресов "Input" ("Входы") и "Output" ("Выходы"). Аналоговые модули, FM и CP получают адреса не из области отображения процесса.

1.4.4 Диагностические адреса

Модули со встроенной функцией диагностики обеспечивают пользователя диагностическими данными, которые могут оцениваться в пользовательской программе. Если централизованные модули имеют адрес данных пользователя (начальный адрес модуля), то для чтения диагностических данных имеется доступ к модулю по этому адресу. Для модулей, не имеющих адреса данных пользователя, например, для источников питания, или для модулей, являющихся частью распределенной периферии, существует диагностический адрес.

Адрес данных диагностики всегда является адресом в I/O области входов и занимает один байт памяти. Длина данных пользователя по этому адресу равна 0; если данные размещены в области отображения процесса (что разрешено), то эти данные игнорируются CPU при обновлении образа процесса.

STEP 7 автоматически назначает диагностический адрес, отсчитывая длину данных вниз от верхнего максимального значения для адресов I/O. Вы можете изменять этот адрес с помощью утилиты для конфигурирования

аппаратной части.

Данные диагностики могут быть только считаны с помощью специальных системных функций; попытки доступа по адресу этих данных с помощью операторов загрузки не будут иметь эффекта (см. также раздел 20.4.1 "Адресация распределенной периферии").

1.4.5 Адреса шинных узлов

Адрес узла, номер станции

Каждая DP-станция (например, ведущее или ведомое DP-устройство или программатор) в подсети PROFIBUS имеет дополнительный адрес узла, с помощью которого станция может быть однозначно адресована на данной шине.

MPI-адрес

Модули, являющиеся узлами в MPI-сети (например, CPU, FM или CP), также имеют MPI-адрес. Этот адрес играет решающую роль для связи с PG, HMI-устройствами и для обмена посредством глобальных данных.

Нужно заметить, что в старших версиях S7-300 модули FM и CP, работающие в таких станциях, получают MPI-адрес, который выводится из MPI-адреса CPU.

В случае CPU 318 модули с MPI-связью локализуются в их собственном сегменте, так как они не имеют MPI-адреса. Они могут быть адресованы программатором посредством номера стойки и номера слота.

1.5 Адресное пространство

Адресное пространство каждого программируемого контроллера включает в себя:

- периферийные входы и выходы;
- области отображения процесса по входу и по выходу;
- области меркеров;
- области таймеров и счетчиков (см. главу 7 "Функции таймеров" и главу 8 "Функции счетчиков");
- области L-стека (см раздел 18.1.5 "Временные локальные данные").

К перечисленному нужно добавить области кода и блоков данных с локальными (внутри блока) переменными, в зависимости от программы пользователя.

1.5.1 Область данных пользователя

В SIMATIC S7 каждый модуль может иметь две области адресов: область данных пользователя, которая может быть непосредственно адресована с помощью операторов LOAD и TRANSFER и область системных данных для записей данных передачи.

При адресации модулей нет разницы в том, локализованы ли модули в стойках при централизованной конфигурации, или используются как распределенные I/O. Все модули находятся в одном и том же (логическом) адресном пространстве.

Свойства данных пользователя в модуле зависят от типа модуля. Для сигнальных модулей данные являются дискретными или аналоговыми входными/выходными сигналами. Для функциональных модулей данные могут быть, например, информацией контроля или состояния (статуса). Объем данных пользователя определяется типом модуля. Есть модули, которые располагают 1, 2, 4 или большим количеством байт в этой области. Адресация всегда начинается с байта 0. Адрес 0 байт является начальным адресом модуля; он задается в таблице конфигурации.

Данные пользователя отражает область I/O адресов, подразделяемая в зависимости от направления передачи данных на PI-область ("peripheral inputs") (область периферийных входов) и PQ-область ("peripheral outputs") (область периферийных выходов). Если данные пользователя расположены в области отображения данных процесса, то при обновлении образа процесса CPU обрабатывает данные автоматически.

Периферийные входы

Адресную область периферийных входов Вы можете использовать при чтении из области данных пользователя входных модулей. Часть PI-области адресов соответствует области отображения данных процесса. Эта часть всегда начинается с 0-го адреса I/O, при этом размер этой области определяется типом CPU.

С помощью операции прямого чтения (Direct I/O Read) Вы можете получить доступ к данным модулей, чей интерфейс не влияет на образ процесса по входу (например, аналоговые входные модули).

Состояния сигналов модулей, которые влияют на образ процесса по входу, могут также быть считаны с помощью этой операции. При этом сканируются мгновенные состояния сигналов входных битов. Необходимо учитывать, что состояния этих сигналов могут отличаться от состояний соответствующих битов области отображения входов процесса, так как образ процесса обновляется в самом начале цикла сканирования программы.

Периферийные входы могут иметь такие же абсолютные адреса, как и периферийные выходы.

Периферийные выходы

Адресную область периферийных выходов Вы можете использовать при записи в область данных пользователя выходных модулей. Часть PQ-области адресов соответствует области отображения данных процесса. Эта часть всегда начинается с 0-го адреса I/O, при этом размер этой области определяется типом CPU.

С помощью операции прямой записи (Direct I/O Write) Вы можете получить доступ к данным модулей, чей интерфейс не влияет на образ процесса по выходу (например, аналоговые выходные модули).

Состояния сигналов модулей, которые влияют на образ процесса по выходу, могут также быть изменены с помощью этой операции. При этом мгновенно изменяются состояния выходных битов. Необходимо учитывать, что изменение состояния этих битов мгновенно изменяет состояние выходных битов области отображения процесса для соответствующих модулей (!), так как нет разницы между отображением процесса по выходу и состоянием сигналов выходных модулей

Периферийные выходы могут иметь такие же абсолютные адреса, как и периферийные входы.

1.5.2 Отображение процесса (образ процесса)

Отображение процесса (образ процесса) состоит из образа дискретных входных и дискретных выходных модулей и, таким образом, подразделяется на образ входов процесса и образ выходов процесса. К образу входов процесса доступ осуществляется с помощью адресной области для входов (I), а к образу выходов процесса доступ осуществляется с помощью адресной области для выходов (Q). Как правило, установкой или процессом управление осуществляется с помощью входов и выходов. Образ процесса может быть разбит на дополнительные образы процесса, которые могут обновляться или автоматически, или под управлением пользовательской программы. Для получения более подробной информации обратитесь к разделу 20.2.1 "Обновление образа процесса".
Для S7-300 CPU и, начиная с октября 1998 г., также для S7-400 CPU Вы можете использовать адреса области отображения процесса, не занятой модулями, как дополнительную область памяти, аналогично области меркеров. Это касается как области и образа входов процесса, и образа выходов процесса.
Для отдельных CPU, скажем, для CPU 417, размер области отображения процесса может задаваться как параметр. Если Вы увеличиваете размер области отображения процесса, Вы, соответственно, уменьшаете размер рабочей (work) памяти. После изменения размера области отображения процесса CPU выполняет инициализацию рабочей (work) памяти, точно также как при холодном перезапуске.

Входы

Вход - это отображение соответствующего бита в дискретном входном модуле. Сканирование входа - это то же самое, что и сканирование бита в самом модуле. Перед выполнением программы в каждом программном цикле операционная система CPU копирует значение сигнала из модуля в образ входов процесса.
Использование образа входов процесса имеет следующие преимущества:

- Входы могут быть просканированы и записаны последовательно бит за битом (I/O биты не имеют прямого доступа).
- Сканирование входов много быстрее, чем процедура получения доступа к входному модулю (например, таким образом Вы избегаете временных потерь из-за переходных процессов в I/O шине, кроме того, время отклика системной памяти меньше, чем время отклика модуля). Следовательно, программа выполняется намного быстрее.
- Состояние входа не меняется на протяжении всего цикла программы (что означает сохранение консистентности данных на протяжении всего цикла программы). При изменении бита входного модуля это изменение состояния сигнала будет перенесено на соответствующий вход образа процесса лишь в начале следующего программного цикла.

- Входы, кроме того, могут быть установлены или сброшены, так как они находятся в RAM-памяти. С другой стороны, биты дискретных входных модулей доступны только для чтения. Входы области отображения процесса могут устанавливаться в целях отладки или запуска, для моделирования состояния датчиков. При этом заметно упрощается тестирование программы.

Эти преимущества теряются с увеличением времени отклика программы (см. раздел 20.2.4 "Время отклика").

Выходы

Выход - это отображение соответствующего бита в дискретном выходном модуле. Установка выхода - это то же самое, что и установка бита в самом модуле. Операционная система CPU копирует значение из образа выходов процесса в выходной модуль.

Использование образа выходов процесса дает следующие преимущества:

- Выходы могут быть установлены или сброшены бит за битом (прямая адресация I/O битов не возможна).
- Установка выходов много быстрее, чем процедура получения доступа к выходному модулю (например, таким образом Вы избегаете временных потерь из-за переходных процессов в I/O шине, кроме того, время отклика системной памяти меньше, чем время отклика модуля). Следовательно, программа выполняется намного быстрее.
- Состояние выхода может многократно меняться на протяжении всего цикла программы при этом состояние сигнала бита выходного модуля остается без изменения. И лишь последнее состояние сигнала будет перенесено в соответствующий выходной модуль в конце текущего программного цикла.
- Выходы, кроме того, могут быть просканированы, так как они находятся в RAM-памяти, тогда как биты дискретных выходных модулей доступны только для записи, но не для чтения. Выходы из области отображения процесса могут использоваться во всей программе.

Эти преимущества теряются с увеличением времени отклика программы. В разделе 20.2.4 "Время отклика" показано, из чего складывается время отклика программируемого контроллера.

1.5.3 Меркеры

Меркеры могут рассматриваться как дополнительные "пусковые реле" контроллера. Меркеры используются в первую очередь для хранения состояния сигналов. Они могут трактоваться как виртуальные выходы. Меркеры располагаются в области системной памяти CPU, и, следовательно, они всегда доступны. Наличное число меркеров зависит от типа выбранного CPU.

Меркеры используются для хранения промежуточных результатов, которые действительны за пределами блока, и могут обрабатываться более чем в одном блоке. Кроме блоков глобальных данных для хранения промежуточных результатов подходят:

- Временные локальные данные, возможные во всех блоках, но являющиеся действительными только для текущего вызова блока.
- Статические локальные данные, возможные только в функциональных блоках, но являющиеся действительными для многих вызовов блоков.

Ременантные меркеры

Некоторые меркеры могут быть назначены ременантными меркерами, что означает, что эти меркеры сохраняют свое состояние даже в условиях выключения питания. Ременантная область всегда начинается с 0-го адреса и заканчивается в заданном месте памяти. Ременантная область может быть задана при параметризации CPU. Более подробная информация находится в разделе 22.2.3 "Ременантность".

Тактовые меркеры

Многие процедуры в контроллере требуют периодического сигнала. Такие сигналы могут быть получены с помощью таймеров (генератор тактовых импульсов), таймерных (watchdog) прерываний (выполнение программы с управлением по времени) или просто с использованием тактовых меркеров.

Тактовые меркеры - это биты, состояния сигнала которых меняются периодически с отношением сигнал/пауза, равным 1:1. Такие биты, объединенные в байт, обеспечивают фиксированные частоты периодических колебаний (см. рис.1.7).

Вы можете задать число тактовых меркеров при параметризации CPU.

Необходимо отметить, что обновление тактовых меркеров асинхронно по отношению к выполнению главной программы.

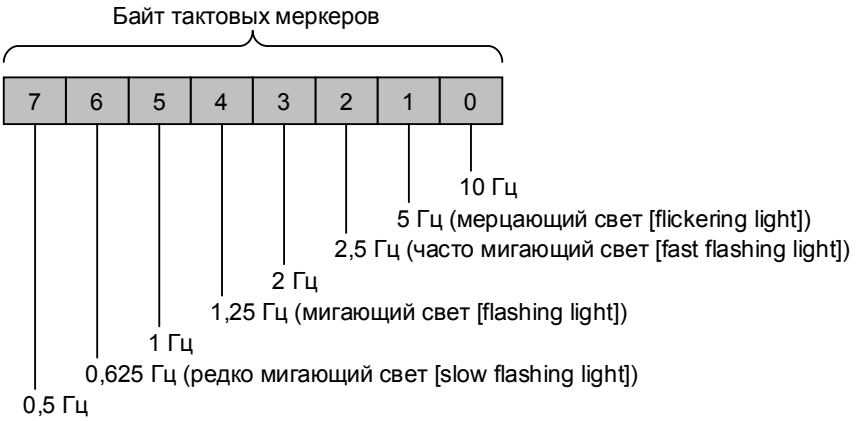


Рис.1.7 Состав байта тактовых меркеров

2 Программное обеспечение STEP 7

2.1 Базовый пакет STEP 7 (STEP 7 Basic Package)

В этой главе описывается базовый пакет STEP 7 (STEP 7 Basic Package) версии 5.1. В то время как в первой главе рассматривается краткий обзор характеристик программируемых контроллеров, в данной главе показано, как задать эти характеристики.

Базовый пакет STEP 7 (STEP 7 Basic Package) содержит следующие языки программирования: STL ("statement list" - список мнемоник), LAD ("ladder diagram" - контактный план), FBD ("function block diagram" - функциональный план). В добавление к базовому пакету возможна поставка по специальному заказу пакетов S7-SCL ("Structured Control Language" – структурированнный язык управления), S7-GRAPH (для графической разработки программ систем автоматизации SIMATIC в виде последовательности шагов и переходов между ними), S7-HiGraph (для графической разработки программ систем автоматизации SIMATIC в виде графа состояний системы и переходов между ними).

2.1.1 Инсталляция

Пакет STEP 7 V 5 является 32-разрядным приложением, работающим под управлением следующих операционных систем: Microsoft Windows 95 (начиная с сервисного пакета Service Pack 1, версии 4.00.950a), Windows 98 или Windows NT (начиная с сервисного пакета Service Pack 2, версии 4.00.1381).

Для работы с программами STEP 7 под управлением Windows 95/98 Вам потребуется программатор (PG) или компьютер (ПК) с процессором не хуже 80486 и с объемом ОЗУ не меньше 32 Мб (рекомендуемая конфигурация: процессор Pentium и объем ОЗУ 64 Мб и выше). Для работы под управлением Windows NT требуются процессор Pentium и объем ОЗУ 32 Мб и выше; кроме того необходима администраторская авторизация для инсталляции STEP 7 под Windows NT.

Если Вы работаете с большими проектами STEP 7, включающими в себя, скажем, несколько станций, состоящих из более чем 100 модулей, то Вам необходим PG или ПК с процессором повышенной производительности.

Пакет STEP 7 V 5 требует от 200 до 380 Мб памяти на жестком диске для каждой локализации (например, для англоязычной) в зависимости от операционной системы и используемой файловой системы. Также требуется предусмотреть свободное место на жестком диске для файла подкачки (от 128 до 256 Мб).

Вы должны обеспечить достаточный объем памяти на диске, содержащем данные Вашего проекта. Отдельные операции, такие как копирование проекта, могут потребовать дополнительной памяти. В случае недостаточного объема свободного пространства на диске для файла подкачки может произойти сбой в работе программы. Поэтому не рекомендуется хранить данные проекта на одном диске с файлом подкачки Windows.

Для инсталляции STEP 7 служит программа SETUP для Windows 9x/NT, которую Вы можете найти на компакт-диске. На программаторах PG STEP 7 устанавливается производителем.

Кроме STEP 7 компакт-диск также содержит программу авторизации (см. ниже), программу NCM для конфигурирования коммуникационных процессоров и электронные справочники по STEP 7 в формате Acrobat Reader V3.01.

Для интерактивного подключения требуется MPI-интерфейс. Программаторы PG имеют встроенный MPI-интерфейс, тогда как компьютеры требуют установки MPI-модуля.

Если Вам необходимо использовать модули памяти ПК, то потребуется также программатор модулей памяти.

Пакет STEP 7 V 5 имеет возможность работы в многопользовательском режиме, что означает, что проект, хранящийся на сервере, может редактироваться одновременно с нескольких рабочих станций. Для настройки Вы должны выполнить необходимые установки в панели управления Windows с помощью программы SIMATIC Workstation. В появившемся диалоговом окне Вы можете задать параметры рабочей станции для однопользовательской или многопользовательской системы с соответствующим протоколом.

2.1.2 Авторизация

Для работы с пакетом STEP 7 требуется выполнить авторизацию (подтвердить право использования). Программа авторизации находится на дискете. После инсталляции STEP 7 Вам будет предложено выполнить авторизацию, если данный жесткий диск пока не содержит авторизации. Вы также можете выполнить авторизацию позже, спустя некоторое время.

Вы можете также переносить авторизацию на другой ПК, сначала возвратив авторизацию на дискету-оригинал и затем установив ее на другом ПК.

В случае потери Вами авторизации, например, из-за выхода из строя жесткого диска, Вы можете в течение ограниченного времени (до замены Вашей авторизации) использовать "аварийную лицензию" (emergency license), находящуюся на той же дискете-оригинале (дискете с авторизацией).

2.1.3 SIMATIC Manager

SIMATIC Manager является главной утилитой STEP 7. Вы найдете ее значок на рабочем столе Windows:

SIMATIC Manager запускается двойным щелчком кнопкой мыши на значке.

При первом запуске активизируется программа "мастер проекта" (Project Wizard). Эта программа может быть использована для быстрого создания новых проектов. Тем не менее, Вы можете выключить эту программу с помощью элемента управления Check box "Display Wizard on starting the SIMATIC Manager" ("Отображать мастер-программу при запуске SIMATIC Manager"). Мастер-программа может быть вызвана при необходимости с помощью команд меню: *File (Файл) -> "New Project" Wizard*.

Процесс программирования начинается при открытии или запуске проекта ("project"). Примеры проектов представляют собой хороший материал для ознакомления.

При открытии примера проекта ZEn01_09_S7_ZEBRA с помощью команд меню: *File (Файл) -> Open (Открыть)*, Вы увидите разделенное окно проекта: слева будет структура открытого объекта (иерархическая), а справа - выбранный объект (Рис.2.1).

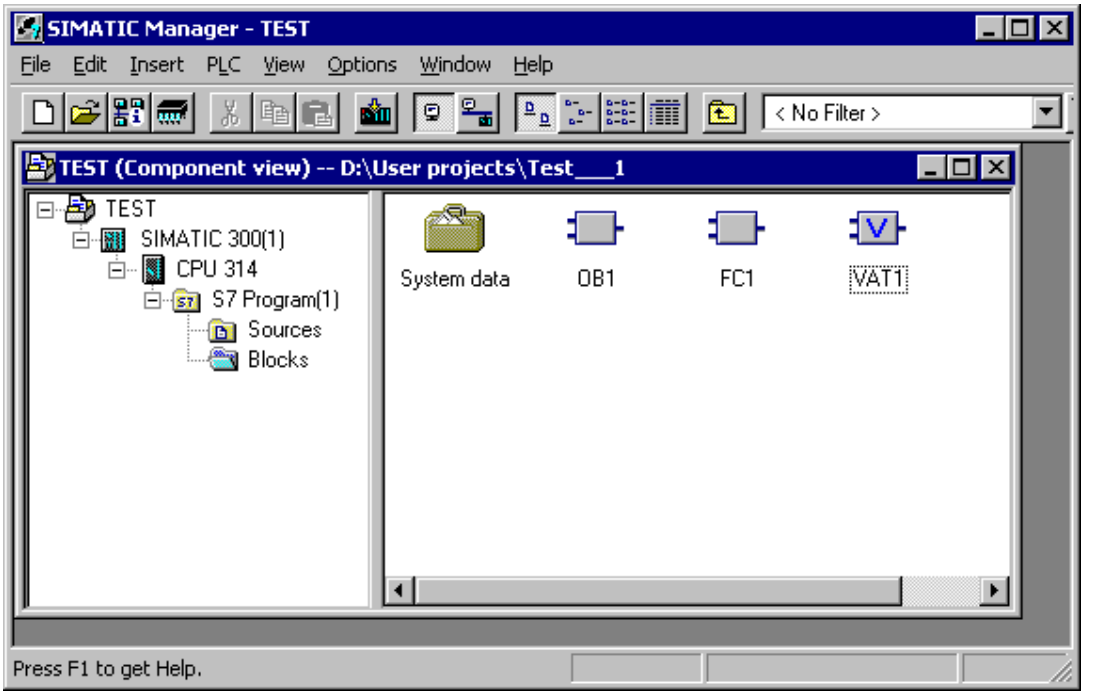


Рис.2.1 Пример открытого окна утилиты SIMATIC Manager

Щелчок на значке квадрата со знаком "+" позволяет открыть вложенные уровни структуры объекта; выбор объекта в левой части окна всегда вызывает отображение его содержания в правой части окна.

С помощью SIMATIC Manager Вы сможете работать в среде STEP 7. "Логические" объекты, отображаемые в окнах SIMATIC Manager, соответствуют "реальным" объектам Вашей установки (процесса). Проект включает в себя установку (процесс) в целом, тогда как станция (station) соответствует программируемому контроллеру (PLC).

Структура	*Объект*	*Описание*
Project	Папка для всех объектов проекта	
MPI [PTP, PROFIBUS, Ethernet]	**Subnet (Подсеть)**	Содержит параметры для подсети
SIMATIC 300/400 station	Папка для всех объектов станции	
Hardware (Аппаратное обеспечение)	**Configuration table (Таблица конфигурации)**	Содержит данные конфигурации для станции и параметры для модулей
CPU xxx	Папка для всех объектов CPU	
Connections (Соединения)	**Connection table (Таблица соединений)**	Содержит данные коммуникаций для узлов сети
S7 program	Папка для всех объектов программы пользователя	
Symbols (Символы)	**Symbol table (Таблица символов)**	Содержит символы (символьные имена) для абсолютной адресации GD
Sources	Папка для всех исходных программ	
Source files (Исходные файлы)	**Source programs (Исходные программы)**	Содержат исходные файлы программы пользователя (STL-, SCL-программы…)
Blocks	Папка для скомпилированных объектов программы пользователя	
OB n FB n FC n DB n	**Организационные блоки** **Функциональные блоки** **Функции** **Блоки данных**	Содержат скомпилированные коды и данные программы пользователя
SFC n SFB n	**Системные функции** **Системные функц. блоки**	Содержат интерфейс вызова системных блоков, встроенных в CPU
System data (Системные данные)	**Системные блоки данных**	Содержат скомпилированные данные для таблицы конфигурации
UDT n	**Data types (Пользовательские типы данных)**	Содержат определения типов данных, определенных пользователем
VAT n	**Variable table (Таблица переменных)**	Содержит переменные для мониторинга и модификации
S7 program	Папка для программы пользователя, которая назначена не для любого CPU (имеет такую же структуру, как и любая S 7-программа, назначенная CPU)	

Рис.2.2 Иерархическая структура объектов проекта STEP 7

Проект может содержать несколько станций, связанных друг с другом, например, посредством подсети MPI. Станция содержит CPU, а CPU содержит S7-программу. В свою очередь программа включает в себя другие объекты, такие как объект *Blocks* (блоки), содержащий среди прочего скомпилированные блоки.

Объекты STEP 7 объединяются в древовидную структуру. На рис. 2.2 показаны наиболее важные компоненты этой древовидной структуры ("main branch" - "главная ветвь"), которые характерны для работы с базовым пакетом S7 в автономном режиме (offline view). Объекты, выделенные жирным шрифтом, содержат другие объекты. В автономном режиме (offline view) все показанные на рисунке объекты доступны пользователю. Эти объекты расположены на жестком диске программатора PG. Если Ваш PG находится в интерактивной связи (online) с CPU (обычная система управления с PLC), Вы можете включить интерактивный режим (online view), выбрав опции меню: *View -> Online (Режим -> Интерактивный)*. Эта опция вызывает другое окно проекта, содержащее объекты назначенного устройства; при этом объекты, выделенные на рисунке, более не отображаются.

Вы можете видеть на панели заголовка окна активного проекта, работаете ли Вы в интерактивном (online) или в автономном (offline) режиме. Для более четкого разделения для панели заголовка и заголовка окна этих режимов могут быть установлены различные цвета. Для этого выберите опции меню: *Options -> Customize (Опции -> Установки пользователя)* и измените соответствующие параметры на вкладке "View" ("Режим").

Выбрав опции меню: *Options -> Customize (Опции -> Установки пользователя)*, можно изменить базовые установки SIMATIC Manager, такие как session language (язык), архив программы и место расположения для проектов, библиотек и конфигурирование архива программы.

Последовательность редактирования

Следующие пункты касаются общего редактирования объектов:

Выбрать объект - означает щелкнуть кнопкой мыши один раз на объекте в одной из частей окна проекта, после чего объект становится выделенным.

Присвоить имя объекту - означает щелкнуть кнопкой мыши на имени выбранного (см. выше) объекта, после чего вокруг имени объекта появится рамка, и Вы сможете изменить имя в окне, или, выбрав опции меню: *Edit -> Object Properties (Редактирование -> Свойства объекта)*, можно изменить имя объекта в появившемся диалоговом окне. Для некоторых объектов, таких как CPU, Вы можете изменить имя только с помощью специальных утилит (приложений), в данном случае с помощью утилиты для конфигурирования оборудования (Hardware Configuration).

Открыть объект - означает щелкнуть кнопкой мыши два раза на объекте, после чего, если объект содержит другие объекты, SIMATIC Manager отобразит его содержание в правой части окна, а если объект находится на нижнем уровне структурной иерархии, то SIMATIC Manager запустит соответствующее приложение для редактирования объекта (например, двойной щелчок на блоке запустит программу для редактирования последнего).

В данной книге пункты на стандартной панели меню в верхней части окна описаны как последовательность операторов. Программисты, имеющие опыт в использовании операторного интерфейса используют значки из панели инструментов. Использование правой кнопки мыши может быть очень полезным. Однократный щелчок правой кнопки на объекте вызывает меню с текущими опциями редактирования.

2.1.4 Проекты и библиотеки (**Project**(s) **и Library**(ies))

В STEP 7 "главные объекты", находящиеся на верхнем уровне структурной иерархии, это проекты (project) и библиотеки (library).

Проекты (projects) используются для систематического хранения данных и программ для решения задачи автоматизации. Важнейшие из них:

- данные конфигурации оборудования;
- параметры для модулей;
- данные конфигурации сетевых коммуникаций;
- программы (коды и данные, символы, исходные программы).

Объекты в проекте организованы в виде иерархической системы. Первым шагом для редактирования всех объектов проекта является открытие проекта. В следующих разделах обсуждается процесс редактирования этих объектов.

Библиотеки (library) используются для хранения многократно используемых компонентов программы. Библиотеки организованы в виде иерархической системы. Они могут содержать STEP 7 программы, которые в свою очередь могут содержать программы пользователя (скомпилированные блоки), исходные тексты программ и таблицы символов. За исключением возможности интерактивной (online) связи (не возможна отладка программы), создание программ или частей программ в библиотеке обеспечивает такие же функциональные возможности как и у объекта.

В комплекте поставки STEP 7 V5 находится стандартная библиотека Standard Library, содержащая следующие разделы:

- System Function Blocks (системные функциональные блоки), содержащие интерфейсы вызовов системных блоков для создания программ в автономном режиме, что функционально обеспечивается в CPU;
- S5-S7 Converting Blocks (блоки S5-S7 преобразования), содержащие загружаемые функции для S5-S7 преобразования (для замены стандартных функциональных блоков S5 в процессе конвертирования программы);
- T1-S7 Converting Blocks (блоки TI-S7 преобразования), содержащие дополнительные загружаемые функции и функциональные блоки для TI-S7 преобразования;
- IEC Function Blocks (функциональные блоки IEC), содержащие загружаемые функции для редактирования комплексных переменных типов DATE_AND_TIME и STRING;
- Communication Blocks (коммуникационные блоки), содержащие загружаемые функции для управления модулями CP;
- PID Control Blocks (блоки ПИД-управления), содержащий загружаемые функциональные блоки для систем автоматического управления;
- Organization Blocks (организационные блоки), содержащий шаблоны для организационных блоков (раздел объявления переменных для стартовой информации).

Вы можете найти обзор содержания этих библиотек в главе 33 "Библиотеки блоков".

Если Вы приобрели S7-модуль со стандартными блоками, программа инсталляции модуля установит эти стандартные блоки на жестком диске в виде библиотеки. В дальнейшем Вы сможете копировать эти блоки из библиотеки в Ваш проект. Библиотека может быть открыта с помощью выбора опций меню: *File -> Open (Файл -> Открыть)*, после чего она может быть отредактирована таким же образом как и проект. Вы можете также создавать свои собственные библиотеки.

С помощью выбора опций меню: *File -> New (Файл -> Создать)* может быть сгенерирован новый объект высшего уровня структурной иерархии (проект или библиотека). Место расположения вновь создаваемого объекта (проекта или библиотеки) в структуре каталогов должно быть определено с помощью опций меню: *Options -> Customize (Опции -> Установки пользователя)* или с помощью диалогового окна New (Создать).

Пункт меню Insert (Вставить) используется для добавления новых объектов в уже существующие объекты (например, для добавления нового блока в программу). Тем не менее, перед этой операцией Вы должны сначала выбрать каталог, в который Вы вставите объект из левой части окна SIMATIC Manager.

Вы можете скопировать объект с помощью опций меню: *Edit -> Copy (Правка -> Копировать)* и *Edit -> Paste (Правка -> Вставить)* или с помощью приема, распространенного среди пользователей Windows, называемого "drag-n-drop", то есть методом перетаскивания объекта с помощью мыши из одного окна в другое. Необходимо помнить, однако, что Вы не сможете отменить удаление объекта или каталога объекта в SIMATIC Manager.

2.1.5 Интерактивная справочная система (Online Help)

Интерактивная справочная система (Online Help) в SIMATIC Manager обеспечит Вас информацией в процессе программирования, что снимает необходимость пользования печатными справочными руководствами. Вы можете выбирать интересующие Вас темы, выбрав пункт меню Help (Справка). Выбор в справочной системе пункта Getting Started (Запуск), к примеру, выводит краткое резюме по использованию утилиты SIMATIC Manager.

При выборе опций: Help -> Contents (Справка -> Содержание) запускается центральная функция справочной системы STEP 7 из любого приложения. Она содержит все основные сведения.

При выборе опций: Help -> Context-Sensitive Help F1 (Справка -> Контекстная справка) запускается контекстная справочная система, то есть если Вы нажмете клавишу F1, то Вы получите информацию, соответствующую выбранному с помощью манипулятора "мышь" объекту, или информацию, соответствующую текущему сообщению об ошибке.

Если на панели Вы щелкните на кнопке со знаками стрелки и вопроса, символ вопроса добавится к указателю мыши. Установив такой указатель на объект (например, на символ или команду меню), Вы получите соответствующую интерактивную справочную информацию.

2.2 Редактирование проектов

При создании проекта Вы должны создать "каталоги" ("папки") для данных проекта, затем Вы должны сгенерировать эти данные и занести их в эти каталоги. Обычно Вы создаете проект, применяя подходящее оборудование, конфигурируете это оборудование (по крайней мере, CPU) и создаете каталог для программы пользователя. Тем не менее, Вы можете поместить S7-программу непосредственно в каталог проекта без включения какого-либо оборудования вообще. Заметьте, что инициализация модулей (изменение адресов, установки CPU, конфигурирование соединений) возможна только с помощью утилиты конфигурирования оборудования Hardware Configuration tool.

Мы настоятельно рекомендуем, чтобы редактирование проекта в целом выполнялось с использованием SIMATIC Manager. Создание, копирование или удаление каталогов или файлов, а также изменение имен (!) в структуре проекта с помощью Windows Explorer (Проводника) может привести к возникновению проблем при использовании в дальнейшем утилиты SIMATIC Manager.

2.2.1 Создание проектов

Project Wizard (Мастер проектов)

Начиная с версии STEP 7 V3.2 программа STEP 7 Wizard помогает пользователю при создании новых проектов. Пользователь должен задать тип используемого CPU, и программа-мастер создаст проект с S7-станцией и выбранным CPU, а также каталог для S7-программы, каталог для исходных программ и каталог блоков с выбранными организационными блоками.

Создание проекта с S7-станцией

Если Вы желаете создать новый проект "вручную", в данном разделе Вы найдете перечень необходимых действий, которые Вы должны будете выполнить. В разделе 2.1.3 "SIMATIC Manager" Вы найдете общую информацию по редактированию объектов.

Создание нового проекта

Выберите опции меню: *File -> New (Файл -> Создать)*, введите имя в диалоговом окне, измените тип и место расположения, если это необходимо, и подтвердите Ваш выбор щелчком на кнопке "OK" или нажатием клавиши "Enter".

Вставка новой станции в проект

Выберите проект и вставьте станцию с помощью опций меню: *Insert -> Station -> Simatic 300 Station (Вставка -> Станция -> Станция S7-300)* (в данном случае станция S7-300).

Конфигурирование станции

Щелкните на прямоугольнике со значком плюса, следующем за объектом *project* в левой части окна проекта и выберите станцию; SIMATIC Manager отображает объект Hardware (оборудование) в правой части окна. Двойным щелчком по *Hardware* запускается утилита конфигурирования оборудования Hardware Configuration, с помощью которой осуществляется редактирование таблиц конфигурации.

Если каталог модулей не показан на экране, то вызовите его с помощью опций меню: View -> Catalog (Вид -> Каталог).

Конфигурирование начинается с выбора несущей шины (rail), например, в "SIMATIC 300" и "RACK 300" и переносом методом "drag-n-drop" посредством мыши на свободное место в верхней половине окна станции (station window). При этом Вы можете наблюдать таблицу, в которой показаны слоты на шине.

На следующем этапе Вы должны выбирать требуемые модули из каталога модулей и, используя процедуру "drag-n-drop", переносить эти модули в соответствующие слоты. Для дальнейшего редактирования структуры проекта требуется установить по крайней мере один CPU, например, CPU 314 в слот 2. Вы можете добавлять остальные необходимые модули позже. Редактирование конфигурации оборудования подробно обсуждается в разделе 2.3 "Конфигурирование станций".

Затем Вы должны сохранить и скомпилировать станцию, после чего закройте ее и вернитесь в SIMATIC Manager. Кроме конфигурации оборудования открытая станция показывает также CPU.

При конфигурировании CPU утилита SIMATIC Manager также создает S7-программу со всеми объектами. Создание структуры проекта при этом завершается.

Просмотр содержания S7-программы

Откройте CPU; в правой части окна проекта Вы можете видеть символы для *S7-программы (S7-program)* и для таблицы соединений (connection table).

Откройте *S7-program*; SIMATIC Manager отображает символы для скомпилированной программы пользователя (*Blocks - Блоки*), каталог для исходных программ и таблицу символов в правой части окна.

Откройте программу пользователя (*Blocks - Блоки*); SIMATIC Manager отображает символы для скомпилированных данных конфигурации (*System data - Системные данные*) и пустой организационный блок для основной (main) программы (OB1) в правой части окна.

Редактирование объектов программы пользователя

Теперь мы достигли нижнего уровня иерархической структуры объектов. При первом открытии OB 1 отображается окно свойств объекта и запускается редактор для редактирования организационного блока. Вы можете добавлять другие пустые блоки для инкрементного редактирования посредством выбора пунктов: Insert -> S7 Block -> … (*Blocks* должно быть выделено) и выбором требуемого типа из представленного списка.

При открытии объекта *System data* (*Системные данные*) будет показан список доступных блоков системных данных. Здесь Вы получаете скомпилированные данные конфигурации.

Блоки системных данных могут быть отредактированы с помощью объекта *Hardware (Оборудование)* в каталоге *Station (Станция)*. Вы можете передать *System data (Системные данные)* в CPU, выбрав опции: *PLC -> Download (PLC -> Загрузить)*, и тем самым параметризировав CPU.

Каталог *Source Files (Исходные файлы)* пуст. Для вставки пустого исходного файла в *Source Files* Вы можете использовать меню: *Insert -> S7 Software -> STL Source File* (*Вставить -> S7ПО -> STL-исходный файл)* или для вставки в *Source Files* исходного файла, созданного в формате ASCII посредством стороннего (не из ПО STEP) редактора, Вы можете использовать меню: *Insert -> External Source File* (*Вставить -> Внешний исходный файл).*

Создание проекта без S7-станции

Если Вы желаете, Вы можете создать программу без предварительного конфигурирования станции. Для этого сами создайте каталог для программы. Выберите проект и сгенерируйте S7-программу, используя опции меню: *Insert -> Program -> S7 Program* (*Вставить -> Программу -> S7- программу)*. В данной S7-программе SIMATIC Manager создает объект *Symbols (Символы)* и каталоги объектов *Sources (Исходные файлы)* и *Blocks (Блоки).* Каталог *Blocks (Блоки)* содержит пустой блок OB 1.

Создание библиотеки

Вы можете также создать программу в объекте *library (библиотека)*, если Вы хотите использовать ее больше, чем один раз. При этом такая стандартная программа будет всегда доступна, и Вы сможете ее копировать полностью или по частям в свою текущую программу. Помните, что при этом у Вас нет возможности интерактивной (online) связи с библиотекой, и Вы сможете отладить S7-программу только в составе проекта.

2.2.2 Управление, перекомпоновка и архивирование

SIMATIC Manager поддерживает перечень всех известных "основных объектов" ("main objects"), организованных в соответствии с проектами пользователя, библиотеками и примерами (образцами) проектов. Инсталлируйте примеры (образцы) проектов со стандартными библиотеками для STEP 7 и проекты пользователя со своими собственными библиотеками.

При активации опции реорганизации *File -> Rearrange (Файл -> Реорганизация)* SIMATIC Manager убирает разрывы в пространстве памяти, получившиеся при выполнении операции удаления, оптимизируя занятое пространство памяти аналогично программе дефрагментации на жестком диске. Процесс реорганизации требует определенного времени, зависящего от объема перемещаемых данных.

Вы можете также архивировать проект или библиотеку с помощью опций: *File -> Archive (Файл -> Архивация)*. В этом случае SIMATIC Manager будет сохранять выбранный объект (каталог проекта или библиотеки со всеми подкаталогами и файлами) в сжатом виде в архивном файле.

Чтобы заархивировать проект или библиотеку, нужно использовать программу архивации. STEP 7 содержит программы архивации ARJ и PKZIP 2.50, но Вы можете использовать и другие программы архивации (например, *winzip,* начиная с версии 6.0, *pkzip,* начиная с версии 2.04g, *JAR,* начиная с версии 1.02 или *LHARC,* начиная с версии 2.13).

Проекты или библиотеки не могут быть отредактированы, когда они находятся в заархивированном (сжатом) состоянии. Вы можете "распаковывать" заархивированные объекты посредством опций: *File -> Retrieve (Файл -> Восстановление),* и после этого Вы можете редактировать эти объекты. Разархивированные объекты автоматически принимаются системой управления проектами или библиотеками.

Вы можете задавать каталоги назначения для файлов архивов и для восстанавливаемых из архивов объектов на вкладке "Archive" ("Архив"), вызываемой с помощью опций меню: *Options -> Customize (Опции -> Установки пользователя)*. Вы сами можете выбирать каталоги назначения для файлов архивов и для восстановленных файлов или можете выбрать опцию "Generate archive name automatically" ("Задавать имя архива автоматически"), что позволяет не делать никаких назначений при архивации/восстановлении, так как имя файла архива будет сгенерировано из имени проекта.

Архивирование проекта в CPU

Начиная с версии ПО STEP 7 V 5.1, при использовании соответствующих S7-400 CPU Вы можете сохранять проект в архивной (сжатой) форме в загрузочной памяти CPU, то есть в модуле памяти. Таким образом, Вы можете сохранять все данные проекта, требуемые для полностью обеспеченной обработки программы пользователя, включая таблицы символов и исходные файлы, непосредственно в установке, для которой они предназначены. Если становится необходимым модифицировать или дополнить программу, то Вы можете выгрузить эти данные на жесткий диск, сделать необходимые изменения в данных проекта и вновь сохранить обновленные данные в CPU.

При загрузке данных проекта в модуль памяти, включенный в CPU, откройте проект, отметьте CPU и выберите *PLC -> Save Project on Memory Card (PLC -> Сохранить проект в модуле памяти).* Выгрузка данных проекта из модуля памяти на жесткий диск производится при выборе: *PLC -> Retrieve Project from Memory Card (PLC -> Восстановить проект из модуля памяти).* Необходимо помнить, что при записи в модуль памяти, включенный в CPU, производится запись всего содержимого загрузочной памяти, включая системные данные и программы пользователя.

Если Вы хотите считать данные проекта, сохраненные в CPU, без создания проекта на жестком диске, то выберите соответствующий CPU с *PLC -> Display Accessible Nodes (PLC -> Отобразить доступные узлы).* Если модуль памяти включен в гнездо программатора PG, то выберите: *File -> S7 Memory Card -> Open* (*Файл -> S7 модуль памяти -> Открыть)* перед передачей данных.

2.2.3 Версии проекта (Project Versions)

Существуют три различные версии проектов SIMATIC. STEP 7 V1 позволяет создавать проекты версии 1, STEP 7 V2 позволяет создавать проекты версии 2, STEP 7 версий V3/ V4/ V5.0 позволяет создавать и редактировать проекты двух версий - 2 и 3. С помощью STEP 7 V5.1 Вы можете создавать и редактировать проекты версии 3 и библиотеки версии 3.

При наличии проекта версии 1 Вы можете преобразовать его в проект версии 2, используя опции меню: *File -> Open Version 1 Project* (*Файл -> Открыть проект версии 1)*. При этом структура проекта в программах, скомпилированные блоки версии 1, исходные STL-программы, таблица символов и конфигурация оборудования остаются неизменными.

Вы можете создавать и редактировать проекты версии 2 с помощью STEP 7 V2, V3, V4 и V5.0 (см. рис.2.3).

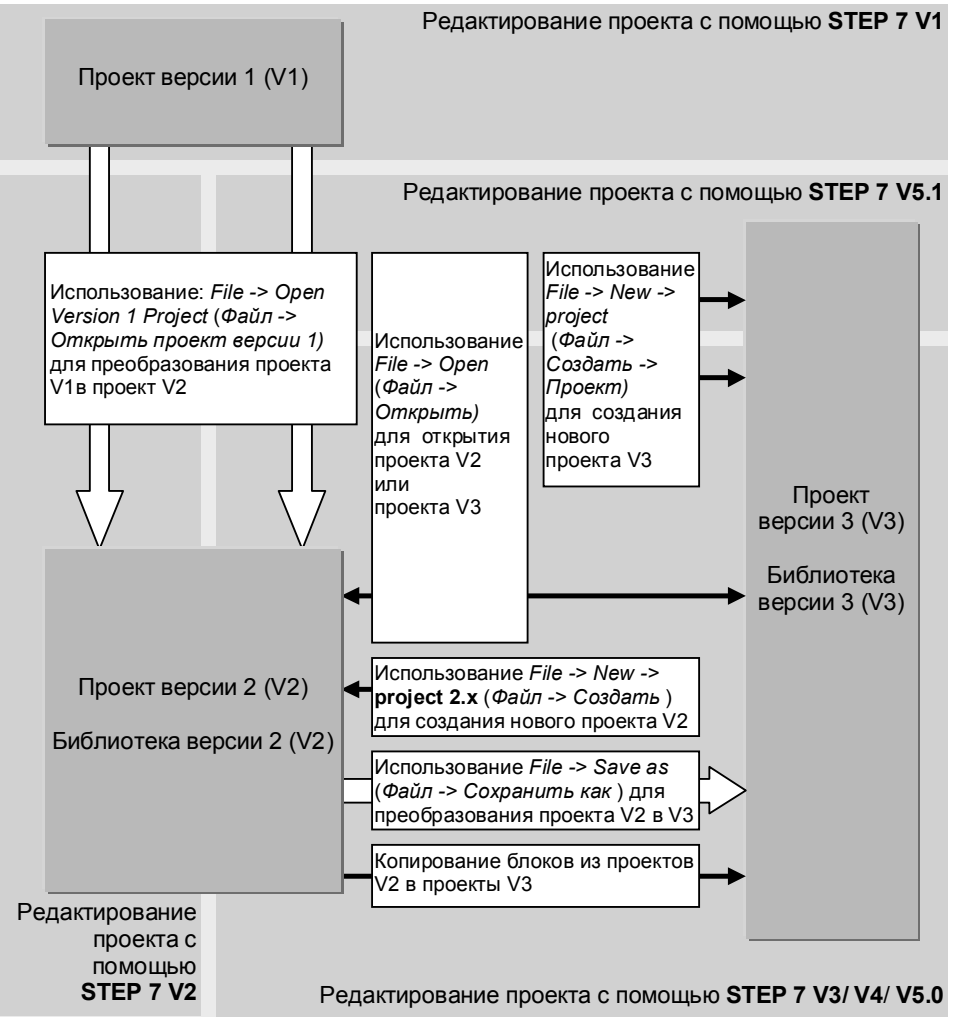


Рис.2.3 Редактирование проектов с помощью STEP различных версий

STEP 7 V5.1 позволяет работать только с проектами версии 3. Тем не менее, в этой версии Вы можете конвертировать проекты V1 в проекты V2, используя опции меню: *File -> Open Version 1 Project* (*Файл -> Открыть проект версии 1)*. Также Вы сможете открыть проект версии 2, используя опции меню: *File -> Open* (*Файл -> Открыть)*. Но при этом невозможно создавать, а также сохранять проекты в формате версии V2.

2.3 Конфигурирование станций

Для планирования конфигурации программируемого контроллера Вы должны использовать утилиту Hardware Configuration. Конфигурирование производится в автономном режиме (offline), т.е. без установления связи с CPU. Вы можете также использовать эту утилиту для адресации и параметризации модулей. Вы можете сконфигурировать оборудование на этапе планирования или же сначала инсталлировать все компоненты оборудования.

Запуск конфигурирования оборудования производится путем выбора станции с последующим выбором опций меню: *Edit -> Open Object (Правка -> Открыть объект)* или просто двойным щелчком на объекте оборудования (Hardware object) в открытом каталоге *SIMATIC 300/400 Station.* Вы можете сделать основные установки (basic settings) для оборудования, выбрав опции меню: *Options -> Customize (Опции -> Установки пользователя)*.

После выполнения конфигурирования оборудования Вы можете проверить Ваши установки на наличие ошибок с помощью выбора опций меню: *Station -> Consistency Check (Станция -> Проверка соответствия)*. При выборе опций меню: *Station -> Save (Станция -> Сохранение)* производится сохранение на жестком диске таблиц конфигурации со всеми сделанными назначениями параметров Вашего проекта.

При выборе опций меню: *Station -> Save and Compile (Станция -> Сохранение и компилирование)* производится не только сохранение на жестком диске таблиц конфигурации со всеми сделанными назначениями параметров проекта, но и их компилирование и сохранение скомпилированных данных в объекте *System data (Системные данные)* в "автономном" (offline) каталоге *Blocks (Блоки)*. После компилирования Вы можете передать сконфигурированные данные в CPU, выбрав опции меню: *PLC -> Download (PLC -> Загрузить).* Объект *System data (Системные данные)* в "интерактивном" (online) каталоге *Blocks (Блоки)* содержит текущие данные конфигурации в CPU. Вы можете "вернуть" эти данные на жесткий диск, выбрав опции меню: *PLC -> Upload (PLC -> Выгрузить).*

Вы можете также экспортировать данные конфигурирования оборудования, выбрав опции меню: *Station -> Export (Станция -> Экспорт)*. В этом случае STEP 7 создаст файл в ASCII формате, который будет содержать данные конфигурации и данные параметризации модулей. При этом Вы можете выбирать между текстовым форматом файла, когда данные можно прочитать в виде английских символов, и компактным (шестнадцатеричным) форматом данных. Вы можете также импортировать соответствующим образом структурированный ASCII файл.

Контрольная сумма (Checksum)

Утилита для конфигурирования оборудования Hardware Configuration генерирует контрольную сумму для корректно скомпилированной станции и сохраняет ее в системных данных. Идентичная системная конфигурация будет иметь точно такую же контрольную сумму, поэтому Вы можете легко сравнивать "автономную" (offline) и "интерактивную" (online) конфигурации.

Собственно контрольная сумма (Checksum) является характеристикой объекта *System data (Системные данные).*

Для считывания контрольной суммы откройте каталог *Blocks (Блоки)* в S7-программе, выберите объект *System data (Системные данные)* и откройте его с помощью опций: *Edit -> Open Object (Правка -> Открыть объект).* Программа пользователя также имеет собственное значение контрольной суммы. Вы можете найти этот параметр среди контрольных сумм системных данных в свойствах *Blocks (Блоки)*: выберите каталог *Blocks (Блоки)*, а затем опции меню: *Edit -> Object Properties (Правка -> Свойства объекта)* на вкладке "Checksums" (контрольные суммы).

Окно станции (Station)

При открытии утилиты для конфигурирования оборудования Hardware Configuration отображается окно станции и каталог оборудования (см. ниже рис. 2.4).

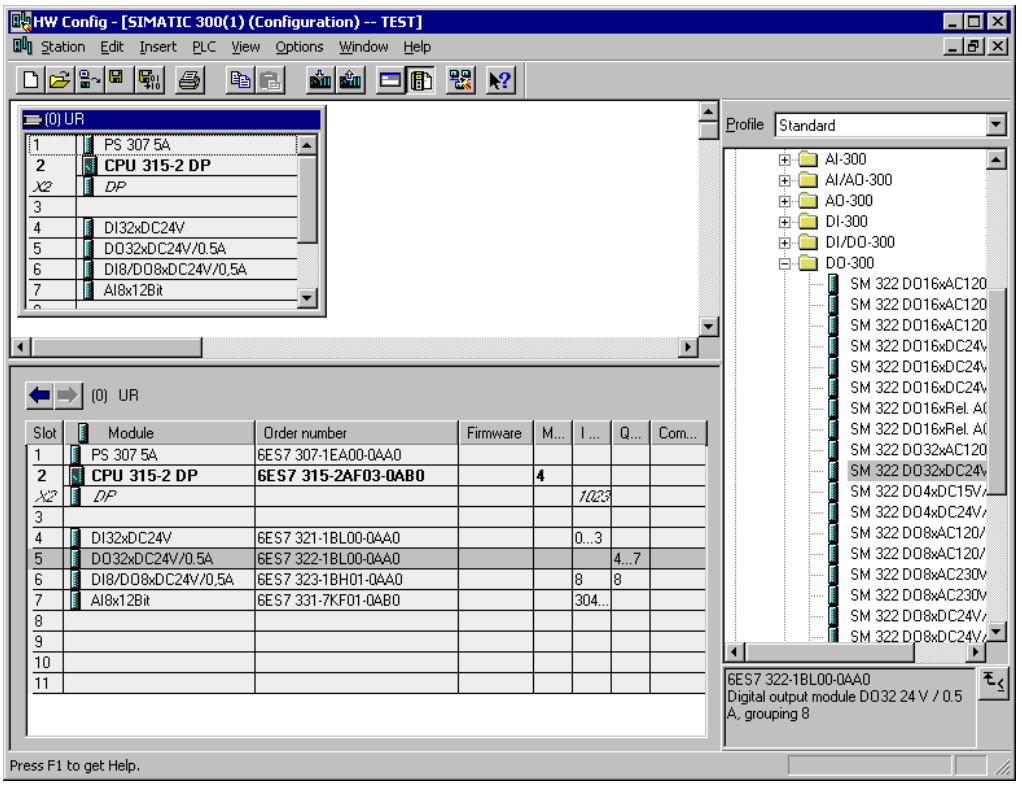


Рис. 2.4 Пример окна станции, открытого утилитой конфигурирования оборудования Hardware Configuration

Для удобства редактирования Вы можете увеличить (максимизировать) окно станции. В верхней части окна показаны монтажные стойки в форме таблиц и DP-станции в форме символов. Если используются несколько стоек, то Вы можете видеть соединения между интерфейсными модулями, а если используется подсеть PROFIBUS, Вы можете видеть систему ведущего DP-устройства. В нижней части окна станции показана таблица конфигурации, которая дает подробную информацию о стойке или о ведомом DP-устройстве, выбранном в верхней части окна.

Каталог оборудования (Hardware)

Вы можете скрывать и показывать каталог оборудования с помощью опций: *View -> Catalog (Вид -> Каталог).* Каталог отображает все доступные монтажные стойки, модули, интерфейсные модули, совместимые с STEP 7. С помощью опций: *Options -> Edit Catalog Profile (Опции -> Редактирование профиля каталога)* Вы можете скомпилировать свой собственный каталог оборудования, который будет отображать только тот набор модулей, который используется Вами в структурах контроллеров. Двойным щелчком на панели заголовка Вы можете "пристыковать" каталог к правому краю окна станции или вновь освободить его.

Таблица конфигурации (Configuration table)

Утилита для конфигурирования оборудования Hardware Configuration работает с таблицами, каждая из которых представляет монтажную стойку, модуль или DP-станцию. Таблица конфигурации показывает слоты с модулями, установленными в них или свойства модулей, такие как адреса или порядковые номера. Двойной щелчок на строке модуля открывает окно свойств модуля (properties), с помощью этого окна можно задать параметры модуля.

2.3.1 Конфигурирование модулей

Конфигурирование начинается с выбора и переноса с помощью манипулятора "мышь" упомянутым выше методом "drag-n-drop" монтажной шины из каталога, например, "SIMATIC 300" или "RACK 300" в верхнюю половину окна станции. Пустая таблица конфигурации для центральной стойки отображается в окне. Теперь выберите требуемые модули из каталога модулей и перенесите тем же способом в подходящие слоты. Символ, говорящий о невозможности назначения данного слота "No Parking" ("Нет установки") для выбранного модуля, появится при попытке назначения уже назначенного слота.

В случае однорядной S7-300 станции слот 3 оставляется пустым: он резервируется для интерфейсного модуля для связи со стойкой расширения.

Вы можете сгенерировать таблицу конфигурации для другой стойки переносом с помощью мыши выбранной монтажной стойки из каталога в окно станции. В системах S7-400 несоединенной стойке (более точно: соответствующему приемному интерфейсному модулю) назначается интерфейс с помощью вкладки "Link" (соединение) в окне свойств ("Properties") передающего ("Send") IM. Для этого выберите модуль, затем опции меню: *Edit -> Object Properties (Правка -> Свойства объекта*).

Конфигурирование станций распределенных I/O модулей описано в главе 20.4.2 "Конфигурирование распределенных I/O".

2.3.2 Адресация модулей

При конфигурировании модулей утилита конфигурирования оборудования Hardware Configuration автоматически назначает начальные адреса модулей.

Вы можете видеть эти адреса в нижней части окна станции в свойствах объекта для соответствующих модулей. Для S7-400 CPU и S7-300 CPU с встроенным DP-интерфейсом Вы можете изменять адреса модулей. При этом необходимо учитывать правила адресации для систем S7-400 и S7-300, также как и диапазоны адресации для отдельных модулей.

Существуют модули, имеющие и входы и выходы, для которых Вы можете (теоретически) резервировать различные начальные адреса. При этом необходимо учитывать специальную информацию, предлагаемую в руководствах по использованию этих изделий; подавляющее большинство функциональных и коммуникационных модулей требуют использовать одинаковые начальные адреса для входов и выходов.

При назначении начальных адресов модулям для системы S7-400, Вы также можете выполнить назначение для дополнительного образа процесса. Если в центральной стойке используется более чем один процессор, то автоматически устанавливается мультипроцессорный режим, и Вы должны назначить модуль для CPU.

С помощью опций: *View -> Address Overview (Вид -> Обзор адресов)* в появившемся окне Вы можете получить информацию о текущей адресации всех модулей для выбранного CPU.

Модули на шине MPI или на коммуникационных шинах имеют MPI-адрес. Вы можете изменять этот адрес. Тем не менее, необходимо помнить, что новые MPI-адреса вступят в силу только после того, как данные конфигурации будут пересланы в CPU.

Символы для адресов, назначенных пользователем

Вы можете использовать утилиту для конфигурирования оборудования Hardware Configuration для назначения символов (имен) входам и выходам, которые заносятся в таблицу символов (Symbol Table).

После конфигурирования и адресации дискретных и аналоговых модулей Вы должны сохранить данные станции. После этого Вы должны выбрать модуль (строку в таблице) и с помощью опций: *Edit -> Symbols (Правка -> Символы)* открыть окно, в котором Вы сможете назначить символы, типы данных и комментарии к абсолютному адресу для каждого канала (побитно для дискретных модулей и пословно для аналоговых модулей).

Кнопка "Add Symbol" ("Назначить символ") служит для замены абсолютной адресации без символов на абсолютную адресацию с символами. Кнопка "Apply" ("Применить") позволяет занести символы в таблицу символов (Symbol Table). Кнопкой "OK" закрывают окно диалога.

2.3.3 Параметризация модулей

При назначении параметров модулям определяются их свойства. Задавать параметры модуля необходимо, только когда Вы хотите изменить параметры, заданные по умолчанию. Для параметризации модуля требуется, чтобы он был в таблице конфигурации.

Для редактирования свойства модуля открываются или двойным щелчком на модуле в таблице конфигурации, или выбором в таблице строки с модулем с

последующим выбором опций меню: *Edit -> Object Properties (Правка -> Свойства объекта*). В раскрывшемся диалоговом окне Вы найдете несколько вкладок с определяемыми параметрами. При использовании этого метода для параметризации CPU Вы можете задавать характеристики выполнения Вашей программы пользователя.

Некоторые модули позволяют устанавливать их параметры в процессе выполнения программы с помощью программы пользователя посредством системных функций SFC 55 WR_PARM, SFC 56 WR_DPARM и SFC 57 PARM_MOD.

2.3.4 Объединение в сеть модулей посредством MPI

Вы должны определить узлы для MPI-подсети в свойствах модулей (Module Properties). Для этого выберите в таблице конфигурации CPU или интерфейсную плату MPI, если она установлена, и откройте свойства устройства, используя опции меню: *Edit -> Object Properties (Правка -> Свойства объекта*). В появившемся окне диалога на вкладке "General" ("Общие") Вы найдете кнопку "Properties" (свойства) на панели "Interface" (интерфейс). Щелкнув на этой кнопке, Вы откроете другое окно диалога с вкладкой "Parameter" (параметр), где расположены данные подсети.

Таким образом удобно также задавать MPI-адрес, который Вы должны установить для данного CPU. Надо заметить, что для старших S7-300 CPU, модулей FM и CP MPI-адрес для MPI-соединения задается автоматически, исходя из CPU.

Старший MPI-адрес должен быть больше или равным старшему MPI-адресу, назначенному в подсети (примите во внимание автоматическое назначение для FM и CP!). Он должен иметь одинаковое значение для всех узлов подсети.

Совет: если у Вас имеется несколько станций с CPU одинакового типа, то для CPU в разных станциях назначьте различные имена (идентификаторы). По умолчанию они все имеют имя "CPUxxx(1)", так что в подсети они могут различаться только по их MPI-адресам. Если Вы не желаете сами назначать имена для CPU, Вы можете изменить номера в скобках, т.е. изменить имя "CPUxxx(1)" на имя "CPUxxx(n)", где "n" равно MPI-адресу.

При назначении MPI-адреса примите во внимание возможность подключения в дальнейшем к MPI-сети программатора PG или панели оператора (OP) для целей управления и технического обслуживания. Вы должны подключить постоянно установленные программаторы PG или панели оператора (OP) непосредственно к MPI-сети; для подключения устройств с помощью отвода (spur-line) предназначен специальный разъем - MPI-коннектор с резьбовым соединением.

Совет: зарезервируйте адрес 0 для программатора обслуживания, адрес 1 - для сервисной панели OP и адрес 3 - для сменного CPU (в соответствии с адресацией, принятой по умолчанию).

2.3.5 Режимы Monitor (мониторинг) и Modify (обновление) в модулях

С помощью утилиты для конфигурирования оборудования Hardware Configuration Вы можете выполнить проверку монтажа установки без программы пользователя. Для этого требуется, чтобы программатор был интерактивно (online) подключен к станции и конфигурация была сохранена, скомпилирована и загружена в CPU. После выполнения выше указанных условий Вы можете вызывать каждый дискретный и аналоговый модуль. Выбрав модуль, с помощью опций меню: *PLC -> Monitor/Modify (PLC -> Мониторинг/Обновление*) установите соответствующие режимы работы и условия запуска.

С помощью кнопки "Status Value" (значение состояния) утилита для конфигурирования оборудования Hardware Configuration покажет Вам состояние сигналов или каналов модулей. С помощью кнопки "Modify Value" (измененное значение) производится запись в модуль значений, записанных в одноименной колонке "Modify Value".

Если активен элемент управления checkbox "I/O Display" (отображение I/O), то вместо входов/выходов образа процесса будут отображаться состояние периферийных входов/выходов (память модуля). Если активен элемент управления checkbox "Enable Periph Outputs" (Разблокировка периферийных выходов), то отменяется блокировка выходных модулей, если CPU находится в режиме STOP (см. раздел 2.7.5 "Разблокировка периферийных выходов").

Вы можете найти и другие способы мониторинга и обновления входов и выходов в разделах 2.7.3 "Разблокировка периферийных выходов" и 2.7.4 "изменение переменной".

2.4 Конфигурирование сети (Network)

Основой коммуникаций в SIMATIC является объединение в сеть S7-станций. Для организации сети требуется наличие подсетей и модулей с коммуникационными свойствами в станциях. Вы можете создавать подсети и станции внутри иерархической структуры проекта посредством утилиты SIMATIC Manager. После этого Вы можете добавлять модули с коммуникационными свойствами (такие как CPU и CP), используя утилиту для конфигурирования оборудования Hardware Configuration; одновременно Вы можете назначать подсети коммуникационные интерфейсы этих модулей. Затем Вы можете установить коммуникационные отношения - соединения (connection) между этими модулями посредством утилиты конфигурирования сети Network Configuration в таблице соединений (connection table).

Утилита конфигурирования сети Network Configuration позволяет графически представить и документировать сконфигурированные сети и их узлы. С помощью утилиты Network Configuration Вы можете также создать все необходимые подсети и станции; затем Вы должны назначить станции в подсетях и параметризировать свойства узлов ("node properties") для модулей с коммуникационными свойствами.

Для установления соединений (connections) посредством утилиты конфигурирования сети Network Configuration Вы можете действовать по следующему плану:

- Откройте объект MPI-подсеть, созданную стандартным способом в каталоге проекта. Если она отсутствует, просто создайте новую подсеть с помощью опций: *Insert -> Subnet (Вставка -> Подсеть)*.
- С помощью утилиты конфигурирования сети Network Configuration создайте необходимые станции и, если требуется, другие подсети.
- Откройте объекты station (станции) и снабдите их модулями с коммуникационными свойствами.
- Соедините модули посредством определенных подсетей.
- Настройте параметры сети, если это необходимо.
- Задайте коммуникационные соединения (communication connections) в таблице соединений (connection table), если это необходимо.

Вы можете также сконфигурировать связь через глобальные данные с помощью утилиты конфигурирования сети Network Configuration: выберите подсеть MPI и затем опции: *Options -> Define Global Data (Опции -> Определить глобальные данные)* (см. раздел 20.5 "Связь посредством глобальных данных").

Опции меню: *Network -> Save (Сеть -> Сохранить)* позволяют сохранить промежуточные результаты конфигурирования сети. Вы можете проверить корректность конфигурации сети посредством опций меню: *Network -> Consistency Check (Сеть -> Проверка корректности).*

Закрытие процесса конфигурирования сети производится посредством выбора опций: *Network -> Save and Compile (Сеть -> Сохранить и скомпилировать).*

Окно Network (Сеть)

Чтобы запустить утилиту конфигурирования сети Network Configuration Вы должны создать проект. Вместе с проектом утилита SIMATIC Manager автоматически создает MPI-подсеть.

Двойным щелчком на этом объекте или на любой другой подсети запускается утилита конфигурирования сети Network Configuration. Вы также можете запустить эту утилиту, если откроете объект *Connections (Соединения)* в каталоге *CPU*.

Ниже на рис. 2.5 представлено окно утилиты конфигурирования сети Network Configuration, отображающее все ранее созданные подсети и станции (узлы) проекта с сконфигурированными соединениями (connections).

Таблица соединений (connection table) показана в нижней части окна. Она появляется, если в верхней части окна выделен модуль, обладающий коммуникационными свойствами, например, S7-400 CPU.

Второе окно отображает каталог объектов сети с выбранными SIMATIC станциями, подсетями и DP-станциями. Вы можете скрыть каталог или вновь открыть его с помощью опций: *View -> Catalog (Вид -> Каталог).* Двойным щелчком на панели заголовка Вы можете "пристыковать" каталог к правому краю окна станции или вновь освободить его.

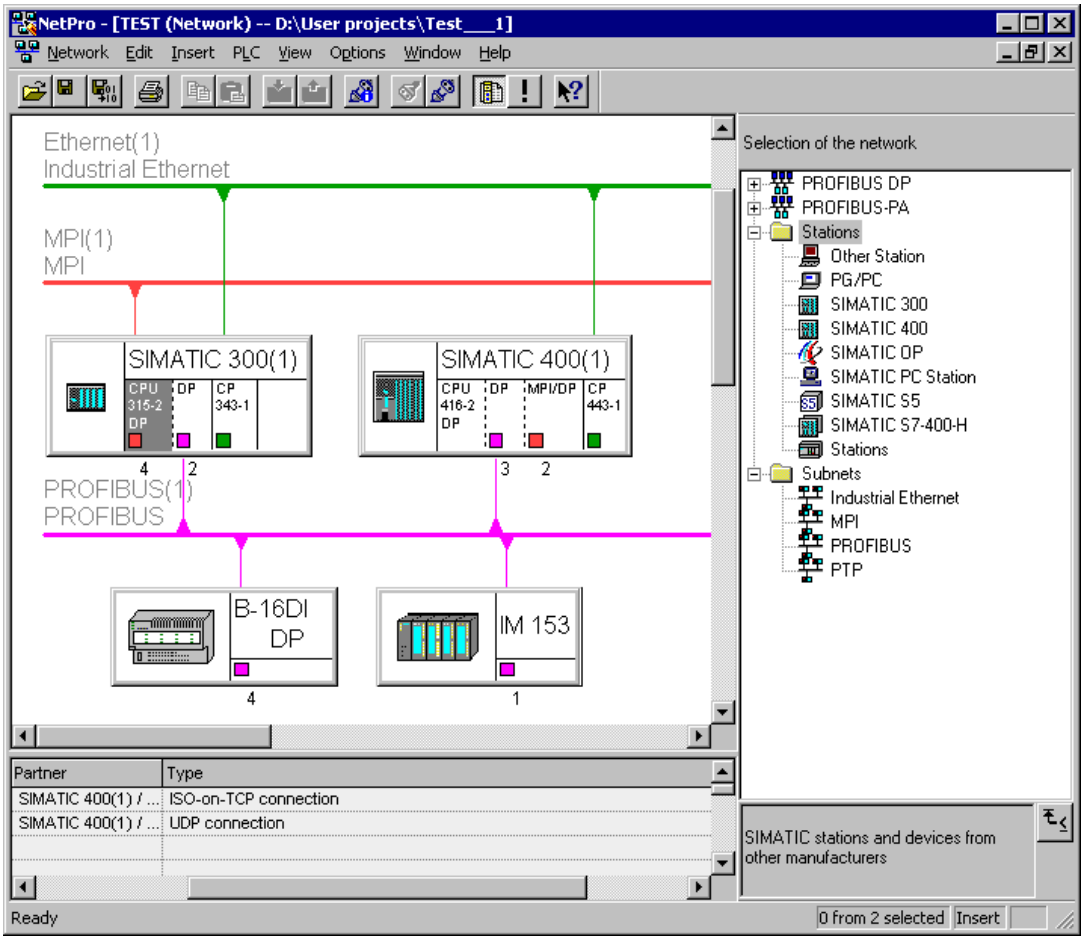


Рис. 2.5 Пример окна утилиты конфигурирования сети Network Configuration

С помощью опций: *View -> Zoom In (Вид -> Увеличить масштаб), View -> Zoom Out (Вид -> Уменьшить масштаб)* и *View -> Zoom Factor (Вид -> Коэффициент масштабирования)* Вы можете настраивать четкость графического представления конфигурации сети.

2.4.1 Конфигурирование графического представления сети (Network View)

Выбор и монтаж компонентов

Конфигурирование сети начинается с выбора типа подсети с помощью манипулятора "мышь" из каталога и переноса этого объекта в окно сети. Подсеть в этом окне представляется горизонтальной линией. Запрещенные позиции для нее отображаются рядом с указателем в виде знака запрета.

Тем же способом Вы должны выбрать и установить станции, сначала без связи с подсетью. Сначала станции "пустые". Двойной щелчок на станции запускает утилиту конфигурирования оборудования Hardware configuration, с помощью которой можно сконфигурировать станцию или, по крайней мере, модуль (модули) для объединения в сеть. После этого необходимо сохранить станцию и вернуться к конфигурированию сети (Network Configuration).

Интерфейс модуля, обладающего коммуникационными свойствами, отображается в окне утилиты конфигурирования сети, как маленький прямоугольник под изображением модуля. Щелкните на нем, удержите кнопку мыши и "перетащите" к требуемой подсети. Соединение с подсетью выглядит на схеме как вертикальная линия.

Выполните такие же операции со всеми другими узлами.

Вы можете перемещать созданные подсети и станции в окне сети. Таким способом Вы можете представить конфигурацию Вашего оборудования визуально.

Установка коммуникационных свойств

После создания графического представления сети, Вы должны параметризировать подсети. Для этого выберите подсети и с помощью опций меню: *Edit -> Object Properties (Правка -> Свойства объекта*) откройте окно свойств. На вкладке окна "General" (Общие) находится идентификатор S7-подсети (ID). ID состоит из двух шестнадцатеричных чисел - номера проекта и номера подсети. Данный ID S7-подсети необходим при переходе в интерактивный режим (online) с программатором без соответствующего проекта, чтобы подключиться к другим узлам посредством подсети. Вы можете установить свойства сети (network properties) на вкладке "Network Settings" ("установки сети"), например, скорость передачи данных (data transfer rate) или старший адрес узла (highest node address).

Выбрав для узла подключение к сети (network connection), Вы можете определить свойства сети с помощью опций меню: *Edit -> Object Properties (Правка -> Свойства объекта)*, например, адрес узла (node address) и подсеть (subnet), к которой он подключен, или же Вы можете создать новую подсеть.

На вкладке "Interfaces" ("Интерфейсы") окна свойств станции Вы можете видеть все модули с коммуникационными свойствами с адресами узлов и используемыми типами подсетей.

Аналогичным образом Вы можете определять свойства модулей узлов (задавая параметры в окнах ввода утилиты для конфигурирования оборудования Hardware Configuration).

2.4.2 Конфигурирование системы ведущего DP-устройства с помощью утилиты конфигурирования сети Network Configuration

Вы можете также использовать утилиту конфигурирования сети Network Configuration для конфигурирования системы распределенных I/O. Используя

опции меню: *View -> with DP Slaves (Вид -> с ведомыми DP-устройствами),* Вы можете отобразить или скрыть изображение ведомых (slave) DP-устройств в графическом представлении сети (Network View).

Для конфигурирования системы ведущего DP-устройства Вам требуются:

- Подсеть PROFIBUS (если подсеть отсутствует, "перетащите" объект подсеть PROFIBUS в окно сети из каталога объектов сети).
- Ведущее DP-устройство (master) в станции (если устройство отсутствует, "перетащите" станцию в окно сети из каталога объектов сети, откройте станцию и выберите DP-устройство с помощью утилиты для конфигурирования оборудования Hardware Configuration либо как встроенное в CPU устройство, либо как автономный модуль).
- Соединение (connection) для DP-устройства с подсетью PROFIBUS (или выделите подсеть с помощью утилиты для конфигурирования оборудования Hardware Configuration, или щелкните кнопкой мыши на соединении для ведущего DP-устройства (master) при использовании утилиты для конфигурирования сети Network Configuration и "перетащите" объект на подсеть PROFIBUS).

В окне сети выберите ведущее DP-устройство (master), которому должно быть назначено ведомое DP-устройство (slave). Найдите ведомое DP-устройство (slave) в каталоге объектов сети в "PROFIBUS" в соответствующем подкаталоге, "перетащите" объект в окно сети и задайте свойства в появившемся окне.

Параметризируйте ведомое DP-устройство (slave) сначала выделив его и далее используя опции меню: *Edit -> Open Object (Правка -> Открыть объект)*. После этого будет запущена утилита для конфигурирования оборудования Hardware Configuration. Теперь Вы можете выполнить адресацию данных пользователя или, в случае использования модульных ведомых устройств (slave), выберите I/O модули (см. раздел 2.3 "Конфигурирование станций").

Вы сможете подключить интеллектуальное ведомое DP-устройство к подсети, только если Вы предварительно создали его (см. раздел 20.4.2 "Конфигурирование распределенных I/O"). В каталоге объектов сети типы интеллектуальных DP-устройств (slave) находятся под "Already created stations" ("Готовые станции"). При выбранном ведущем DP-устройстве (master) Вы можете "перетащить" объект в окно сети и задать свойства в появившемся окне свойств объекта (как в утилите для конфигурирования оборудования Hardware Configuration).

С помощью выбора опций меню: *View -> Highlight -> Master System (Вид -> Выделить -> Система ведущего DP-устройства)* Вы можете выделять назначения узлов системы ведущего DP-устройства (DP-master system). При этом Вы должны сначала выбрать (выделить) ведущее (master) или ведомое (slave) устройство этой системы.

2.4.3 Конфигурирование соединений (Connections)

Соединение (Connection) описывает коммуникационные отношения между двумя устройствами. Соединения должны быть сконфигурированы

- если Вы хотите установить SFB-коммуникации между двумя SIMATIC S7-устройствами ("Communications via configured connections" - "коммуникации посредством сконфигурированных соединений") или
- если коммуникационный партнер не является SIMATIC S7-устройством.

Примечание: Вам нет необходимости конфигурировать соединение для прямого интерактивного (online) соединения программатора с MPI-сетью в целях программирования или отладки. Если же необходимо установить связь программатора с другими узлами, скомпонованными в других связанных подсетях, Вы должны будете сконфигурировать соединение программатора. Для этого двойным щелчком выберите в каталоге объектов сети (Network Object Catalog) объект PG/PC в каталоге Stations (Станции), откройте PG/PC двойным щелчком в окне сети (network), выберите интерфейс и назначьте его для подсети.

Connection table (таблица соединений)

Коммуникационные соединения конфигурируются в таблице соединений (Connection table).

Требование: Вы должны создать проект со всеми станциями, которые могут обмениваться данными друг с другом, также Вы должны назначить модули с коммуникационными свойствами для подсети.

Объект *Connections (Соединения)* в каталоге *CPU* представляет таблицу соединений (Connection table). Двойной щелчок на объекте *Connections (Соединения)* запускает утилиту конфигурирования сети Network Configuration, также как и двойной щелчок на подсети в каталоге проекта.

Для конфигурирования соединений выберите S7-400 CPU в утилите конфигурирования сети Network Configuration. В нижней части окна сети располагается таблица соединений (Connection table) (см. пример: таблица 2.1). Если таблица не видна, то поместите указатель мыши на нижний край окна и, когда изменится его форма, перемещайте край окна вверх.

Таблица 2.1 Пример таблицы соединений (Connection table)

Local ID (локальный ID)	Partner ID (ID партнера)	Partner (партнер)	Type (тип)	Active Connection Buildup (активация соединения)	Send Operating State Message (посылать сообщение о рабочем состоянии)
1	1	Station 416/CPU416(5)	S7 connection	Yes (да)	No (нет)
2	2	Station 416/CPU416(5)	S7 connection	Yes (да)	No (нет)
3		Station 315/CPU315(7)	S7 connection	Yes (да)	No (нет)
4	1	Station 417/CPU414(4)	S7 connection	Yes (да)	No (нет)

Вы можете вводить новые коммуникационные соединения с помощью опций: *Insert -> New Connections (Вставка -> Новое соединение)* или дважды щелкнув на пустой строке.

Вы должны создавать соединение (connection) для каждого активного ("active") CPU. Надо заметить, что для S7-300 CPU Вы не сможете создать таблицу соединений; S7-300 CPU могут быть только "пассивными" ("passive") партнерами в S7-соединениях.

В окне "New Connection" ("Новое соединение") Вы можете выбирать коммуникационных партнеров в диалоговых окнах "Station" ("Станция") и "Module" ("Модуль") (см. рис. 2.6); выбираемые станция или модуль при этом должны уже существовать. Вы также можете определять тип соединения.

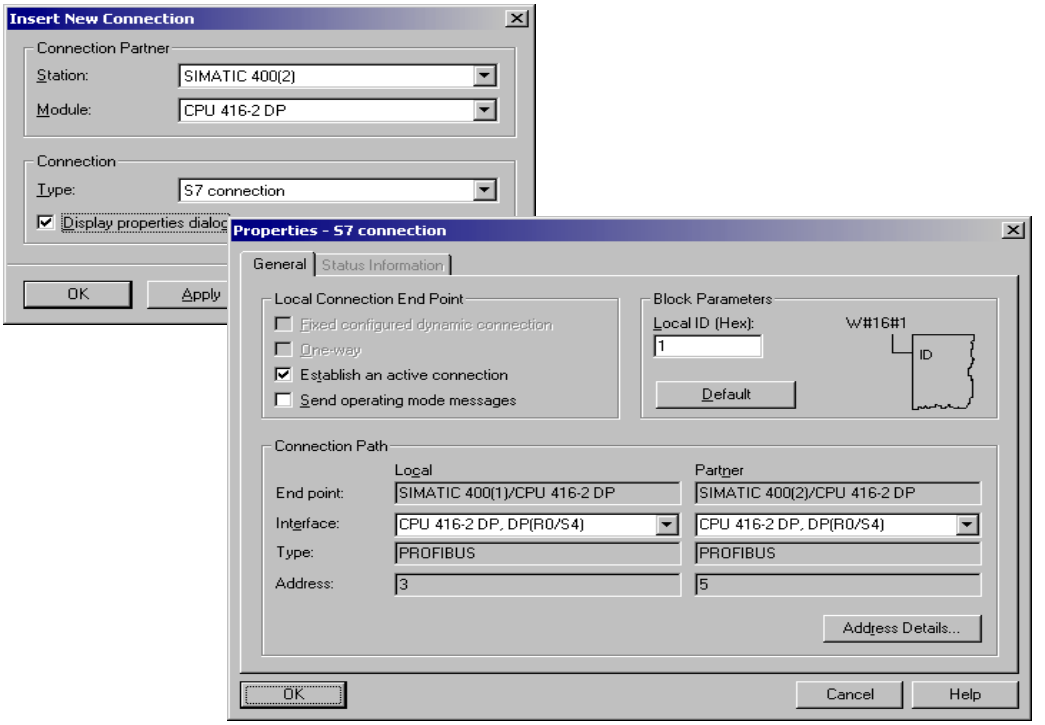


Рис. 2.6 Конфигурирование коммуникационных соединений

Если необходимо изменить дополнительные свойства соединения, активизируйте элемент управления check box "Show Properties Dialog" ("Показать диалоговое окно свойств").

Таблица соединений содержит все данные сконфигурированных соединений. Для точного отображения этих данных используйте опции меню: *View -> Display Columns (Вид -> Отобразить столбцы)*, после чего выберите интересующую Вас информацию.

Connection ID (идентификатор соединения)

Число устанавливаемых соединений определяется типом CPU. STEP 7 устанавливает ID для каждого соединения и для каждого партнера. Такая спецификация Вам потребуется при использовании коммуникационных блоков в Вашей программе.

Local ID (Локальный ID)

Вы можете модифицировать локальный ID (столбец **local ID** - ID соединения открытого в настоящий момент модуля). Такая необходимость может возникнуть, если Вы уже запрограммировали коммуникационные блоки и хотите использовать в них определенный локальный ID для соединения.

Вы должны ввести значение нового локального ID (local ID) в виде шестнадцатеричного числа. Оно может лежать внутри следующих диапазонов значений, в зависимости от типа соединения, и не должно быть к текущему моменту времени использовано:

- Диапазон значений для S7-соединений:

  0001_{16} … $0FFF_{16}$

- Диапазон значений для PtP-соединений:

  1000_{16} … 1400_{16}

Partner ID (ID партнера)

Вы можете также изменить ID партнера (столбец **partner ID**), перейдя к таблице соединений CPU партнера с последующим изменением локального ID (local ID); для этого необходимо выбрать строку с интересующим Вас соединением и затем использовать опции меню: *Edit -> Object Properties (Правка -> Свойства объекта)*. Если STEP 7 не дает изменить ID партнера (partner ID), то это означает, что имеет место односторонняя связь (соединение) (см. ниже).

Partner (Партнер)

В данном столбце отображены коммуникационные партнеры. Если необходимо зарезервировать коммуникационный ресурс без указания имени устройства партнера, введите в окне ввода "Station" ("Станция") значение "unspecified" ("не определено").

При односторонней связи (**one-way connection**) инициализация коммуникаций производится со стороны только одного партнера; например: SFB-коммуникации между S7-400 и S7-300 CPU. Хотя коммуникационные SFB-функции не доступны для S7-300, в данном случае обмен данными может обеспечиваться S7-400 CPU с помощью SFB 14 GET и SFB 15 PUT. Для такого соединения не нужно выполнения программы пользователя в S7-300, так как управление обменом данными обеспечивается операционной системой.

Для односторонней связи (one-way connection) соединение конфигурируется с помощью таблицы соединений (connection table) "активного" ("active") CPU. Только после этого STEP 7 назначает локальный ID ("Local ID"). Вы должны загружать это соединение только в локальной станции.

При двусторонней связи (**two-way connection**) оба партнера могут проявлять коммуникационную активность; это могут быть, например, два S7-400 CPU. В данном случае обмен данными может обеспечиваться, например, с помощью SFB 8 BSEND и SFB 9 BRCV.

Для двусторонней связи (two-way connection) соединение конфигурируется только один раз для одного из двух партнеров. После этого STEP 7 назначает локальный ID ("Local ID"), ID партнера ("Partner ID") и генерирует коммуникационные данные для обеих станций. Вы должны загружать каждого партнера с его собственной таблицей соединений.

Type (Тип)

В данном столбце таблицы соединений устанавливается тип соединения.

Базовый пакет STEP 7 обеспечивает для конфигурирования сети следующие типы соединений (connection type):

Соединение **PtP connection** ("Point-to-point", "точка к точке") применяется для подсети PTP (процедуры 3964 (R) и RK 512) для SFB-коммуникаций. Соединение PtP служит для последовательной связи между двумя партнерами. Это могут быть два устройства SIMATIC S7 с соответствующими коммуникационными процессорами CP или одно устройство SIMATIC S7 и одно устройство стороннего производителя (не из семейства SIMATIC), например, принтер или считыватель штрих-кода.

Соединение **S7 connection** применяется для подсетей MPI, PROFIBUS и Industrial Ethernet с коммуникационными SFB-функциями. Соединение S7 обеспечивает связь между устройствами SIMATIC S7, включая программаторы PG и устройства HMI (устройства человеко-машинного интерфейса). Посредством соединения S7 производится обмен данными или выполняются функции управления или программирования.

Соединение **Fault-tolerant S7 connection** (отказоустойчивое соединение S7) применяется для подсетей PROFIBUS и Industrial Ethernet с коммуникационными SFB-функциями. Отказоустойчивое соединение S7 устанавливается между отказоустойчивыми устройствами SIMATIC S7 и может также устанавливаться для соответствующим образом оснащенного ПК.

Опционные пакеты "NCM S7 for PROFIBUS" и "NCM S7 for Industrial Ethernet" позволяют производить параметризацию коммуникационных процессоров CP.

В зависимости от установленного программного обеспечения NCM у Вас будут дополнительные типы соединений: FMS-соединение, FDL-соединение, ISO transport-соединение, TCP-соединение, ISO-on-TCP-соединение, UDP-соединение и E-mail-соединение.

Установка активного соединения (Active Connection Buildup)

Перед тем, как начать передачу данных, необходимо установить соединение (инициализировать). Если коммуникационные партнеры имеют такую возможность, то Вы можете задать устройство для установления соединения. Делается это с помощью элемента управления check box "Active connection buildup" ("Установка активного соединения") в окне свойств соединения: выделите соединение, затем выберите опции меню: *Edit -> Object Properties (Правка -> Свойства объекта*).

Посылка сообщений о рабочем состоянии (Sending operating state messages)

Коммуникационные партнеры с сконфигурированной двусторонней связью могут обмениваться сообщениями о рабочем состоянии. Если локальный узел должен посылать сообщения о своем рабочем состоянии, активируйте соответствующий элемент управления check box в окне свойств соединения. В программе пользователя CPU партнера эти сообщения могут приниматься с помощью SFB 23 USTATUS.

Расположение соединения (Connection Path)

Окно свойств соединения отображает конечные пункты соединения и подсети, через которые соединение осуществляется, в виде (адреса) расположения соединения (Connection Path). Если присутствует несколько подсетей на выбор, то STEP 7 выбирает их в следующей последовательности: сначала Industrial Ethernet, затем Industrial Ethernet/TCP-IP, затем MPI и, наконец, PROFIBUS.

Станция и CPU, через которые соединение осуществляется, отображаются как конечные пункты соединения. Модули с коммуникационными свойствами приводятся в списке в окне с пометкой "Interface" ("Интерфейс") с указанием номера стойки и номера слота. Если оба CPU расположены в одной стойке (например, два S7-400 CPU в многопроцессорном режиме), в окне отразится запись "PLC-internal" ("внутри PLC").

В окне с пометкой "Type" ("Тип") Вы можете выбирать подсети, через которые соединение должно осуществляться. Если оба коммуникационных партнера, например, подключены к одной MPI-подсети и к одной PROFIBUS-подсети, то в окне Вы увидите "MPI". Вы можете изменить эту спецификацию на "PROFIBUS", и STEP 7 автоматически примет остальные установки. После этого Вы увидите адрес MPI или адрес PROFIBUS для узла в окне "Address" ("Адрес").

Соединения между проектами (Connections between projects)

Для обмена данными между двумя S7 модулями, принадлежащими различным SIMATIC-проектам, Вы должны ввести значение "unspecified" ("не определено") для коммуникационного партнера в таблице соединений (в локальной станции в обоих проектах).

Убедитесь, что данные таблицы соединений согласованы, так как STEP 7 не проверяет согласованность этих данных самостоятельно. После сохранения и компиляции данных таблицы соединений Вы должны загрузить их в локальные станции в каждом проекте.

Соединения с не S7-станцией (Connection to non-S7 station)

В проекте Вы можете также определять станции, не относящиеся к S7-станциям, в качестве коммуникационных партнеров:

- Другие станции (устройства сторонних [не Siemens] производителей, а также S7-станции в других проектах)
- Программаторы PG / компьютеры (ПК)
- SIMATIC S5-станции

Необходимые условия для таких соединений заключаются в том, что другая (не S7) станция должна существовать как объект в каталоге проекта, кроме того должно быть выполнено подключение этой станции к соответствующей подсети в свойствах станции (например, выберите станцию с помощью утилиты конфигурирования сети Network Configuration, затем выберите опции: *Edit -> Object Properties* [*Правка -> Свойства объекта*] и подключите станцию к требуемой подсети на вкладке "Interfaces" ["Интерфейсы"]).

2.4.4 Переходы между подсетями (Network Transitions)

Если программатор подключен к подсети, он может иметь доступ ко всем узлам данной подсети. При этом из одной точки подключения Вы можете программировать и отлаживать программы для всех S7-станций, подключенных к MPI-сети. Если какая-либо S7-станция подключена также к другой подсети, такой как PROFIBUS, программатор может также иметь доступ ко всем узлам и этой подсети. Для этого должно выполняться требование, чтобы станция с переходом между подсетями имела возможность для программирования канала передачи фреймов сообщений.

Когда конфигурирование сети завершено, для станций с переходом между подсетями автоматически генерируются таблицы маршрутизации (routing table), содержащие всю необходимую информацию. Все доступные коммуникационные партнеры должны быть сконфигурированы в сети автоматизируемой установки в S7-проекте и должны "знать", к каким станциям имеется доступ и с помощью каких подсетей и переходов между подсетями.

Если необходимо для программатора, подключенного к подсети, обеспечить доступ ко всем узлам данной подсети из одной точки подключения, то Вы должны сконфигурировать соединение (точку подключения). Вы должны для этого ввести "placeholder" ("местодержатель"), PG/ПК станцию из каталога сетевых объектов (Network Object Catalog) в конфигурацию сети в соответствующей подсети. После этого PG/ПК станция должна быть сконфигурирована в каждой подсети, к которой необходимо будет подключать программатор PG.

Во время работы Вы будете подключать PG к подсети и выбирать опции меню: *PLC -> Assign PG/PC (PLC -> Назначить PG/ПК)*. Это позволяет настраивать интерфейсы программатора для работы с выбранной подсетью. Перед отключением PG от подсети требуется выбрать опции меню: *PLC -> Undo PG/PC Assignment (PLC -> Отменить назначение PG/ПК)*.

Если необходимо перейти в интерактивный (online) режим с программатором, в котором нет необходимого проекта, Вам потребуется ID S7-подсети для доступа к сети. ID S7-подсети содержит два номера: номер проекта и номер подсети. Вы можете получить ID подсети из данных конфигурации сети. Для этого выберите сначала подсеть, затем - опции меню: *Edit -> Object Properties (Правка -> Свойства объекта)* на вкладке "General" ("Общие").

2.4.5 Загрузка таблицы соединений (Loading the Connection Data)

Для активации соединений Вы должны сохранить, скомпилировать и загрузить таблицу соединений ("connection table") в PLC (все таблицы соединений во все "активные" CPU).

Требование: Когда Вы открываете окно сети (network window), таблица соединений отражается на экране. Программатор является узлом подсети, с помощью которой могут быть загружены данные таблицы соединений в модули с коммуникационными свойствами. Все узлы подсети имеют уникальные адреса. Все модули, в которые должны быть переданы данные таблицы соединений, должны находиться в режиме STOP.

С помощью опций меню: *PLC -> Download -> … (PLC -> Загрузить -> …)* Вы можете передать данные таблицы соединений и данные конфигурации в доступные модули. В зависимости от того, какой объект и какие команды мены выбраны, Вы можете выбирать из следующего ряда опций:

-> *Selected Stations (Выбранные станции)*

-> *Selected and Partner Stations (Выбранные и станции партнера)*

-> *Selected Connections (Выбранные соединения)*

-> *Stations on Subnet (Станции в подсети)*

-> *Connections and Gateways (Соединения и шлюзы)*

Для удаления всех данных таблицы соединений в программируемом модуле загрузите в него пустую таблицу соединений (connection table).

Скомпилированные данные таблицы соединений также являются частью системных данных (*System data*) в каталоге *Blocks*. Передача системных данных и последующий запуск CPU приводит к передаче данных таблицы соединений в модули с коммуникационными свойствами.

Для интерактивной (online) работы с помощью MPI для программатора не требуется дополнительного оборудования. Если же к сети Вы подключаете ПК или подключаете PG к сетям Ethernet или PROFIBUS, то Вам потребуются соответствующие интерфейсные модули. Параметризация модулей производится с помощью приложения "Set PG/PC Interface" из панели управления Windows.

2.5 Создание S7-программ

2.5.1 Введение

Программа пользователя создается в каталоге (в объекте) *S7 Program.* Вы можете назначать этот объект в объекте CPU в структурной иерархии проекта, или вне зависимости от CPU. В свою очередь объект *S7 Program* включает в себя объект *Symbols (Символы)* и каталоги *Source Files (Исходные файлы)* и *Blocks (Блоки)* (см. рис. 2.7).

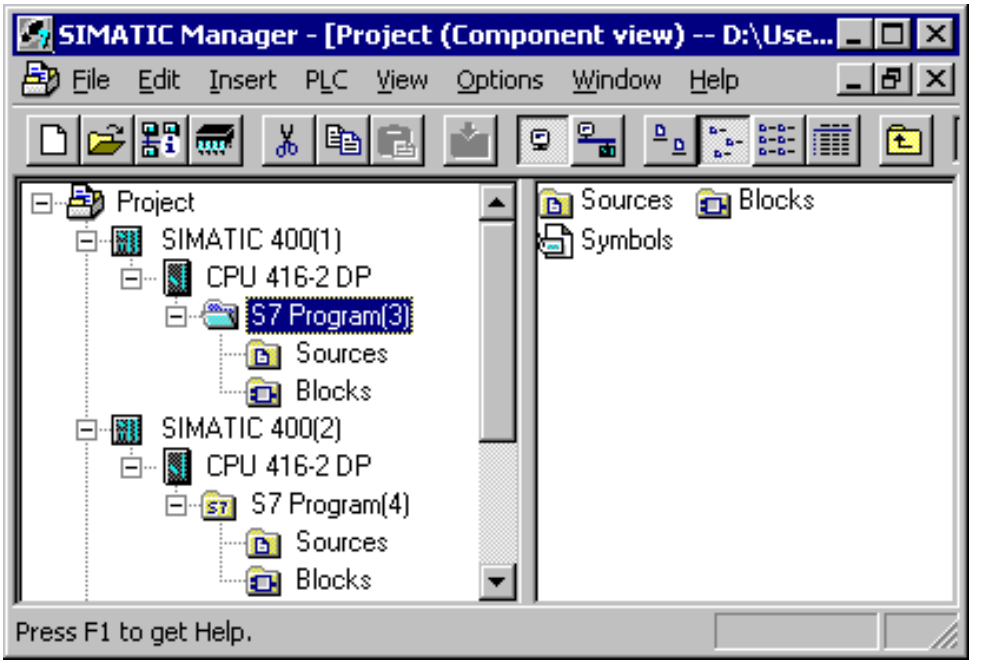


Рис. 2.7 Объекты, участвующие при генерации программы.

В случае создания программы путем написания исходных файлов ("**source-oriented**") Вы должны создать одну или несколько исходных программ и сохранить их в виде файлов в каталоге *Source Files (Исходные файлы)*. Исходные программы - это текстовые файлы формата ASCII, которые содержат операторы программы для одного или нескольких блоков, возможно даже целиком всю программу. Вы должны скомпилировать исходные программы; скомпилированные блоки программы помещаются в каталог *Blocks (Блоки).* Скомпилированные блоки содержат код MC7 и выполняются в S7 CPU.

В случае создания программы "инкрементным" путем ("**incremental**"), - методом добавления - Вы вводите программу блок за блоком. Вводимые блоки немедленно проверяются на наличие синтаксических ошибок. При поступлении команды на сохранение блок сначала компилируется, затем сохраняется в каталоге *Blocks (Блоки).* При создании программы данным методом Вы можете также редактировать блоки в интерактивном (online) режиме в CPU, даже во время рабочего режима.

В программе обрабатываются значения сигналов или значения адресов. Адрес - это, например, вход I1.0 (абсолютная адресация). С помощью таблицы символов **Symbol Table** в объекте Symbols, Вы можете назначить адресу символьное имя, например, "Switch motor on" ("Включение мотора") и после этого обращаться к этому адресу, используя данное символьное имя (символьная адресация). В свойствах автономного объекта *Blocks (Блоки)* Вы можете определить, каким способом будут адресоваться переменные в таблице символов (Symbol Table) после корректировки - абсолютным или символьным в уже скомпилированных блоках, согласно приоритету адресации (*address priority*).

Требования к памяти

Требования к памяти для скомпилированного блока можно найти в свойствах блока для этого при выбранном в SIMATIC Manager блоке, выберите опции меню: *Edit -> Object Properties (Правка -> Свойства объекта),* вкладка "General-Part 2" ("Общие - часть 2").

Вы можете узнать требования к памяти для Вашей программы в целом, если выбрать в SIMATIC Manager программу из объекта *Blocks (Блоки)* и затем выбрать опции меню: *Edit -> Object Properties (Правка -> Свойства объекта).* На вкладке *Blocks (Блоки)* Вы найдете размер программы в загрузочной памяти (load memory) и в рабочей памяти (work memory), так же как и число блоков каждого типа.

Системные данные не рассматриваются среди этой информации; они занимают дополнительное пространство в загрузочной памяти.

2.5.2 Таблица символов (Symbol Table)

В управляющей программе Вы имеете дело с адресами, т.е. с входами, с выходами, таймерами и блоками. Вы можете назначать абсолютные адреса (например, I1.0) или символьные адреса (например, Start signal [сигнал запуска]). При символьной адресации используются символьные имена. Это делает программу легко читаемой, благодаря тому, что символьные имена несут смысловую нагрузку.

При использовании символьной адресации различаются локальные *(local)* и глобальные *(global)* символы (символьные имена). Локальный *(local)* символ распознается только в блоке, в котором они определены. Поэтому при необходимости Вы можете использовать одинаковые локальные символьные имена в различных целях в разных блоках. Глобальный символ распознается в любом месте программы и имеет одинаковое значение во всех блоках программы. Вы должны определить глобальный символ в таблице символов (объект *Symbols* в каталоге *S7 Program*).

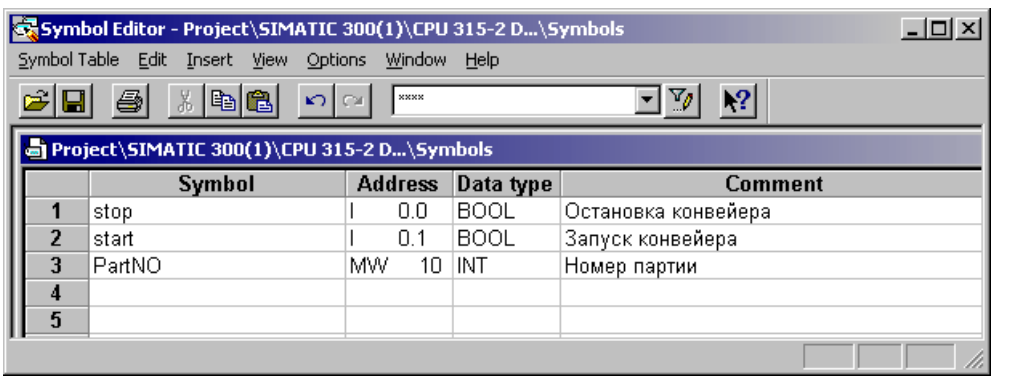
Глобальный символ начинается с символа алфавита и может иметь в длину до 24 символов. Глобальный символ может также содержать пробелы, специальные символы и национальные символы, например, такие как умляут. Исключения составляют символы 00 $_{hex}$, FF $_{hex}$ и кавычки ("). При программировании Вы должны заключать спецсимволы в кавычки. В скомпилированном блоке программный редактор отображает все глобальные символы в кавычках. Комментарий к символу может составлять в свою очередь запись из 80 символов.

В таблице символов Вы можете назначать имена следующим адресам и объектам:

- Входам I, выходам Q, периферийным входам PI и выходам PQ
- Меркерам M, таймерам T и счетчикам C
- Блокам кодов OB, FB, FC, SFC, SFB и блокам данных DB
- Типам данных, определенным пользователем, UDT
- Таблице переменных VAT

Адреса данных в блоках данных находятся среди локальных адресов; связанные символы определяются в разделе описаний (declaration section) блоков данных в случае глобальных блоков данных и в разделе описаний (declaration section) функциональных блоков в случае экземплярных блоков данных.

При создании S7-программ SIMATIC Manager создает также пустую таблицу символов *Symbols*. Вы можете открыть эту таблицу и определить глобальные символы и назначить их абсолютным адресам (рис. 2.8).



	Symbol	Address	Data type	Comment
1	stop	I 0.0	BOOL	Остановка конвейера
2	start	I 0.1	BOOL	Запуск конвейера
3	PartNO	MW 10	INT	Номер партии
4				
5				

Рис. 2.8 Пример таблицы символов Symbol Table

В S7-программе может быть только одна таблица символов *Symbols.*

Тип данных является частью определения символа. Он определяет особые свойства данных, в частности представление содержимого данных. Например, тип данных BOOL идентифицирует двоичную переменную, а тип данных INT обозначает переменную в цифровой форме, содержание которой определяется 16-битным целым числом. Для получения более подробной информации обратитесь к разделу 3.7 "Переменные и константы" и к разделу 24 "Типы данных", содержащим соответственно обзор и подробное описание типов данных в STEP 7.

В случае "инкрементного" программирования Вы создаете таблицу символов до ввода программы; Вы можете также добавить или скорректировать отдельные символы во время ввода программы. При создании программы путем, ориентированным на создание исходных текстов программы готовая таблица символов должна быть доступна к моменту компиляции программы.

Импорт, экспорт

Таблица символов может быть импортирована и экспортирована. Здесь "экспортируемый" файл означает созданный файл, содержащий данные Вашей таблицы символов. Здесь может быть как таблица символов целиком, так и отдельные строки таблицы. Для файла Вы можете выбирать следующие форматы данных: текстовый ASCII (с расширением *.asc), sequential assignment list (последовательный список выражений - с расширением *.seq), System Data Format (формат системных данных - с расширением *.sdf для Microsoft Access) и Data Interchange Format (формат обмена данными - с расширением *.dif для Microsoft Excel). Вы можете редактировать экспортируемый файл с помощью подходящего редактора. Вы можете также импортировать таблицу символов в одном из выше упомянутых форматов.

Специальные свойства объектов

Выбрав опции меню: *Edit -> Special Object Properties (Правка -> Специальные свойства объекта),* Вы можете установить атрибуты для каждого символа в таблице символов. Эти атрибуты или свойства используются для следующих целей:

- Для HMI функций для мониторинга с использованием WinCC
- Для конфигурирования коммуникаций
- Для конфигурирования сообщений
- Для мониторинга процесса посредством S7-PDIAG

Выбрав опции меню: *View -> Columns O, M, C, R (Вид -> Столбцы O, M, C, R),* Вы сделаете атрибуты видимыми. С помощью опций меню: *Options -> Customize (Опции -> Установки пользователя)* Вы можете установить, будут или нет специальные свойства объектов копироваться, и Вы сможете определить поведение при импортировании символов.

2.5.3 Редактор STL-программ (STL Program Editor)

Для создания программы пользователя в базовый пакет STEP 7 (STEP 7 Basic Package) включены редакторы для языков программирования LAD, FBD и STL. При работе с редактором STL-программы Вы можете вводить программу "инкрементно" (непосредственно) или генерировать исходный

текст программы и компилировать его позднее. На рис. 2.9 показаны возможные действия, связанные с созданием STL-программы.

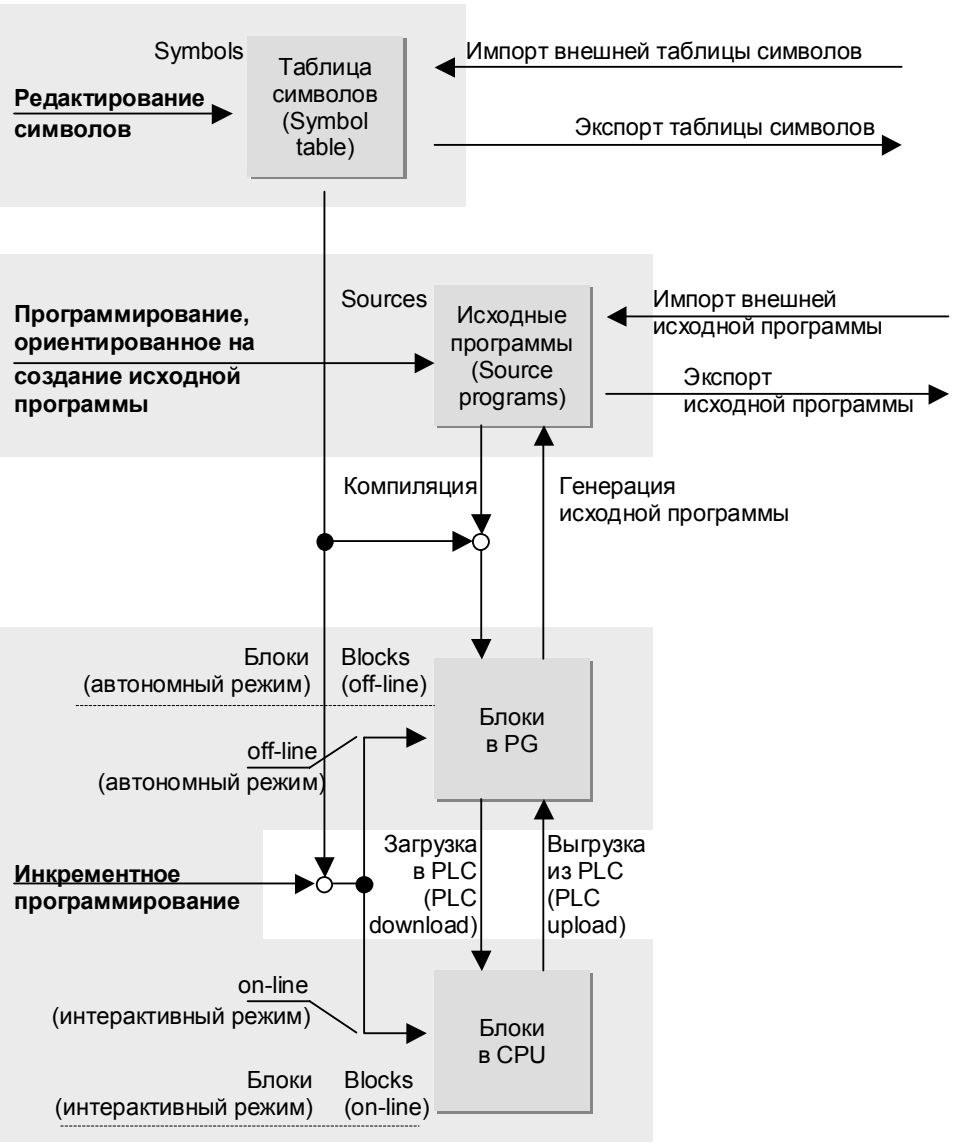


Рис. 2.9 Создание программы с использованием редактора STL Editor.

Если при инкрементном программировании используется символическая адресация глобальных адресов, то предварительно должно быть выполнено присвоение символических имен абсолютным адресам. Тем не менее, Вы можете вводить новые символы или изменять ранее присвоенные во время ввода программы. В случае программирования, ориентированного на создание исходной программы, к моменту компиляции программы таблица символов должна быть заполнена.

STL-блоки могут быть "декомпилированы" ("decompiled"), т.е. из MC7 кода может быть извлечен пригодный для чтения блок без автономной базы данных (offline database) (Вы можете прочитать любой блок из CPU, используя программатор PG без связанного проекта). Вдобавок, исходная STL- программа может быть восстановлена из любого скомпилированного блока.

Запуск редактора STL- программ

Вы можете получить доступ к редактору программ при открытии блока в SIMATIC Manager, например, двойным щелчком на автоматически сгенерированном символе для организационного блока OB1 или с помощью меню панели задач Windows: *Start -> Simatic -> STEP 7 -> LAD, STL, FBD - Program S7 Blocks.*

Вы можете задать свойства для редактора программ с помощью опций меню: *Options -> Customize (Опции -> Установки пользователя).* На вкладке "Editor" ("Редактор") выберите свойства, с которыми новый блок должен быть сгенерирован и отображен, такие, например, как язык создания, установки для комментариев и символов.

При открытии скомпилированного блока в каталоге *Blocks (Блоки) (например, двойным щелчком)* блок открывается для инкрементного программирования. Для программирования, ориентированного на создание исходных текстов программы, Вы должны открывать исходный файл программы в каталоге *Source files (Исходные файлы).*

Вы можете также создавать программу, используя попеременно то один метод, то другой метод программирования, т.е. некоторые блоки вводятся непосредственно, а другие блоки создаются с помощью исходных файлов. Также в программе можно вызывать отдельные блоки, созданные с использованием других языков программирования, таких как LAD и FBD. Программа пользователя создается блок за блоком и в результате представляет собой исполняемую программу в коде MC7 независимо от языка программирования.

Для создания программы пользователя рекомендуется применять метод, ориентированный на создание исходных текстов программы, с использованием символьной адресации. Редактирование получается проще, меньше случается синтаксических ошибок и можно использовать какой-либо другой текстовый редактор. С помощью таблицы символов Вы можете определять различные абсолютные адреса всякий раз перед компиляцией программы, так, что Вы можете создавать многократно используемые "стандартные программы" независимо от конфигурации оборудования.

Способ программирования, ориентированный на создание исходных текстов программы, является единственно возможным способом, обеспечивающим блокам Вашей программы защиту (block protection KNOW_HOW_PROTECT).

Инкрементное программирование, тем не менее, является оптимальным для быстрой проверки изменений в программе непосредственно в CPU. Если изменение программы выдержало проверку, обновите и вновь скомпилируйте исходную программу. Таким образом, у Вас всегда будет текущая версия программы в формате ACSII-текстового файла. Инкрементное программирование также очень удобно для тестирования программы с помощью нескольких операторов, включаемых в интерактивном (online) режиме, которые в дальнейшем (после отладки) использоваться не будут.

Способ, ориентированный на создание исходных текстов программы

Способ программирования "Source-oriented" (ориентированный на создание исходных текстов программы) используется для редактирования исходных файлов STL-программы в каталоге *Source Files (Исходные файлы).* STL-файл имеет формат чисто ACSII-текстового файла. Он может содержать исходную программу для одного или нескольких блоков данных или блоков кода, также как определения данных пользовательского типа.

В SIMATIC Manager выберите каталог *Source Files (Исходные файлы)* и создайте новый исходный файл с помощью опций меню: *Insert -> S7 Software -> STL Source File (Вставка -> ПО S7 -> Исходный STL-файл).*

Можно создавать новые блоки намного более простым способом, если использовать опции меню: *Insert -> Block Template -> … (Вставка -> Шаблон блока -> …)* (в редакторе). Программа-редактор использует шаблоны из каталога …\Step7\S7ska, которые находятся в текстовых файлах S7kaf*nn*x.txt. Вы можете привести эти шаблоны к виду, отвечающему Вашим требованиям.

Вы также можете выбрать в качестве способа создания нового исходного STL-файла из одного или нескольких скомпилированных блоков с помощью опций меню: *File -> Generate Source File (Файл -> Создать исходный файл).*

Если Вы создали исходный файл с помощью стороннего текстового редактора, Вы можете использовать опции из меню SIMATIC Manager: *Insert -> External Source File (Вставка -> Внешний исходный файл)* для помещения файла в каталог *Source Files (Исходные файлы).* Вы можете скопировать выбранный исходный файл в каталог по Вашему выбору с помощью опций: *Edit -> Export Source File (Правка -> Экспорт исходного файла).*

При программировании, ориентированном на создание исходных файлов программы, Вы должны соблюдать определенные правила и использовать ключевые слова, зарезервированные для данного компилятора. В разделах 3.4.3 "Программирование кодовых блоков, ориентированное на создание исходных файлов на STL" и 3.6.2 "Программирование блоков данных, ориентированное на создание исходных файлов" представлена структура исходного STL-файла.

Компилирование исходного STL-файла

Вы можете сохранить исходную программу в любой момент во время редактирования, даже если программа еще не закончена. Программный редактор генерирует исполняемые блоки только по завершении создания программы и помещает их в каталог *Blocks (Блоки).* Если необходимо использовать глобальные символы в исходном STL-файле, то скомпилированная таблица символов должна быть доступна к моменту его компиляции.

На вкладке "Source Files" ("Исходные файлы") в диалоге, вызванном с помощью опций меню: *Options -> Customize (Опции -> Установки пользователя)* Вы можете задать свойства компилятора, такие как выбор условия компиляции исходных блоков, определяющего, должны ли переписываться существующие блоки, или блоки должны обновляться только, когда вся программа целиком будет свободна от ошибок. На вкладке "Generate Block" ("Создать блок") Вы можете задать автоматическое обновление ссылок при компиляции блока.

С помощью опций меню: *File -> Consistency Check (Файл -> Проверка соответствия)* можно осуществить синтаксическую проверку программы без

компилирования блоков.

Если исходная программа открыта, Вы можете начать компилирование посредством опций меню: *File -> Compile (Файл -> Компиляция).* Все не содержащие ошибок блоки исходной программы будут скомпилированы. Также как любой блок, содержащий ошибки не будет скомпилирован. Если при компиляции было выдано предупреждение, блок будет скомпилирован в любом случае, но, тем не менее, выполнение программы в CPU, возможно, будет с ошибками.

Вызываемые блоки должны уже присутствовать в скомпилированном виде при компиляции программы, или они должны располагаться в программе до точки их вызова (для более детального освещения вопроса о порядке расположения блоков обратитесь к разделу 3.4.3 "Программирование кодовых блоков, ориентированное на создание исходных файлов на STL").

Обновление или генерирование исходного STL-файла

На вкладке "Source Files" ("Исходные файлы") в диалоге, вызванном с помощью опций меню: *Options -> Customize (Опции -> Установки пользователя)* Вы можете выбрать опцию "Generate source files automatically" ("Создать исходный файл автоматически"), так что при сохранении запрограммированного инкрементным способом блока будет обновлен ранее существовавший исходный файл программы или будет сгенерирован новый файл (если таковой ранее не существовал). Вы можете задать имя нового исходного файла программы исходя из абсолютного или символьного адреса. Исходному файлу программы могут быть переданы адреса в абсолютной или символьной форме.

С помощью кнопки "Execute" ("Выполнение") Вы выбираете в последующем диалоговом окне блок, с которого Вы желаете начать генерацию исходного файла программы.

Инкрементное программирование

При инкрементном программировании Вы можете редактировать блоки и в автономном (offline), и в интерактивном (online) каталоге *Blocks (Блоки)*. При данном методе программирования редактор проверяет введенные в программу изменения сразу же после завершения текущей строки программы. При закрытии блока, он немедленно компилируется, так что сохранить можно только блоки, не содержащие ошибок.

На вкладке "Create Block" ("Создать блок"), выбранной с помощью опций меню: *Options -> Customize (Опции -> Установки пользователя)*, Вы можете выбрать опцию автоматического обновления ссылок (reference data) при создании блока.

Блоки могут редактироваться в автономном (offline) режиме в базе данных программатора и в интерактивном (online) режиме в CPU (или, как обычно говорится, в программируемом контроллере ["programmable controller"] или в PLC). Для этих целей в SIMATIC Manager используются два окна ("offline" и "online"); окна отличаются друг от друга строкой заголовка.

В "автономном" ("offline") окне Вы можете редактировать блоки непосредственно в базе данных PG. Из среды редактора Вы можете сохранить измененный блок в автономной ("offline") базе данных с помощью опций меню: *File -> Save (Файл -> Сохранить)* и передать его в CPU посредством опций меню: *PLC -> Download (PLC -> Загрузить).* Если нужно

сохранить открытый блок с другим номером или в другом проекте, если необходимо переслать его в библиотеку или в другой CPU, то Вы должны использовать команду: *File -> Save as (Файл -> Сохранить как).*

Для редактирования блока в CPU откройте блок в "интерактивном" ("online") окне. Это действие перешлет блок из CPU в программатор для редактирования. Вы можете вновь переслать отредактированный блок в CPU командой: *PLC -> Download (PLC -> Загрузить).* Если CPU находится при этом в рабочем (RUN) режиме, то обработка отредактированного блока начнется со следующего цикла сканирования программы. Если необходимо сохранить блок, отредактированный в "интерактивном" ("online") режиме также и в автономной ("offline") базе данных, то это можно сделать с помощью опций меню: *File -> Save (Файл -> Сохранить).*

В разделах 2.6.4 "Загрузка программы пользователя в CPU" и 2.6.5 "Обработка блока" содержится дополнительная информация по интерактивному (online) программированию. В разделах 3.4.2 "Инкрементное программирование кодовых блоков на STL" и 3.6.1 "Инкрементное программирование блоков данных на STL" показано, как вводится STL-блок.

2.5.4 Редактор SCL-программ (SCL Program Editor)

Опционное программное обеспечение S7-SCL дает возможность пользователю создавать программы на языке программирования SCL. При инсталляции ПО S7-SCL этот редактор встраивается в SIMATIC Manager. Вы можете использовать его точно также как и редакторы для стандартных языков программирования. Работая с SCL, Вы будете использовать метод программирования, ориентированный на создание исходных файлов программы (см. рис. 2.10).

Вы создаете сначала исходный текст программы, который затем необходимо скомпилировать. Вы можете также вызывать ранее скомпилированные блоки, находящиеся в каталоге *Blocks (Блоки),* встраивая их в Вашу программу. Эти блоки могут быть написаны с использованием иного языка программирования, например, на STL.

Если в программе Вы используете символьную адресацию для глобальных адресов, то готовая таблица символов уже должны быть доступна к моменту компиляции программы.

Однако Вы не сможете сгенерировать исходный SCL-файл из скомпилированного блока, например, если Вы удалили исходный файл по ошибке. (Примечание: скомпилированный блок будет выполняться в CPU, даже если исходный файл программы блока станет недоступным).

Запуск редактора STL- программ

Редактор SCL-программ запускается при открытии в SIMATIC Manager скомпилированного SCL-блока или исходного SCL-файла или с помощью меню панели задач Windows: *Start -> Simatic -> STEP 7 -> S7-SCL -> Program S7 Blocks.*

Если программный редактор при открытии скомпилированного SCL-блока не находит соответствующего ему исходного файла программы, например, если этот исходный файл был удален или перемещен, то блок будет открыт для редактирования с помощью редактора STL-программ.

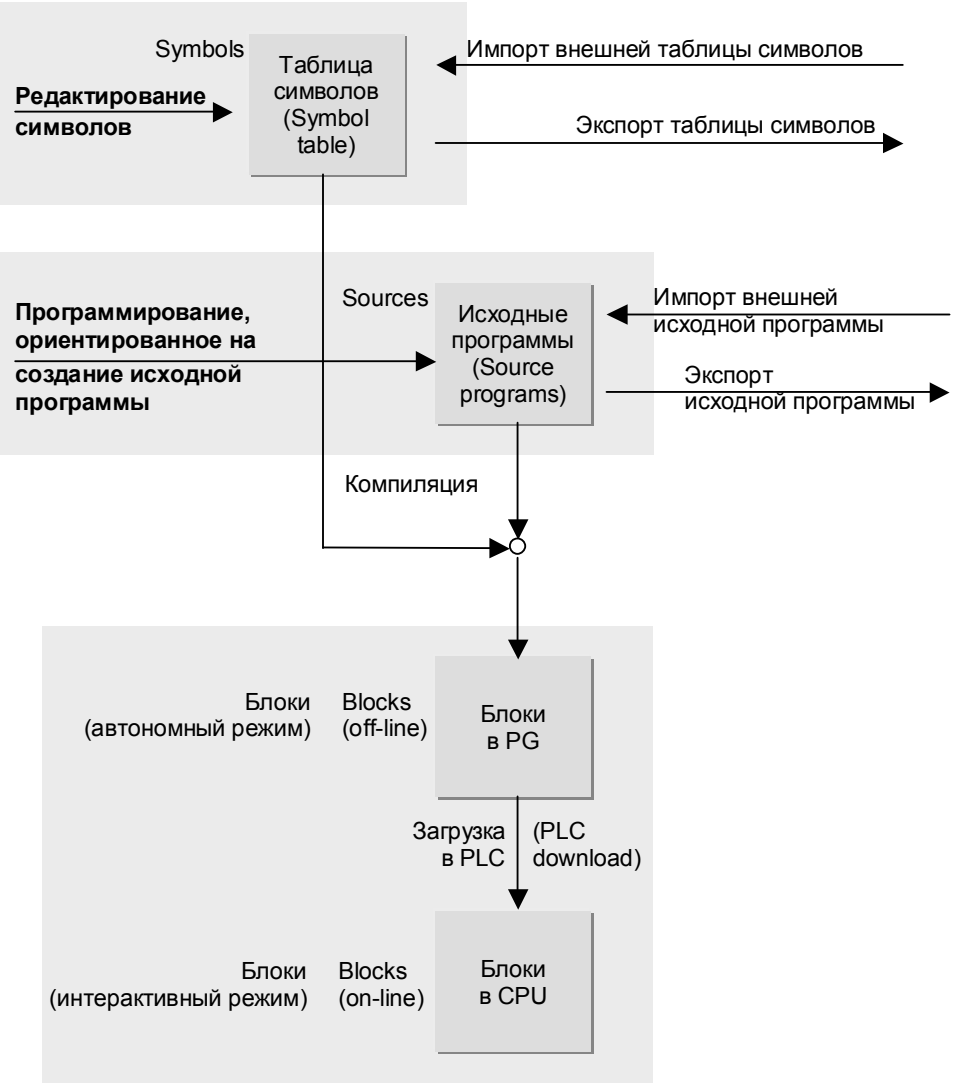


Рис. 2.10 Создание программы с использованием редактора SCL Program Editor.

Вы можете задать свойства для редактора SCL-программ с помощью опций меню: *Options -> Customize (Опции -> Установки пользователя).* На вкладке "Editor" ("Редактор") выберите свойства, с которыми новый блок должен быть сгенерирован и отображен, такие, например, как отображение программы с нумерацией строк.

Создание исходного SCL-файла

Выбрав каталог Source files (Исходные файлы) в SIMATIC Manager и затем опции меню: *Insert -> S7 Software -> SCL Source File (Вставка -> ПО S7 -> Исходный SCL-файл),* можно создать новый исходный файл. Двойным щелчком на исходном файле Вы можете его открыть. Простой способ создания нового блока возможен также при использовании опций меню: *Insert -> Block Template -> … (Вставка -> Шаблон блока -> …),* а также Вы можете

вставить готовые программные структуры в исходный программный файл в позиции курсора.

Если Вы создали исходный SCL-файл с помощью стороннего текстового редактора, Вы можете использовать опции из меню SIMATIC Manager: *Insert -> External Source File (Вставка -> Внешний исходный файл)* для помещения файла в каталог *Source Files (Исходные файлы).* Вы можете скопировать выбранный исходный файл в каталог по Вашему выбору с помощью опций: *Edit -> Export Source File (Правка -> Экспорт исходного файла).*

Если измененный исходный файл еще не сохранен, это индицируется символом звездочки после имени этого файла на панели заголовка окна редактора или в меню "Window" ("Окно").

При программировании, ориентированном на создание исходных файлов программы, Вы должны соблюдать определенные правила и использовать ключевые слова, зарезервированные для данного компилятора. В разделах 3.5.2 "Программирование кодовых блоков на SCL" и 3.6.2 "Программирование блоков данных, ориентированное на создание исходных файлов" представлена структура исходного SCL-файла.

Компилирование исходного SCL-файла

Вы можете сохранить исходную программу в любой момент во время редактирования, даже если программа еще не закончена. Программный редактор генерирует исполняемые блоки только по завершении создания программы и помещает их в каталог *Blocks (Блоки)*. Если необходимо использовать глобальные символы в исходном SCL-файле, то скомпилированная таблица символов должна быть доступна к моменту его компиляции.

Вы можете задать следующие установки среди других на вкладке "Compiler" ("Компилятор") в диалоговом окне, вызванном с помощью опций меню: *Options -> Customize (Опции -> Установки пользователя).*

- Create object code (Создать объектный код):
  Если выбрана данная опция, генерируются программные блоки, при условии, что не было выявлено ошибок при компиляции; с другой стороны Вы можете проверить программу на наличие синтаксических ошибок без создания блоков.
- Optimize object code (Оптимизировать объектный код):
  Если выбрана данная опция, созданные программные блоки оптимизируются с учетом требований к памяти и времени их выполнения.
- Monitor array limits (Отслеживать размерность массивов):
  Если выбрана данная опция, компилятор генерирует дополнительный программный код, который позволяет проверять, например, размерность массивов во время выполнения программы.
- Create debug info (Создать информацию для отладки):
  Если все еще необходимо отлаживать скомпилированную программу с помощью Program Status, Вы выбираете эту опцию. (Эта информация создается внутренне - без сохранения дополнительных программ)
- Set OK flag (Установить бит OK):
  Вы должны выбрать данную опцию, если в программе используется переменная OK или механизм EN/ENO.

Если исходный файл программы открыт, Вы можете начать компилирование посредством опций меню: *File -> Compile (Файл -> Компиляция).* Все не содержащие ошибок блоки исходной программы будут скомпилированы. Также как любой блок, содержащий ошибки не будет скомпилирован. Если при компиляции было выдано предупреждение, блок будет скомпилирован в любом случае, но, тем не менее, выполнение программы в CPU, возможно, будет с ошибками. Если необходимо скомпилировать только отдельные избранные блоки, выберите опции: *File -> Partial Compile (Файл -> Частичная компиляция).*

Вызываемые блоки должны уже присутствовать в скомпилированном виде при компиляции программы, или они должны располагаться в программе до точки их вызова (для более детального освещения вопроса о порядке расположения блоков обратитесь к разделу 3.5.2 "Программирование кодовых блоков на SCL"). SCL-компилятор автоматически создает отсутствующие экземпляры DB, если вызываются функциональные блоки. Для DB номер берется из таблицы символов (Symbol Table) или же выбирается наименьший свободный номер.

При компилировании стандартные блоки, такие, например, как IEC-функции, при первом вызове копируются в каталог *Blocks (Блоки)* из стандартной библиотеки.

С помощью опций: *PLC -> Download (PLC -> Загрузить)* можно загрузить в подключенный CPU все блоки, которые были созданы или были скопированы из стандартной библиотеки в каталог *Blocks (Блоки)* и скомпилированы при последней компиляции программы.

Компилирование файла управления

Язык программирования SCL облегчает процесс компиляции нескольких исходных файлов, которые должны выполняться вместе, но в определенном порядке. Вы должны создать файл управления компиляцией с помощью выбора опций меню: *Insert -> SCL Compilation Control File (Вставка -> Файл управления компиляцией)* при выбранном каталоге *Source Files (Исходные файлы).*

Откройте файл управления компиляцией и определите с помощью названий исходных файлов порядок, в котором эти файлы должны быть скомпилированы.

Посредством опций меню: *File -> Compile (Файл -> Компиляция)* Вы можете начать процедуру компиляции.

2.5.5 Перекомпоновка (Rewiring)

Функция перекомпоновки *Rewiring* позволяет Вам изменить адреса в отдельно скомпилированных блоках или в пользовательской программе в целом. Например, Вы можете изменить входные биты I 0.0 … I 0.7 на входные биты I 16.0 … I 16.7. Допустимыми адресами являются входы, выходы, меркеры, таймеры и счетчики, а также функции FC и функциональные блоки FB.

В SIMATIC Manager Вы должны выбрать объекты, в которых необходимо выполнить перекомпоновку; выберите отдельный блок, группу блоков, удерживая клавишу Ctrl, и щелкните кнопкой мыши на этих объектах или на пользовательской программе в целом - на каталоге *Blocks (Блоки).*

Для работы с таблицей, в которой необходимо указать старые адреса и адреса замены, выберите опции меню: *Options -> Rewire (Опции -> Перекомпоновка).* После того, как Вы подтвердите щелчком по кнопке OK, сделанные изменения, SIMATIC Manager выполнит замену адресов. После этого информационный файл покажет, в каких блоках были сделаны изменения и в каком количестве.

Существуют также дополнительные способы перекомпоновки:

- Для отдельно скомпилированных блоков Вы можете использовать также функцию *Address priority (Приоритет адреса).*
- В случае использования метода программирования, ориентированного на создание исходных текстов программы с использованием символьной адресации, Вы можете до компиляции внести изменения в таблицу символов, и после компиляции Вы получите перекомпонованную (*rewire*) программу.

2.5.6 Приоритет адресов (Address Priority)

В окне свойств автономного ("offline") объекта *Blocks (Блоки)* на вкладке "Blocks" ("Блоки") Вы можете назначить приоритет для одного из типов адресации (абсолютной или символьной) для уже сохраненных блоков, если эти блоки отображены. Перед этим все изменения в таблице символов или все назначения блоков глобальных данных должны быть сохранены.

По умолчанию принята установка: "Absolute address has priority" (абсолютная адресация имеет приоритет) (такое же поведение, как и в предыдущих версиях STEP 7). Такая установка означает, что если делаются изменения в таблице символов, то остается в силе абсолютная адресация, а символьные адреса соответственно изменяются.

Если задана установка: "Symbol has priority" (символьная адресация имеет приоритет), то это означает, что остается в силе символьная адресация, а абсолютные адреса изменяются.

Пример:

Пусть таблица символов содержит следующую информацию:

I 1.0 "Limit_switch_up" ("Верхний концевой переключатель")

I 1.1 "Limit_switch_down" ("Нижний концевой переключатель"),

и в программе скомпилированного блока сканируется вход I 1.0:

A **I 1.0 "Limit_switch_up"**

Если задана установка: "Symbol has priority" (символьная адресация имеет приоритет), а в таблице символов поменять значения выше указанных входов следующим образом:

I 1.0 "Limit_switch_down" ("Нижний концевой переключатель")

I 1.1 "Limit_switch_up" ("Верхний концевой переключатель"),

то программа будет содержать строку:

A I 1.1 **"Limit_switch_up"** ("Верхний концевой переключатель"),

а если задана установка: "Absolute address has priority" (абсолютная адресация имеет приоритет), то программа будет содержать строку:

A **I 1.0** "Limit_switch_down" ("Нижний концевой переключатель")

Если в результате изменений в таблице символов нет больше никаких назначений символьных имен абсолютным адресам, тогда при установке "Absolute address has priority" (абсолютная адресация имеет приоритет), выражения в программе будут содержать абсолютную адресацию (даже в режиме отображения символов), так как символьные имена как таковые отсутствуют. Если при тех же условиях задана установка: "Symbol has priority" (символьная адресация имеет приоритет), тогда те же выражения будут отброшены как ошибочные (так как обязательный абсолютный адрес игнорируется).

Если задана установка: "Symbol has priority" (символьная адресация имеет приоритет), то инкрементно программируемые блоки с символической адресацией сохранят свои символы в случае изменений в таблице символов. В этом случае такой блок с уже готовой программой может быть перекомпонован (rewired) с помощью изменения назначения адресов.

Примечание: такая перекомпоновка (rewiring) не может выполняться автоматически, так как скомпилированные блоки содержат исполняемый MC7-код с абсолютной адресацией. Изменения делаются только в соответствующих блоках при поступлении соответствующего сообщения, после которого они могут быть открыты и окончательно вновь сохранены.

2.5.7 Ссылки (Reference Data)

В добавление к собственно программе утилита SIMATIC Manager предоставляет Вам ссылки (Reference Data), которые Вы можете использовать как основу для корректировки или тестирования программы. Ссылочные данные содержат:

- Cross references (Перекрестные ссылки)
- Reserved locations (Зарезервированные области: I, Q, M, T, C)
- Program structure (Структура программы)
- Unused symbols (Неиспользуемые символы)
- Addresses without symbols (Адреса без символов)

Для генерации ссылочных данных выберите объект *Blocks (Блоки)* и затем опции меню: *Options -> Reference Data -> Display (Опции -> Ссылочные данные -> Отобразить)*. Представление ссылочных данных может изменяться в соответствии с рабочим окном с помощью опций: *View -> Filter… (Вид -> Фильтр…);* Вы можете сохранять установки для последующих сессий редактирования с помощью опций: *Save as Standard (Сохранить как стандарт).* Вы можете выводить на экран одновременно несколько списков.

С помощью опций меню: *Options -> Customize (Опции -> Установки пользователя)* в редакторе программы на вкладке "Create Blocks" ("Создать блоки") Вы можете включить или отменить обновление ссылочных данных при компиляции исходного файла программы или при сохранении инкрементно введенного блока.

Примечание: ссылки доступны только при автономной (offline) обработке данных; "автономные" (offline) ссылочные данные могут отображаться даже, если функция вызывается в блоке, открытом интерактивно (online).

Перекрестные ссылки (Cross references)

Список перекрестных ссылок показывает использование адресов и блоков в программе пользователя. В него входят абсолютные адреса, символьные (если есть), в блок в котором есть обращение к адресу, как используется адрес (для записи или для чтения) и зависящая от языка программирования информация. Для STL-программы столбец, зависящий от языка программирования, содержит информацию о сети, в которой адрес используется; для SCL-программы - содержит номер строки и столбца. Щелкните на заголовке столбца для сортировки таблицы по содержанию столбца.

Если адрес выбран, Вы с помощью выбора опций меню: *Edit -> Go To -> Line (Правка -> Перейти -> На строку)* можете запустить редактор программ, и при этом будет отображен фрагмент программы с выбранным адресом.

Список перекрестных ссылок можно просматривать с использованием фильтра (опции: *View -> Filter… (Вид -> Фильтр…)* для получения информации по интересующим адресам (например, проверить использование меркеров). Если дважды щелкнуть на адресе, будет в редакторе открыт блок на строке, в которой данный адрес используется. STEP 7 всякий раз при открытии списка перекрестных ссылок использует сохраненную (как "Standard") настройку фильтра.

Польза от применения: перекрестные ссылки показывают, были ли адреса использованы для считывания или сброса сигнала. Также они показывают, в каких блоках адреса использованы (возможно, больше одного раза).

Назначения (Assignments)

Список ссылок на I/Q/M показывает, какие биты в адресных областях I, Q и M имеют назначения в программе. В каждой строке показывается побитно один байт. Также указывается тип доступа (побайтный, пословный или двухсловный). Список ссылок на T/C показывает таймеры и счетчики, использованные в программе. В строке показываются по десять таймеров или счетчиков.

Польза от применения: список ссылок показывает, были ли определенные адресные области назначены (заняты) или какие адреса остаются все еще доступными.

Структура программы (Program structure)

Структура программы показывает иерархию вызовов блоков в пользовательской программе. Блоки вызова ("starting blocks") в иерархии вызовов выбираются с помощью настроек фильтров. Вы можете выбирать из двух различных видов:

"Древовидная" структура (tree structure) показывает все уровни вложения в системе вызовов блоков. Пользователь может изменять отображение уровней вложения, используя элементы управления в структуре - квадраты со значками "+" и "-". Требования, предъявляемые к временным локальным данным ("temporary local data"), показаны в целом для цепочки ("path") вызовов. Щелкните правой кнопкой мыши в поле разворачивающегося меню для открытия соответствующего блока, перейдите к месту вызова или к экрану с дополнительной информацией по блоку.

Структура "Родитель-потомок" (Parent-child structure) показывает 2 уровня с одним вызываемым блоком, а также информацию, зависящую от языка.

Польза от применения: структура программы показывает, какие блоки используются, все ли запрограммированные блоки вызываются, какие требования предъявляются к временным локальным данным, удовлетворяют ли локальные данные определенным требованиям для приоритетного класса (для организационного блока).

Неиспользуемые символы (Unused symbols)

Данный список показывает все адреса, имеющие назначения в таблице символов, но не используются в программе. Список показывает символ, адрес, тип данных и комментарий из таблицы символов.

Польза от применения: список позволяет определить, не были ли отдельные адреса по невнимательности забыты при записи программы, или, может быть, они избыточны и в действительности не нужны.

Адреса без символов (Addresses without symbols)

Список показывает все используемые в программе адреса, которые не имеют символов. Список показывает сами адреса, и как часто они используются.

Польза от применения: список позволяет определить, не используются ли отдельные адреса по невнимательности, (по случайности или по ошибке).

2.5.8 Многоязыковая поддержка комментариев и отображаемых текстов

Утилита SIMATIC Manager позволяет пользователю управлять несколькими версиями языка для комментариев и отображаемых текстов.

Выбор языка для комментариев и отображаемых текстов определяет язык текстов, вводимых пользователем. Язык сессии, заданный, например, для меню, сообщений об ошибках, устанавливается вместе с STEP 7 и выбирается с помощью утилиты SIMATIC Manager в опциях меню: *Options -> Customize (Опции -> Установки пользователя)* на вкладке "Language" ("Язык"). Здесь же Вы можете задать мнемоники, т.е. язык операторов и операндов, используемый в STEP 7. Все три установки не зависят друг от друга.

Общая процедура

Допустим, Вы ввели тексты на языке оригинала, (например, на английском языке), и Вам необходимо получить версию Вашей программы на немецком языке. Чтобы сделать это, экспортируйте требуемые тексты или типы текстов. Например, экспортируемый файл является *.csy файлом, который Вы можете редактировать с помощью Microsoft Excel. Вы можете ввести перевод (translation) для каждого текста и затем импортировать законченную таблицу трансляции (translation table) обратно в Ваш проект. Теперь Вы можете переключать языки текстов (сообщений) в проекте. У Вас есть возможность использовать для этого несколько языков.

Экспорт и импорт текстов (Exporting and importing texts)

С помощью утилиты SIMATIC Manager выберите объект, содержащий комментарии, которые необходимо перевести (транслировать), например, таблицу символов, каталог блоков (block container), несколько блоков или отдельный блок. Выберите опции меню: *Options -> Manage Multilingual Text -> Export (Опции -> Многоязыковая поддержка текста -> Экспорт).*

В появившемся окне диалога введите местоположение экспортируемого файла (storage location) и язык, на который требуется транслировать текст (target language). Выберите типы текстов (Text type), которые Вы хотите транслировать (Таблица 2.2).

Таблица 2.2

Типы транслируемых текстов (Text type) (извлечение)

Text type (Типы текста)	**Meaning (Значение)**
BlockTitle	Block title (Заголовок блока)
BlockComment	Block comment (Комментарий блока)
NetworkTitle	Network Title (Заголовок сегмента)
NetworkComment	Network Comment (Комментарий сегмента)
LineComment	Line Comment (Комментарий строки)
InterfaceComment	Comment in (Комментарий • declaration table of code blocks - таблицы объявлений блоков кода • data blocks - блоков данных • user data type UDT - пользовательских типов данных)
SymbolComment	Symbol Comment (Комментарий символа)

Для каждого типа текста генерируется отдельный файл, например, для комментариев из таблицы символов – файл SymbolComment.csv. Существующие экспортированные файлы могут быть расширены.

Откройте экспортированный(ые) файл(ы) в диалоговом окне Microsoft Excel с помощью опций меню: *File -> Open (Файл -> Открыть)* (Не открывать двойным щелчком на файле). Экспортированные тексты отображаются в первом столбце, и Вы можете ввести перевод данных текстов (транслировать) во втором столбце.

Вы можете вернуть переведенные тексты обратно в проект посредством опций меню: *Options -> Manage Multilingual Text -> Import (Опции -> Многоязыковая поддержка текста -> Импорт).* Файл протокола (log-файл) содержит информацию об импортированных текстах и о любых ошибках, которые могли произойти.

Примечание: Имя импортированного файла не может быть изменено, так как есть прямая связь между именем и типом текста (Text type), который содержится в файле.

Выбор и удаление языка

Вы можете переключать отображаемые тексты на любой инсталлированный в системе язык с помощью утилиты SIMATIC Manager в опциях меню: *Options -> Manage Multilingual Text -> Change Language (Опции -> Многоязыковая поддержка текста -> Сменить язык).* Процедура смены языка выполняется для объектов (блоков, таблицы символов), для которых соответствующие тексты были импортированы. Информация об этом содержится в файле протокола (в log-файле).

Вы можете также удалить любой инсталлированный в системе язык с помощью утилиты SIMATIC Manager в опциях меню: *Options -> Manage Multilingual Text -> Delete Language (Опции -> Многоязыковая поддержка текста -> Удалить язык).*

2.6 Интерактивный режим (Online Mode)

Пользователь создает конфигурацию оборудования и пользовательскую программу, в основном опираясь на технические средства “engineering system” (ES). S7-программа сохраняется автономно (offline) на жестком диске в текстовой форме, а также в скомпилированном виде.

Для переноса программы в CPU Вы должны соединить программатор PG с CPU, после чего - установить интерактивное ("online") соединение. Вы можете использовать это соединение для определения рабочего состояния CPU и назначенных (assigned) модулей, т.е. Вы можете реализовать функции диагностики.

2.6.1 Подключение к PLC (Connection a PLC)

Соединение MPI-интерфейса программатора PG и MPI-интерфейса CPU является аппаратным требованием для интерактивного (online) соединения. Такое соединение является единственным, если CPU является единственным подключенным программируемым модулем. Если существует несколько CPU в MPI-подсети, то каждому CPU должен быть назначен уникальный номер узла (MPI-адрес). Вы должны установить MPI-адрес при инициализации CPU. Перед связыванием всех CPU в единую подсеть подключите программатор только к одному CPU и перешлите туда объект *System Data (Системные данные)* из автономного каталога *Blocks (Блоки)* или непосредственно из утилиты конфигурирования Hardware Configuration с помощью опций меню: *PLC -> Download (PLC -> Загрузить).* При этом CPU получит свой собственный специальный MPI-адрес ("naming" - наименование) вместе с другими характеристиками.

MPI-адрес CPU в MPI-сети может быть изменен в любое время с помощью пересылки записи данных с новыми параметрами, содержащими новый MPI-адрес в CPU.

Будьте внимательны: изменение MPI-адреса имеет мгновенный эффект.

Программатор немедленно настраивается на новый адрес, поэтому необходимо настроить остальные параметры, такие, например, как коммуникации посредством глобальных данных на новый MPI-адрес.

MPI-параметры остаются в CPU даже после сброса памяти. Таким образом, CPU может быть адресован даже после сброса памяти.

Во всех случаях программатор может быть задействован в интерактивном режиме (online) с CPU, даже при использовании программы, независящей от модулей, и даже если в PG не установлено соответствующего проекта.

Если в PG не установлено соответствующего проекта, Вы должны установить соединение с CPU с помощью опций меню: *PLC -> Display Accessible Nodes (PLC -> Отобразить доступные узлы)*. При этом будет показано окно проекта (project) со структурой:

"*Accessible Nodes*" - "Module (MPI=n)" - "Online User Program (*Blocks*)" ["*Доступные узлы*" - "Модуль (MPI=n)" - "Интерактивная программа пользователя (*Блоки*)"].

При выборе объекта *Module* Вы можете использовать интерактивные (online) функции, такие как изменение рабочего состояния (operational status) и проверка состояния модулей (module status). Если выбрать объект *Blocks (Блоки),* то будут отображаться блоки, находящиеся в пользовательской памяти в CPU. При этом Вы можете редактировать (изменять, удалять, вставлять) отдельные блоки.

Вы можете считать системные данные из подключенного CPU с целью, скажем, последующей работы с существующей конфигурацией в отсутствие соответствующего проекта в системе управления данными PG (data management system).

Для этого создайте новый проект в SIMATIC Manager, выберите проект, затем - опции меню: *PLC -> Upload Station (PLC -> Считать конфигурацию станции).* После определения требуемого CPU в появившемся окне диалога интерактивные (online) системные данные записываются на жесткий диск.

Если программа не зависит от CPU, Вы должны включить окно проекта для интерактивного (online) режима. Если в MPI-подсети имеется несколько CPU и они доступны, то выбрав интерактивную (online) S7-программу и опции меню: *Edit -> Object Properties (Правка -> Свойства объекта),* задайте номера монтажных стоек и слотов установки CPU в таблице адресации модулей "Addresses Module".

Если Вы выбрали *S7-программу* в окне проекта для интерактивного (online) режима, то Вам будут доступны все интерактивные (online) функции для подключенного CPU. *Blocks (Блоки)* показывает блоки, размещенные в пользовательской памяти CPU. Если блоки в автономной (offline) программе соответствуют блокам в интерактивной (online) программе, то Вы можете редактировать блоки в пользовательской памяти, используя информацию системы управления данными программатора PG (символьные адреса, комментарии).

Если Вы переключаете **программу, предназначенную для CPU**, в интерактивный (online) режим, Вы можете делать изменения в программе только, если Вы открыли независимую от CPU программу. Вдобавок, теперь Вы можете конфигурировать SIMATIC-станцию, т.е. установить параметры CPU, а также адресовать и параметризировать модули.

2.6.2 Защита программы пользователя

Для соответствующим образом оснащенных CPU доступ к программе пользователя может быть защищен паролем. Каждый, кто знает пароль, имеет неограниченный доступ к пользовательской программе. Для тех, кто не знает пароля, Вы можете установить три уровня (степени) защиты. Установка уровней защиты производится на вкладке "Protection" ("Защита") в соответствующем окне утилиты для конфигурирования оборудования Hardware Configuration при параметризации CPU.

Уровень защиты 1: блокировка положения переключателя
(Protection level 1: keylock switch position)

Данный уровень защиты устанавливается по умолчанию (если не введен пароль). В этом случае программа пользователя защищена блокировкой функций переключателя режимов на передней панели CPU. В положениях RUN-P и STOP Вы имеете неограниченный доступ к программе пользователя; в позиции RUN возможен доступ в режиме "только чтение" с использованием программатора PG. Для этой позиции Вы можете также удалять блокировку, так чтобы режим не мог быть в дальнейшем изменен с помощью переключателя.

Вы можете обходить защиту посредством блокировки положения RUN, выбрав опцию, "Can be revoked with password" ("Может быть отменено при вводе пароля"), например, если CPU и его заблокированный переключатель режима находятся далеко или в трудно доступном месте.

Уровень защиты 2: защита от записи
(Protection level 2: write protection)

Данный уровень защиты обеспечивает доступ к программе пользователя только в режиме чтения вне зависимости от положения переключателя режимов.

Уровень защиты 3: защита в режиме чтения и записи
(Protection level 3: read/write protection)

Данный уровень защиты полностью блокирует доступ к программе пользователя вне зависимости от положения переключателя режимов.

Защита паролем (Password protection)

Если Вы выбираете уровни защиты 2 или 3 или уровень защиты 1 с опцией "Can be revoked with password" ("Может быть отменено при вводе пароля"), то Вам будет предложено определить пароль. Пароль может иметь размер до 8 символов.

При попытке получить доступ к программе пользователя, защищенной паролем, пользователю будет предложено ввести пароль. Если необходимо получить доступ к защищенному паролем CPU, пользователь также может ввести пароль посредством опций меню: *PLC -> Access Rights (PLC -> Права доступа)*. Необходимо сначала выбрать CPU или S7-программу.

В диалоговом окне "Enter Password" ("Введите пароль") Вы можете выбрать опцию "Use password for other protected modules" ("Использовать пароль для других защищенных модулей") для предоставления доступа ко всем защищенным модулям с одинаковым паролем.

При этом доступ с паролем авторизации ("Password access autorization") остается в силе до тех пор, пока последнее приложение системы S7 не будет деинсталлировано.

Каждый, кто знает пароль, имеет неограниченный доступ к пользовательской программе в CPU независимо от установленного уровня защиты и от блокировки переключателя режимов.

2.6.3 Информация о CPU (CPU Information)

При интерактивном (online) режиме Вам доступна представленная ниже информация, касающаяся CPU. Команды меню выводятся на экран, если Вы выбираете какой-либо модуль (в интерактивном [online] режиме и без проекта) или S7-программу (в интерактивном [online] окне проекта).

- *PLC -> Diagnose Hardware (PLC -> Диагностика оборудования)*

(см. раздел 2.7.1 "Диагностика оборудования")

- *PLC -> Module Information (PLC -> Информация о модуле)*

Общая информация (такая, например, как версия), диагностический буфер, память (текущее распределение [map] рабочей памяти [work memory] и загрузочной памяти [load memory], сжатие [compression]), время цикла [cycle time] (длина последнего цикла, длина самого длинного и длина самого короткого цикла программы), система времени (внутренние часы CPU, синхронизация времени, счетчик рабочего времени), рабочие характеристики (конфигурация памяти, размеры адресных областей, число доступных блоков, SFC и SFB), коммуникации (скорость передачи данных и коммуникационные соединения), стеки в режиме STOP (B-стек, I-стек и L-стек) в таблице адресации модулей "Addresses Module".

- *PLC -> Operating Mode (PLC -> Рабочий режим)*

Отображение рабочего режима (например, RUN или STOP), изменение рабочего режима.

- *PLC -> Clear/Reset (PLC -> Очистка/Сброс)*

Сброс CPU в STOP-режиме.

- *PLC -> Set Date and Time (PLC -> Установить дату и время)*

Установка внутренних часов CPU.

- *PLC -> CPU Messages (PLC -> Сообщения CPU)*

Сообщения об асинхронных системных ошибках и сообщения, определяемые пользователем, генерируются в программе с помощью SFC 52 WR_USMSG, SFC 18 ALARM_S и SFC 17 ALARM_SQ.

- *PLC -> Display Force Values, (PLC -> Отобразить форсированные значения),*

  PLC -> Monitor/Modify Variables, (PLC -> Мониторинг/модификация переменных)

(см. раздел 2.7.3 "Мониторинг/модификация переменных" и раздел 2.7.4 "Форсирование переменных").

2.6.4 Загрузка пользовательской программы в CPU

При пересылке в CPU Вашей пользовательской программы (а именно, скомпилированных блоков и данных конфигурации) она загружается в загрузочную (load) память CPU. Физически загрузочная (load) память может быть построена на элементах RAM или флэш EPROM, она может быть встроенной в CPU или быть в виде отдельного модуля.

Если модуль памяти типа флэш EPROM, то он может быть использован как перемещаемый носитель информации: данные на него могут быть записаны с помощью программатора PG, после чего этот модуль может быть вставлен в CPU, находящийся в выключенном состоянии. После включения питания и последующего сброса памяти соответствующие данные из модуля памяти пересылаются в рабочую (work) память CPU. Некоторые типы CPU позволяют перезаписывать вставленные в них модули флэш EPROM-памяти, но при этом перезаписывается только программа в целом.

Если загрузочная (load) память имеет тип RAM, то перенос целиком программы пользователя в CPU осуществляется следующим образом: CPU переводится в состояние STOP, затем выполняется сброс памяти и пересылается программа пользователя. Пересылаются также конфигурационные данные. Программа в RAM-памяти стирается при сбросе памяти или если отключается резервная батарея питания (backup battery).

Если Вы хотите изменить только данные конфигурации (свойства CPU, сконфигурированные соединения, GD-коммуникации, параметры модулей и т.п.), то Вам нужно загрузить в CPU только объект *System data (Системные данные)*. Для этого выберите объект и перешлите его с помощью опций меню: *PLC -> Download (PLC -> Загрузить)*. Параметры CPU вступают в силу немедленно; параметры для остальных модулей пересылаются в эти модули во время запуска (startup).

Нужно иметь в виду, что в PLC полная конфигурация загружается с объектом *System data (Системные данные)*. Если Вы используете опции меню: *PLC -> Download (PLC -> Загрузить)* в приложении, например, в GD-коммуникациях, то только отредактированные в приложении данные будут пересланы в PLC.

Примечание: для загрузки сжатого архивного файла используйте опции меню: *PLC -> Save Project on Memory Card (PLC -> Сохранить проект в модуле памяти)* (см. раздел 2.2.2 "Управление, перекомпоновка и архивация"). Проект, находящийся в заархивированном состоянии, не может быть отредактирован непосредственно ни с помощью программатора PG, ни в CPU.

2.6.5 Работа с блоками (Block Handling)

Перенос блоков

Если загрузочная (load) память имеет тип RAM, то Вы можете не только переносить программу целиком в интерактивном режиме (online), но также изменять, удалять или перезагружать отдельные блоки.

Вы можете пересылать отдельные блоки в CPU, выбрав их в "автономном" (offline) окне с последующим использованием опций меню: *PLC -> Download (PLC -> Загрузить).* Открыв одновременно "интерактивное" (online) и "автономное" (offline) окна, Вы можете с помощью манипулятора "мышь" перетаскивать блоки из одного окна в другое (метод "drag-n-drop").

Особое внимание необходимо при пересылке отдельных блоков во время рабочего режима. Если блоки, которые не доступны в памяти CPU, вызываются внутри блока, Вы должны сначала загрузить блоки "нижнего уровня". Это также касается блоков данных, адреса которых используются в загружаемом блоке. Вы должны загрузить блок "высшего уровня" последним. Затем, будучи вызванным, он будет незамедлительно выполнен в следующем программном цикле.

Утилита SIMATIC Manager позволяет Вам также пересылать отдельные блоки или программу в целом из "автономного" (offline) каталога "Blocks" ("Блоки") в CPU на языке SCL. Обратный перенос из CPU на жесткий диск скомпилированных блоков не имеет никакого смысла, так как они не могут быть напрямую отредактированы в редакторе SCL-программ. Вы можете редактировать только исходную SCL-программу и после этого вновь из нее создавать скомпилированные блоки.

Изменение блоков в интерактивном (online) режиме

Вы можете инкрементно редактировать на языке STL блоки программы пользователя в интерактивном (online) режиме (то есть в CPU), точно таким же образом, как и программу в автономном (offline) режиме. Однако, если "интерактивная" и "автономная" программы пользователя отличаются друг от друга, то это может привести к тому, что редактор не сможет отобразить дополнительную информацию из "автономной" базы данных; при этом эти данные (символьные идентификаторы, метки перехода, комментарии, пользовательские типы данных) могут быть потеряны.

Блоки, которые необходимо изменять в интерактивном режиме, лучше хранить автономно (offline) на жестком диске для того, чтобы избежать потери консистентности данных (например, конфликт "несовпадение временных меток" ["time stamp conflict"], когда интерфейс вызываемого блока отстает от программы в вызывающем блоке).

Удаление блоков

Если загрузочная (load) память построена исключительно на RAM - элементах, то блоки можно изменять или удалять.

Если пользовательская программа расположена в модуле памяти типа флэш EPROM, то блоки могут также быть изменены или удалены при условии, что доступный объем дополнительной RAM-памяти достаточен. Блоки в модуле памяти типа флэш EPROM выделяются как "invalid" ("неправильные"). Тем не менее, после сброса памяти или после включения питания без резервной батареи блоки вновь пересылаются из загрузочной (load) памяти на модуле типа флэш EPROM в рабочую (work) память.

Вы можете удалять модуль памяти типа флэш EPROM только в программаторе PG.

Сжатие (compressing)

Если Вы загружаете новый или измененный блок в CPU, CPU помещает блок в загрузочную (load) память и передает релевантные данные в рабочую (work) память. Если в рабочей (work) памяти уже находится блок с таким же номером, то этот "старый блок" объявляется неправильным (invalid) (пользователю предлагается это подтвердить), и после этого новый блок добавляется в "конец" занятой области памяти (после последней записи). Если даже блок удален, он помечается как неправильный (invalid), но в действительности из памяти не удаляется.

Это приводит к разрывам (gap) в непрерывном занятом пространстве памяти и уменьшает общий объем доступной памяти. Эти разрывы могут быть заполнены только при использовании функции сжатия *Compress*. При использовании этой функции в режиме RUN блоки, выполняющиеся в текущий момент, не могут быть перемещены; только в режиме STOP Вы можете эффективно ликвидировать разрывы в непрерывном занятом пространстве памяти.

Текущее состояние загрузки памяти может быть отображено в процентах с помощью опций меню: *PLC -> Module Information (PLC -> Информация о модуле)* на вкладке "Memory" ("Память"). В появившемся диалоговом окне Вы можете включить функцию сжатия.

С помощью SFC 25 COMPRESS Вы можете запрограммировать запуск функции сжатия, управляемый событием.

Блоки данных в интерактивном (online) и автономном (offline) режимах

Адресам данных в блоке данных могут быть назначены начальные значения (*initial value*) и фактические значения (*actual value*) (см. раздел 3.6 "Программирование блоков данных"). Если блок данных загружен в CPU, то начальные (initial) значения пересылаются в загрузочную (load) память, а фактические значения (actual) пересылаются в рабочую (work) память. Каждое изменение значения в соответствии с адресацией данных в программе предопределяет изменение соответствующего фактического значения.

Если Вы выгружаете блок данных из CPU, значения данных блока берутся из рабочей (work) памяти, в которой хранятся все фактические (actual) значения данных. Вы можете наблюдать фактические (actual) значения данных во время их считывания с помощью опций меню: *View -> Data View (Вид -> Вид данных).* Если Вы модифицируете фактические (actual) значения данных в блоке данных, а затем вновь записываете блок в CPU, то модифицированные значения поступают в рабочую (work) память.

Если в качестве загрузочной (load) используется модуль памяти типа флэш EPROM, то блоки из модуля памяти переносятся в рабочую (work) память после сброса памяти CPU. При этом блок памяти сохраняет первоначально запрограммированные в нем значения данных. То же самое происходит при включении питания при отключенной резервной батарее. Для S7-300 Вы можете предотвратить загрузку первоначальных значений данных, если Вы объявите эти области данных реманентными (retentive)

Блок данных, созданный с активированным свойством "UNLINKED" ("несвязанный"), не переносится в рабочую (work) память; он остается в загрузочной (load) памяти. Блок данных с активированным свойством "UNLINKED" ("несвязанный") может быть считан только с помощью функции SFC 20 BLKMOV.

2.7 Тестирование программы

После выполнения соединения с CPU и загрузки пользовательской программы Вы можете тестировать (отлаживать) программу в целом или по частям, отдельными блоками. Необходимо инициализировать переменные значениями, определенными, например, с помощью модулей симулятора, и оценить информацию отклика, полученного программой в виде значений данных. Если в результате ошибки CPU переходит в состояние STOP, Вы можете использовать, в частности, информацию о CPU.

Большие программы обычно отлаживаются по частям. Если Вам, например, необходимо отлаживать один блок, то загрузите этот блок в CPU и вызовите его в организационном блоке OB1. Если блок OB1 построен таким образом, что программа может быть отлажена фрагмент за фрагментом от начала до конца, то Вы можете выбирать для отладки отдельные блоки или фрагменты программы, используя функции перехода, чтобы миновать разделы,

не нуждающиеся в отладке.

С помощью опционного (поставляемого по отдельному заказу) программного обеспечения PLCSIM, Вы можете моделировать CPU в программаторе PG и таким образом отлаживать Вашу программу без дополнительного оборудования.

2.7.1 Диагностика оборудования

В случае отказа Вы можете считать диагностическую информацию из отказавших модулей с помощью функции диагностики оборудования "Diagnose Hardware". Для этого Вам сначала необходимо подключить программатор PG к MPI-шине и запустить утилиту SIMATIC Manager.

Если проект, связанный с конфигурацией установки, доступен в базе данных программатора PG, то откройте интерактивное (online) окно проекта с помощью опций меню: *View -> Online (Вид - Интерактивный режим)*. В противном случае выберите опции: *PLC -> Display Accessible Nodes (PLC - Отобразить доступные узлы)* и затем выберите CPU.

Теперь Вы можете получить краткий обзор сбойных модулей с помощью опций: *PLC -> Diagnose Hardware (PLC - Диагностика оборудования)* (по умолчанию). Утилита конфигурирования оборудования Hardware Configuration поддерживает функцию детальной диагностической информации о модулях при интерактивном режиме; этот режим устанавливается в утилите SIMATIC Manager на вкладке "View" ("Вид") при выборе опций меню: *Options -> Customize (Опции -> Установки пользователя)*.

Вы можете получить информацию о состоянии (status) и рабочем состоянии (operating state) модулей, доступных в интерактивном режиме, в форме отображения проекта (project view - отображение станций проекта, сообщающих об ошибках), в форме отображения станции (station view - отображение модулей станции, сообщающих об ошибках) и в форме отображения модуля (module view - отображение соответствующей диагностической информации).

2.7.2 Определение причины перехода в состояние STOP

Если CPU переходит в состояние STOP из-за ошибки, первое, что нужно сделать для определения причины перехода в это состояние, - это вывести для чтения содержимое диагностического буфера. CPU вводит в диагностический буфер все сообщения, в том числе, сообщение о причине перехода в состояние STOP и сообщения об ошибках, которые привели к этому.

Для вывода содержимое диагностического буфера переключите программатор PG в интерактивный (online) режим, выберите S7-программу и активируйте вкладку *Diagnostics Buffer* (*Диагностический буфер*) с помощью опций меню: *PLC -> Module Information (PLC -> Информация о модуле)*. Последнее событие из буфера (первое событие имеет номер 1) и есть причина перехода CPU в состояние STOP, например, "STOP because programming error OB not loaded" ("Состояние STOP из-за ошибки программы - блок OB не загружен").

Ошибка, которая привела к переходу CPU в состояние STOP, описана в предыдущем сообщении, например: "FC not loaded" ("FC не загружен"). Щелчком на номере сообщения Вы можете вывести на экран дополнительный комментарий в следующем нижнем поле экрана. Если сообщение касается ошибок программирования в блоке, Вы сможете открыть и отредактировать тот блок, нажав кнопку "Open Block" ("Открыть блок").

Если, например, причиной перехода CPU в состояние STOP является ошибка программирования, Вы можете установить "обстоятельства окружения" фрагмента программы, содержащего ошибку, с помощью вкладки "Stacks" ("Стеки"). Когда Вы откроете вкладку "Stacks" ("Стеки"), Вы увидите B-стек (block stack - стек блоков), который показывает расположение вызова всех незавершенных блоков вплоть до блока, в котором находится точка прерывания. Используя кнопку "I stack", Вы получите данные стека прерываний (interrupt stack), показывающего содержание регистров CPU (аккумуляторов, адресного регистра, регистра блока данных, слово состояния) в точке прерывания в тот момент, когда произошла ошибка. Используя кнопку "L stack" (local data stack - стек локальных данных), Вы получите доступ к локальным данным блока, который можно выбрать в окне B-стека. Перейти к окну B-стека можно с помощью щелчка манипулятора "мышь" на соответствующей кнопке.

2.7.3 Мониторинг и модификация переменных (Monitoring and Modifying Variables)

Есть замечательное средство для отладки пользовательской программы - функция для мониторинга и модификации переменных (Monitoring and Modifying of Variables), использующая VAT-таблицу (таблицу размещения переменных). Состояния сигналов или значения переменных простых типов данных могут быть отображены с помощью этого средства. При наличии доступа к пользовательской программе Вы можете также модифицировать переменные, т.е. изменять состояния сигналов или назначать новые значения.

Предупреждение: Вы должны избегать опасных состояний в Вашей установке, могущих возникать при изменении значений переменных!

Создание таблицы переменных

Для того, чтобы использовать функцию для мониторинга и модификации переменных (Monitoring and Modifying of Variables), Вы должны создать VAT-таблицу (таблицу размещения переменных), содержащую переменные и форматы соответствующих данных. Вы можете генерировать до 255 таблиц переменных (VAT1 … VAT255) и назначить им имена в таблице символов (Symbol Table). Максимальный размер VAT-таблицы составляет 1024 строки с содержанием до 255 символов (см. рис. 2.11).

Вы можете создать VAT-таблицу автономно (offline), выбрав пользовательскую программу *Blocks (Блоки)*, а затем опции меню: *PLC -> Monitor/Modify Variables (PLC -> Мониторинг/модификация переменных)*.

Вы можете определять переменные с помощью абсолютной или символьной адресации и выбрать для них тип данных (формат отображения переменной).

Для изменения выберите строки, затем: *View -> Display Format (Вид -> Отобразить формат)*, или просто щелкните правой кнопкой мыши на заголовке столбца "Display Format" ("Отобразить формат").

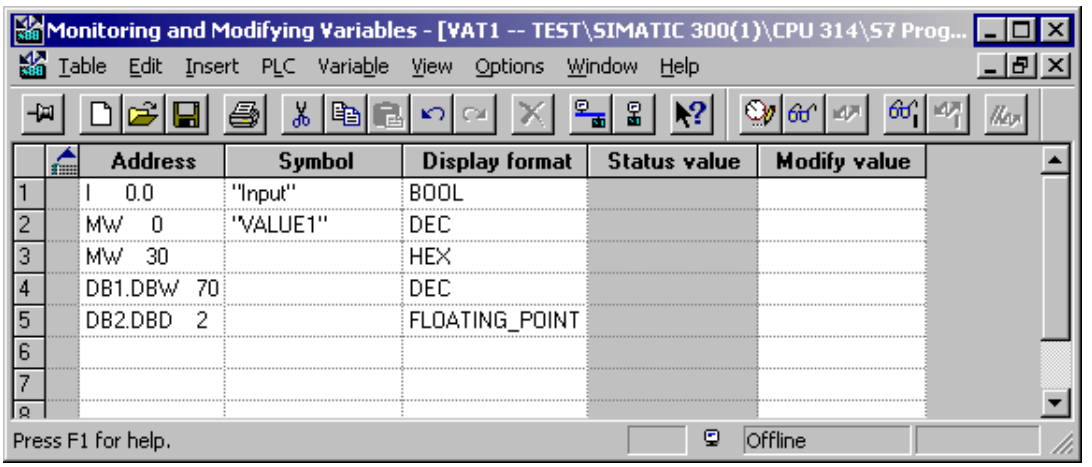


Рис. 2.11 Пример таблицы переменных (Variable Table)

Используйте строки комментариев для разделения таблицы на отдельные секции и придания отдельным частям таблицы заголовков. Вы можете также определять вид таблицы, а именно, какие столбцы должны быть отображены. В любое время Вы можете изменить переменные или формат их отображения, добавить или удалить строки таблицы. Таблица переменных должна быть сохранена в каталоге объекта *Blocks (Блоки)* с помощью опций: *Table -> Save (Таблица -> Сохранить)*.

Установление интерактивного (online) соединения

Для работы с VAT-таблицей, которая была создана автономно (offline), переключите ее в интерактивный режим с помощью опций меню: *PLC -> Connect To … (PLC -> Подключить к …)*. Вы должны переключать в интерактивный режим каждую таблицу отдельно, а после работы с таблицей - отключать это соединение с помощью опций: *PLC -> Disconnect (PLC -> Разъединить)* .

Условия запуска (Trigger conditions)

В таблице переменных выберите опции меню: *Variable -> Trigger (Переменная -> Запуск)* для установки точки запуска (trigger point) и условий запуска (trigger conditions) отдельно для функций мониторинга и модификации. Точка запуска (trigger point) - это такая точка, в которой CPU считывает значения из системной памяти или записывает значения в системную память.

Вы должны определить, будет ли считывание или запись происходить один раз или будет периодическим.

Если для функции мониторинга и модификации имеются одинаковые условия запуска, то функция мониторинга будет выполняться до функции модификации. Если Вы выбрали для функции модификации точку запуска "Start of cycle" ("Начало цикла"), то переменные будут модифицированы после обновления отображения входов процесса и перед вызовом блока OB 1. Если Вы выбрали для функции мониторинга точку запуска "End of cycle" ("Конец цикла"), то состояния переменных будут выведены после завершения OB1 и перед установкой выходов в соответствии с отображением выходов процесса.

Мониторинг переменных (Monitoring of Variables)

Для выбора функции мониторинга используйте опции меню: *Variable -> Monitor (Переменная -> Мониторинг)*. Переменные из VAT-таблицы обновляются в соответствии с определенными условиями запуска. Постоянный мониторинг позволит Вам следить за изменениями значений переменных на экране. Значения отображаются в формате данных, который Вы задаете в столбце Display Format (Формат отображения). Закончить непрерывный процесс мониторинга позволяет кнопка Esc.

Использование опций: *Variable -> Update Monitor Values (Переменная -> Обновить отслеживаемые значения)* позволяет однократно обновить наблюдаемые переменные, причем это обновление происходит мгновенно и не зависит от заданных условий запуска.

Модификации переменных (Modifying of Variables)

Для выбора функции модификации переменных (для пересылки в CPU измененных значений переменных) в соответствии с заданными условиями запуска используйте опции меню: *Variable -> Modify (Переменная -> Модификация)*. Вводите модифицированные значения только в те строки VAT-таблицы, в которых содержатся переменные, которые нужно изменить. Вы можете распространить комментарий на значение ("закомментировать" значение переменной) с помощью символов "//" или с помощью опций: *Variable -> Modify Value Valid (Переменная -> Модификация значения разрешена)*; такие значения не берутся в расчет при модификации переменных. Значения отображаются в формате данных, который Вы задаете в столбце Display Format (Формат отображения). Закончить непрерывный процесс модификации можно с помощью кнопки Esc.

Использование опций: *Variable -> Activate Modify Values (Переменная -> Активировать модификацию значений)* позволяет однократно модифицировать переменные, причем это происходит мгновенно и не зависит от заданных условий запуска.

2.7.4 Форсирование переменных (Forcing Variables)

Отдельные типы CPU позволяют использовать особую функцию - форсирование переменных (Forcing Variables), заключающуюся в том, что Вы с ее помощью можете задавать фиксированные значения некоторым переменным.

При этом пользовательская программа не сможет изменить эти значения. Форсирование разрешено для любого режима CPU и выполняется немедленно после запуска функции.

Предупреждение: Вы должны избегать опасных состояний в Вашей установке, могущих возникнуть при форсировании значений переменных!

Отправной точкой для форсирования переменных является VAT-таблица. Вы должны создать VAT-таблицу, после этого - задать адреса, для которых требуется форсирование значений. Затем необходимо установить соединение с CPU. Вы можете открыть окно, содержащее форсируемые значения, выбрав опции меню: *Variable -> Display Force Values (Переменная -> Отобразить форсированные значения).*

Если форсированные значения уже активны в CPU, это отображается в окне функции форсирования (force window) с помощью выделенного шрифта. Вы можете теперь перенести некоторые или все адреса из таблицы переменных в окно функции форсирования или внести в этом окне новые адреса. После определения переменных для форсирования значений Вы должны сохранить содержание окна функции форсирования с помощью опций меню: *Table -> Save As (Таблица -> Сохранить как).*

Функция форсирования значений может быть использована для следующих адресных областей:

- Входы I (отображение процесса)

  [S7-300 и S7-400]
- Выходы Q (отображение процесса)

  [S7-300 и S7-400]
- Периферийные входы PI

  [только S7-400]
- Периферийные выходы PQ

  [S7-300 и S7-400]
- Меркеры M

  [только S7-400]

Вы можете запустить функцию форсирования с помощью опций меню: *Variable -> Force (Переменная -> Активировать форсирование значений).* CPU использует форсированные значения для заданных переменных и не разрешает в дальнейшем изменять значения этих переменных.

Пока активна функция форсирования:

- Все попытки чтения по адресу форсированной переменной из пользовательской программы (например, load [загрузить]) и из системной программы (например, обновление образа процесса) всегда оканчиваются с одним результатом: величина переменной соответствует форсированному значению.
- В S7-400 все попытки записи по адресу форсированной переменной из пользовательской программы (например, transfer [переслать]) и из системной программы (например, посредством SFC) всегда оканчиваются без результата: изменения переменной запрещены. В S7-300 из пользовательской программы можно изменить ранее форсированное значение переменной.

Функция форсирования переменных в S7-300 соответствует функции модификации в циклическом режиме: после обновления отображения входов процесса CPU перезаписывает входы форсированными значениями; перед установкой выходов процесса в соответствии с отображением выходов процесса CPU перезаписывает последние форсированными значениями.

Примечание: функция форсирования не завершается с закрытием окна функции форсирования, таблицы переменных или при разрыве связи с CPU!

Остановить работу функции форсирования переменных можно, если только Вы используете опции меню: Variable -> Delete Force (Переменные -> Отменить функцию форсирования).

Функция форсирования также может быть остановлена, если выполнить сброс памяти или выключить (перевключить) питание, при условии, что CPU не имеет резервной батареи питания.

Если функция форсирования остановлена, соответствующие адреса продолжают содержать форсированные значения до тех пор, пока они не будут изменены или из пользовательской, или из системной программы.

Функция форсирования имеет стабильный эффект только для изменения I/O в CPU. Если после перезапуска форсированные PI и PQ больше не назначаются (например, в результате новой параметризации), то эти PI и PQ не поддерживают форсированные значения.

Обработка ошибок

Если при считывании оказывается, что "ширина доступа" (access width) больше, чем размер форсируемых данных (например, форсируется байт [byte] в слове [word]), то не форсируемая часть значения адреса считывается как обычно. Если при этом происходит ошибка синхронизации (ошибка доступа или ошибка длины данных [access or area length error]), то программой пользователя или CPU фиксируется "ошибка вставки значения" ["error substitute value"] или же CPU переходит в состояние STOP.

Если при записи оказывается, что "ширина доступа" (access width) больше, чем размер форсируемых данных (например, форсируется байт [byte] в слове [word]), то не форсируемая часть значения адреса записывается как обычно. При подобной ошибке доступа при записи форсированный компонент адреса остается неизменным, то есть защита от записи (write protection) не отменяется ошибкой синхронизации (synchronization error).

Считывание (loading) форсированных периферийных выходов дает в результате форсированные значения. Если "ширина доступа" (access width) соответствует размеру форсируемых данных, входные модули, которые вставляются в стойку взамен отказавших или для расширения, могут получить форсированные значения.

Вход I в образе процесса, связанный с форсированным периферийным входом PI, не форсируется; заранее он не определен и может быть переопределен. При обновлении образа процесса данный вход получает форсированное значение периферийного входа.

При форсировании периферийных выходов PQ связанный выход Q в образе процесса не обновляется и не форсируется (форсирование действует только "внешне" ["externally"] на выходы модуля). Значения выходов сохраняются и могут быть перезаписаны; считывание с выходов показывает записанные значения (не форсированные значения). Если выходной модуль форсирован,

и если потом этот модуль отказал или удален, то он будет вновь принимать форсированные значения, когда он будет вновь включен в стойку в работоспособном состоянии.

Выходные модули выводят состояние сигнала "0" или предустановленное значение (substitute value) по OD сигналу (блокировка выходных модулей в режимах STOP [стоп], HOLD [пауза] и RESTART [перезапуск]) - даже если периферийные выходы форсированы (исключение составляют аналоговые модули без распознавания сигнала OD, которые продолжают выдавать на выход форсированное значение сигнала). Если сигнал OD выключен, функция форсирования вновь продолжает действовать.

Если в режиме STOP активирована функция *Enable PQ (Разблокировать PQ)*, то форсированные значения также имеют эффект в режиме STOP (благодаря деактивации OD-сигнала). Кода действие функции *Enable PQ (Разблокировать PQ)* прекращается, модули вновь переходят в безопасное ("safe") состояние (состояние сигнала "0" или предустановленное значение [substitute value]); при этом форсированное значение выхода вновь становится действительным при переходе в режим RUN.

2.7.5 Разблокировка периферийных выходов (функция Enable peripheral outputs)

В режиме STOP выходные модули обычно заблокированы OD-сигналом. С помощью функции "Enable peripheral outputs" ("Разблокировка периферийных выходов") Вы можете деактивировать OD-сигнал, таким образом, Вы сможете модифицировать сигналы выходных модулей даже в случае, если CPU находится в режиме STOP. Модификация сигналов производится с помощью таблицы переменных. Модифицируются только назначенные CPU периферийные выходы. Возможное применение для этого: тестирование монтажа выходов в STOP-режиме без пользовательской программы.

Предупреждение: Вы должны предотвратить возможность возникновения опасных ситуаций при разблокировке периферийных выходов.

Создайте таблицу переменных и введите в нее периферийные выходы (PQ), теперь модифицируйте значения переменных. Переключите таблицу переменных в интерактивный (online) режим с помощью опций меню: *PLC -> Connect To (…,* и остановите CPU, если необходимо, например, с помощью опций меню: *PLC -> Operating Mode*, затем выберите "STOP".

Вы можете деактивировать OD-сигнал с помощью опций меню: *Variable -> Enable Peripheral Outputs (Переменные -> Разблокировка периферийных выходов);* выходы модуля имеют состояние сигнала "0", или предустановленное значение (substitute value), или форсированное значение (force value). Вы можете модифицировать периферийные выходы с помощью опций меню: *Variable -> Activate Modify Values (Переменные -> Активировать изменение выходов).* Вы можете изменить значения для модификации значений переменных и вновь после этого активизировать функцию модификации.

Вы можете деактивировать функцию модификации периферийных выходов с помощью опций меню: *Variable -> Enable Peripheral Outputs (Переменные -> Разблокировка периферийных выходов)* или с помощью кнопки ESC.

При этом вновь активируется OD-сигнал, и выходы модуля принимают состояние сигнала "0", или предустановленное значение (substitute value), или форсированное значение (force value) сбрасывается.

Если режим STOP деактивируется в то время, пока активна функция "Enable peripheral outputs" ("Разблокировка периферийных выходов"), все периферийные входы сбрасываются, OD-сигнал активируется при переходе к режиму перезапуска (RESTART), а затем вновь активируется при переходе к режиму RUN.

2.7.6 Функция "Program Status" ("Состояние программы") для STL

С помощью функции "Program Status" ("Состояние программы") программный редактор обеспечивает дополнительные возможности для тестирования пользовательской программы. С помощью этой функции редактор отображает последовательно для каждой строки программы состояние выбранного Вами регистра. Настройка отображаемых данных выполняется на вкладке "STL" после выбора опций меню: *Options -> Customize (Опции -> Установки пользователя)* ("Standard" ["Стандарт"] предполагает отображение аккумулятора 1 или значений таймера или счетчика).

Для отладки блок программы помещается в пользовательскую память (user memory) CPU, вызывается и обрабатывается в редакторе. Откройте этот блок, например, с помощью двойного щелчка кнопки манипулятора "мышь" на блоке в интерактивном (online) окне SIMATIC Manager. Редактор запускается и отображает программу, содержащуюся в блоке.

Выберите подсеть, которую необходимо отладить. С помощью опций меню: *Debug -> Monitor (Отладка -> Мониторинг)* активируйте функцию Program Status (Состояние программы). Теперь Вы можете наблюдать состояния адресов, результат логической операции и назначения регистров. С помощью опций меню: *Debug -> Monitor (Отладка -> Мониторинг)* Вы можете также деактивировать функцию Program Status (Состояние программы).

С помощью опций меню: *Debug -> Call Environment (Отладка -> "Обстоятельства вызова")* Вы можете задать условия запуска функции. Такие установки Вам потребуются, если блок, который Вы отлаживаете, вызывается более чем один раз в Вашей программе. Вы можете инициировать запись состояний (status recording), или определив порядок вызовов, или сделав его зависимым от открытого блока данных. Если блок вызывается только один раз, то выберите "No Condition" ("Нет условий").

Вы можете модифицировать переменные, используя функцию Program Status (Состояние программы). Выберите адрес данных, которые должны быть модифицированы, затем выберите опции: *Debug -> Modify Address (Отладка -> Модифицировать адрес)*.

Запись информации функции Program Status (Состояние программы) требует дополнительного времени в цикле выполнения программы. Поэтому Вы можете выбирать один из двух рабочих режимов для отладки программы: "debug mode" (режим отладки) и "process mode" (режим обработки процесса). Первый из указанных режимов (режим "debug mode" [режим отладки]) позволяет использовать все функции отладки без ограничения. Этот режим выбирается, например, для отладки блоков без подключения к системе, так как отдельные функции отладки сильно увеличивают время выполнения цикла сканирования программы.

В режиме "process mode" (режим обработки процесса) необходимо стремиться минимизировать удлинение цикла сканирования из-за функций отладки, поэтому здесь накладываются определенные ограничения, например, на программные циклы (не все проходы циклов программы отображаются). Отдельные CPU позволяют установку рабочего режима на вкладке "Protection" ("Защитный режим") при параметризации CPU. Если при параметризации CPU был установлен "debug mode" (режим отладки), то Вы сможете изменить режим работы только во время повторной процедуры параметризации. Другими словами, режим может быть изменен в диалоговом окне. Установка рабочего режима отображается с помощью опций: *Debug -> Operation (Отладка -> Работа).*

Функции отладки Breakpoints (точки прерывания), Single-step Mode (пошаговый режим)

Для блоков, написанных на языке STL, некоторые CPU позволяют выполнять отладку программы оператор за оператором в пошаговом режиме "Single-step mode". CPU находится в режиме HOLD; в целях безопасности периферийные выходы заблокированы. Используя точки прерывания (breakpoints), Вы можете остановить программу в любой момент и отлаживать в пошаговом режиме (step-by-step).

Должен быть установлен режим "debug mode" (режим отладки). Если при параметризации CPU был установлен "debug mode" (режим отладки), то Вы можете изменить режим работы только во время повторной процедуры параметризации. Другими словами, режим может быть изменен в диалоговом окне.

Для установки точки прерывания (breakpoint) установите курсор в строке с нужным оператором и выберите опции: *Debug -> Set Breakpoint (Отладка -> Установка точки прерывания)*. Для отладки выберите опции меню: *Debug -> Breakpoints Active (Отладка -> Активировать точки прерывания)*; эта команда вызовет перенос точек прерывания в CPU и активирует их. CPU необходимо перезапустить, в процессе обработки программы CPU при достижении точки прерывания переходит в состояние паузы HOLD. При этом в специальном окне будет отображено текущее содержание регистра.

Теперь Вы можете запустить программу для выполнения в построчном (в пошаговом) режиме, выбрав опции меню: *Debug -> Execute Next Statement (Отладка -> Выполнить следующий оператор)*. При этом выполнение программы будет прерываться на каждом операторе, и при этом будет отображаться текущее содержание регистра. Если в текущем операторе производится вызов блока, Вы можете продолжить пошаговую обработку программы в этом блоке, если выберете опции: *Debug -> Execute Call (Отладка -> Выполнять вызов)*.

С помощью опций меню: *Debug -> Resume (Отладка -> Продолжить)* программа выполняется с нормальной скоростью, пока не встретит следующую точку прерывания.

Блоки, содержащие точки прерывания, не могут быть изменены или перезагружены в интерактивном режиме (online). Все точки прерывания сначала должны быть удалены. Также все точки прерывания должны быть удалены при выходе из режима отладки с точками прерывания. С помощью опций меню: *Debug -> Resume (Отладка -> Продолжить)* CPU снова переключается в режим RUN.

2.7.7 Отладка SCL-программ

Если необходимо отладить SCL-программу, Вы должны скомпилировать ее с опцией "Create debug info" ("Создать отладочную информацию"). Вы можете установить эту опцию на вкладке "Compiler" ("Компилятор") после выбора опций меню: *Options -> Customize (Опции -> Установки пользователя)* в редакторе SCL Editor. Далее выполняется компиляция при выборе команды "Create object code" ("Создать объектный код"), затем программа переносится в CPU по команде: *PLC -> Download (PLC -> Загрузить)*.

Отладчик для SCL-программ является встроенным компонентом редактора SCL Editor.

Функция "Program Status" ("Состояние программы") для SCL

С помощью этой функции отладки Вы можете отлаживать группы операторов, "ведя мониторинг области" ("monitor area") во время работы. Размер переменных "области мониторинга" зависит от используемых в программе операторов. Значения переменных в этой области обновляются и отображаются циклически.

Если область мониторинга находится в той части программы, которая выполняется в каждом программном цикле, то значения переменных из связанных в каскад циклов обычно не могут быть достоверными. Значения переменных, которые изменялись во время текущего прохода цикла, отображаются выделенным (черным) цветом шрифта, а значения, которые не изменялись, представляются светло-серым цветом.

Для того чтобы отладить SCL-программу, переключите CPU в RUN или RUN-P режим и откройте исходный файл программы. Выберите рабочий режим: *Debug -> Operation -> Debug Operation (Отладка -> Работа -> Отладка).*

Установите курсор на начало области, которую нужно отладить. Начните процесс отладки с помощью выбора опций меню: *Debug -> Monitor (Отладка -> Мониторинг)*. Имена и значения переменных в области мониторинга отображаются построчно в правой части окна редактора.

Вы можете прервать процесс отладки, выбрав вновь опции меню: *Debug -> Monitor (Отладка -> Мониторинг)*; при выборе опций меню: *Debug -> End Debug (Отладка -> Конец отладки)* процесс отладки завершается.

Функции отладки Breakpoints (точки прерывания),
Single-step Mode (пошаговый режим)

При отладке в пошаговом режиме "Single-step mode" Вы можете контролировать выполнение программы строка за строкой и при этом вести мониторинг значений переменных. Используя точки прерывания (breakpoints), Вы можете остановить программу в любой момент и отлаживать в пошаговом режиме (single-step) с этой точки программы.

Для отладки программы в пошаговом режиме (single-step) должны быть выполнены следующие требования: отлаживаемый блок не должен быть защищен, он должен быть открыт интерактивно (online) и не должен при этом модифицироваться в редакторе.

Пошаговый режим (single-step) отладки работает только на тех CPU, которые обеспечивают поддержку этого режима. Должен быть установлен режим "debug mode" (режим отладки), а отладка с использованием функции Program Status (состояние программы) должна быть выключена. CPU переходит в режим HOLD в точке прерывания (breakpoint), и отладка в пошаговом режиме (step-by-step) возможна только в режиме HOLD.

Для отладки откройте исходный файл программы и определите точки прерывания (breakpoints) установкой курсора в строке с нужным оператором и выберите опции: *Debug -> Set Breakpoint (Отладка -> Установка точки прерывания)*. Необходимо предотвратить возможность возникновения опасных состояний в установке при активации режима отладки. Для активации режима отладки выберите опции меню: *Debug -> Breakpoints Active (Отладка -> Активировать точки прерывания)*. CPU необходимо перезапустить, при достижении точки прерывания в процессе обработки программы CPU переходит в состояние паузы HOLD (см. рис. 2.12).

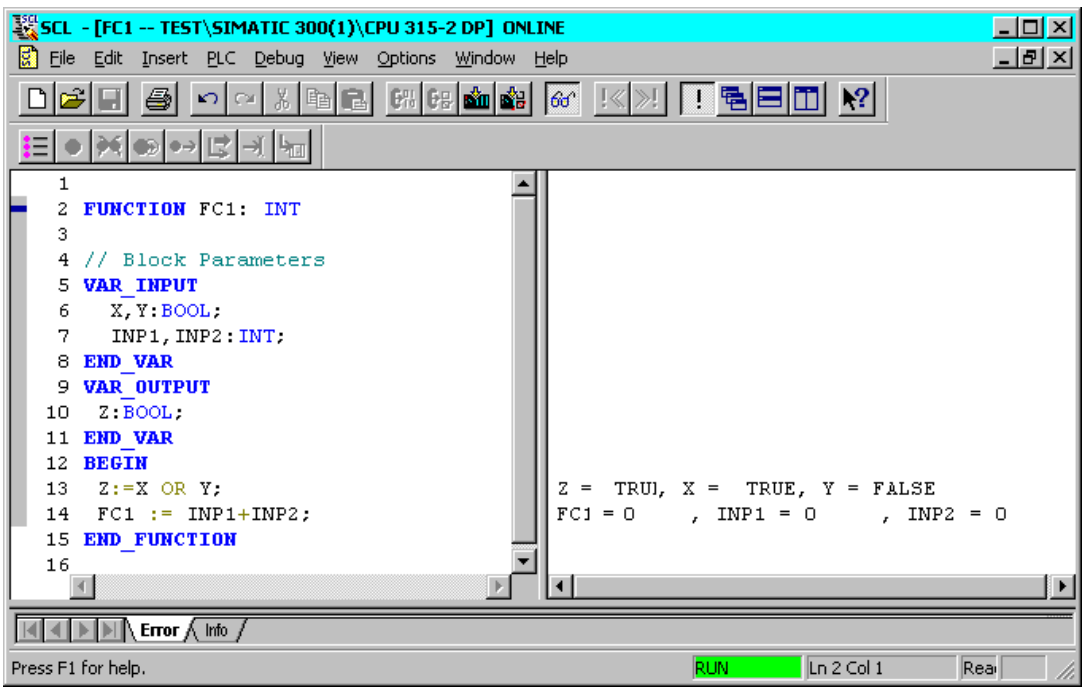


Рис. 2.12 Отладка SCL-программы

Теперь Вы можете запустить программу для выполнения в построчном (в пошаговом) режиме, выбрав опции меню: *Debug -> Execute Next Statement (Отладка -> Выполнить следующий оператор)*. Выполнение программы будет прерываться на каждом операторе, и при этом значения переменных из текущего оператора отображаются в правой части окна редактора. Отображение символьных имен может быть включено или выключено с помощью опций: *View -> Symbolic Representation (Вид -> Символьное представление)*.

С помощью опций меню: *Debug -> Resume (Отладка -> Продолжить)* программа выполняется с нормальной скоростью, пока не встретит следующую точку прерывания. При выборе опций меню: *Debug -> Execute to Selection (Отладка -> Выполнять до выбранного)* программа выполняется до раздела, выбранного с помощью курсора.

Вы можете управлять точками прерывания в программе с помощью опций меню: *Debug -> Edit Breakpoints (Отладка -> Редактировать точки прерывания)*. Вы можете также прервать процесс отладки программы с помощью повторного выбора опций меню: *Debug -> Breakpoints Active (Отладка -> Активировать точки прерывания)*.

Опции: *Debug -> End Debug (Отладка -> Завершение отладки)* вызывают завершение процесса отладки.

Примечание: с помощью опций меню: *Debug -> Execute Next Statement (Отладка -> Выполнить следующий оператор)* и *Debug -> Execute to Selection (Отладка -> Выполнять до выбранного)* устанавливаются и активируются точки прерывания. Необходимо обеспечить, чтобы при задании точек прерывания не было превышения максимально возможного количества точек прерывания, которое зависит от типа применяемого CPU.

3 SIMATIC S7-программа

В этой главе представлена структура пользовательской программы для CPU систем SIMATIC S7-300/400, начиная от различных приоритетных классов (определяющих характер выполнения программы) и отдельных частей программы (блоков) и заканчивая переменными и типами данных. В данной главе основное внимание уделяется программированию блоков на языках STL и SCL. Типы данных подробно описаны в главе 24 "Типы данных".

Вы должны определить структуру пользовательской программы на этапе разработки, когда Вы согласовываете технологические и функциональные характеристики; это будет решающим фактором при создании программы, ее отладке и тестировании. Для получения оптимальной программы необходимо обратить особое внимание на ее структуру.

3.1 Обработка программы

В целом программное обеспечение для CPU состоит из операционной системы (operating system) и пользовательской программы (user program).

Операционная система - это совокупность всех инструкций и деклараций, которые управляют системными ресурсами и процессами, использующими эти ресурсы. Операционная система включает в себя такие функции как резервирование данных в случае сбоя электропитания, активация приоритетных классов и т.п. Операционная система - это такой компонент CPU, к которому Вы, как пользователь, не имеете доступа в режиме записи. Тем не менее, Вы можете перезагружать операционную систему с модуля памяти, например, в случае обновления программы.

Пользовательская программа (user program) - это совокупность всех инструкций и деклараций (в данном случае - программных элементов), для обработки сигналов, с помощью которых установка (процесс) регулируется в соответствии с определенной задачей управления.

3.1.1 Методы обработки программы

Пользовательская программа может состоять из программных разделов, которые обрабатываются в CPU в зависимости от конкретного события. Одним из таких событий может быть, например, запуск системы автоматического управления, прерывание или обнаружение программной ошибки (см. рис. 3.1). Программы, назначенные для обработки событий, разделяются по приоритетным классам (*priority class*), с помощью которых определяется порядок обработки отдельных разделов ("mutual interruptibility" - система взаимных прерываний) программы в случаях, когда происходит одновременно несколько событий.

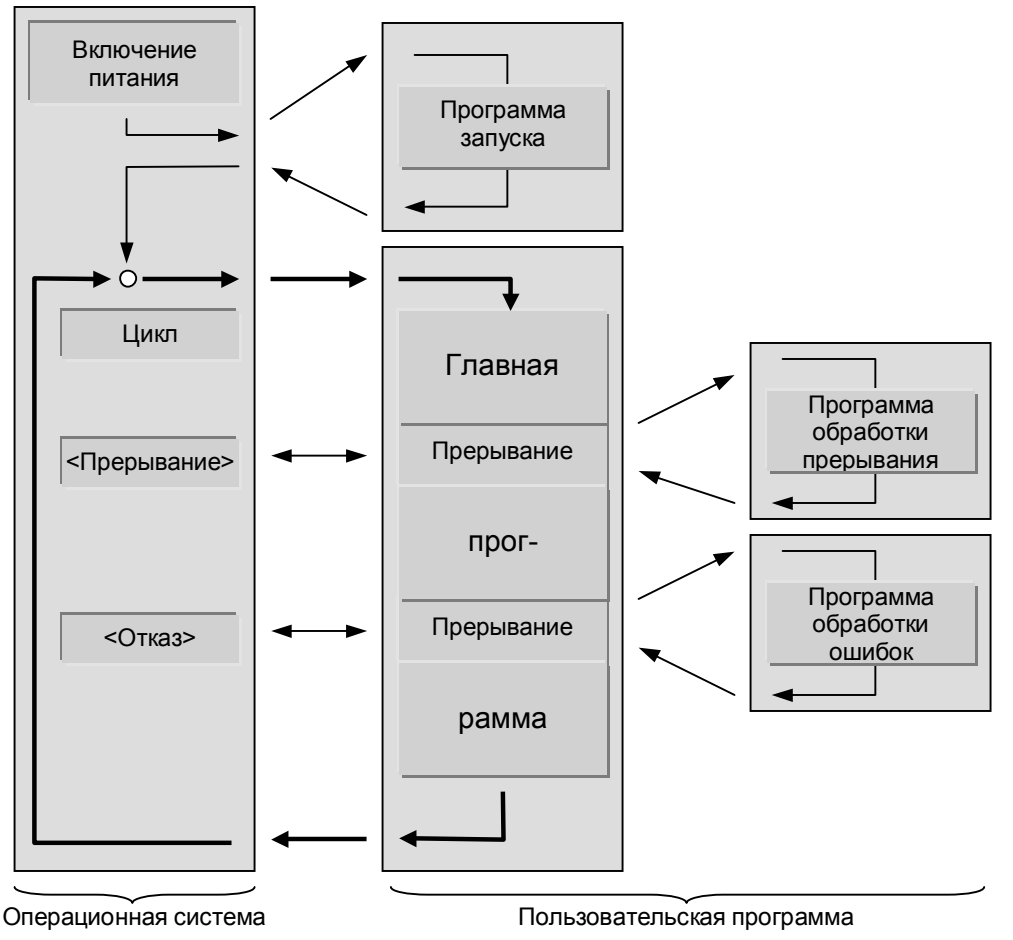


Рис. 3.1 Методы обработки пользовательской программы

Программа с наинизшим приоритетом является главной (main) программой, которая циклически обрабатывается CPU. События могут прерывать главную (main) программу в любой момент, после чего CPU выполнит связанную с прерыванием обслуживающую программу или программу обработки ошибок и затем вновь вернет управление в главную программу.

Специальный организационный блок (OB) соответствует каждому событию. Организационные блоки имеют приоритетные классы в пользовательской программе. Если происходит событие, CPU активизирует соответствующий ему организационный блок. Организационный блок (OB) - это часть пользовательской программы, которую Вы сами можете написать.

Перед тем, как CPU начнет обработку главной программы, он выполняет программу запуска (startup routine). Эта программа может быть запущена такими событиями как включение питания, поворот переключателя режимов на передней панели CPU или с помощью программатора PG. Обработка программы, следующая за выполнением программы запуска, в системе S7-300 всегда начинается с начала главной программы ("complete restart" - "полный перезапуск"), а в системе S7-400 может также использоваться режим, когда продолжается сканирование программы с точки, в которой оно было прервано ("warm restart" - "теплый перезапуск").

Главная (main) программа располагается в организационном блоке OB 1, который используется практически во всех пользовательских программах. Начало пользовательской программы соответствует первому сегменту в блоке OB 1. После завершения выполнения OB 1, что соответствует окончанию выполнения программы, CPU возвращает управление в операционную систему и после вызова различных функций операционной системы, таких как обновление отображений процесса, вновь вызывает для выполнения OB 1.

События, которые могут вмешиваться в работу программы, - это прерывания и ошибки. Источником прерываний могут стать либо процесс (аппаратное прерывание), либо CPU (прерывания по времени ["watchdog", то есть таймерные], прерывания по времени суток ["time-of-day"] и т.д.)

Что касается ошибок, то они подразделяются на синхронные и асинхронные. Асинхронные ошибки - это такие ошибки, которые не зависят от выполняемой программы, например, это может быть сбой питания на устройстве расширения или такое событие, как удаление модуля из стойки.

Синхронные ошибки - это такие ошибки, которые возникают при выполнении программы, например, ошибка будет зафиксирована при попытке доступа к несуществующему адресу или ошибка может произойти при преобразовании типов данных. Типы и номера регистрируемых событий и соответствующих организационных блоков определяются типом CPU; не каждый CPU способен обрабатывать все события, возможные в STEP 7.

3.1.2 Классы приоритетов

В таблице 3.1 показаны доступные в SIMATIC S7 организационные блоки с возможными для них классами приоритетов (приоритетными классами).

Таблица 3.1 Организационные блоки SIMATIC S7

Организационный блок	Условия вызова	Приоритет	
		по умолчанию	возможные изменения
OB свободного цикла OB 1	Периодически вызывается операционной системой	1	Нет
TOD прерывания OB 10 … OB 17	В определенное время суток или через равные промежутки времени (например, ежемесячно)	2	2 … 24
С задержкой времени OB 20 … OB 23	По истечении запрограммированного времени; управление из пользовательской программы	3 … 6	2 … 24
Watchdog прерывания OB 30 … OB 38	Регулярный вызов через запрограммированные интервалы времени (например, каждые 100 мс)	7 … 15	2 … 24
Прерывания процесса OB 40 … OB 47	Вызов по сигналу прерывания от I/O модулей	16 … 23	2 … 24
Мультипроцессорное прерывание OB 60	Вызов по приходу события в мультипроцессорном режиме; управление из пользовательской программы	25	Нет
Ошибки резервирования OB 70 OB 72 OB 73	 В случае потери резервирования из-за ошибок I/O В случае ошибки резервирования CPU В случае ошибки резервирования коммуникаций	 25 28 25	 2 … 26 2 … 28 24 … 26
Асинхронные ошибки OB 80 OB 81 … OB 84, 86, 87 OB 85	В случае ошибок, не связанных с выполнением программы (например, ошибка времени [time error], диагностическое прерывание, прерывание вставки / удаления модуля, отказ стойки / станции)	 26[2)] 26[2)] 26[2)]	 26 2 … 26 24 … 26
Фоновая обработка OB 90	Продолжительность минимального цикла еще не достигнута	29[1)]	Нет

Таблица 3.1 Организационные блоки SIMATIC S7 (*продолжение*)

Организационный блок	Условия вызова	Приоритет	
		по умолчанию	возможные изменения
Программа запуска OB 100, 101, 102	При запуске PLC	27	Нет
Синхронные ошибки OB 121, OB 122	В случае ошибок, связанных с выполнением программы (например, ошибка I/O доступа)	Приоритет OB, вызвавшего ошибку.	

[1] см. текст

[2] при запуске: 28

В некоторых классах приоритетов Вы можете изменять заданный приоритет при параметризации CPU. В таблице 3.1 для организационных блоков показаны возможные нижний и верхний приоритетные классы. Каждый CPU имеет свой диапазон значений для нижнего и верхнего приоритетных классов. В данном обзоре представлены отдельные типы CPU.

Организационный блок OB 90 (фоновая обработка) может выполняться вместо OB 1 и, как в случае с OB 1, его обработка может быть прервана любыми прерываниями и ошибками.

Программа запуска может быть в организационном блоке OB 100 (полный перезапуск) или в организационном блоке OB 101 (теплый перезапуск); она имеет приоритетный класс 27. Асинхронные ошибки, происходящие в программе запуска, имеют приоритетный класс 28. Диагностические прерывания рассматриваются как асинхронные ошибки.

Вы должны определить, какие доступные приоритетные классы Вы будете использовать при параметризации CPU. Неиспользуемые приоритетные классы (организационные блоки) должны получить приоритет 0. Соответствующие организационные блоки должны быть запрограммированы для всех используемых приоритетных классов; иначе CPU вызовет OB 85 ("Program Processing Error" - "Ошибка выполнения программы") или перейдет в режим STOP.

Для каждого выбранного приоритетного класса во области временных локальных данных (L-стек) должно быть достаточно места (более подробная информация находится в разделе 18.1.5 "Временные локальные данные").

3.1.3 Спецификации для обработки программы

Операционная система CPU обычно использует параметры, принятые по умолчанию. Вы можете изменить эти установки при параметризации CPU с помощью утилиты конфигурирования оборудования Hardware Configuration для того, чтобы параметры системы удовлетворяли Вашим особым требованиям. Вы можете изменить эти параметры в любое время.

Каждый CPU имеет свой собственный особый набор параметров. В приведенном ниже списке представлен обзор всех параметров STEP 7 и их наиболее важные установки.

- Sturtup (параметры запуска)

  Характеристика типа запуска ("cold restart" ["холодный перезапуск"] / ("warm restart" ["теплый перезапуск"]); мониторинг сигналов "Ready" или параметризации модуля; максимальная продолжительность времени, которое может произойти до момента "теплого перезапуска".

- Cycle/clock memory (цикл/тактовые меркеры)

  Включение/выключение циклического обновления отображения процесса; задание продолжительности периода мониторинга и минимальной продолжительности времени цикла; продолжительность времени цикла в процентах для коммуникаций; число тактовых меркеров; размер области отображения процесса.

- Retentive memory (реманентная память)

  Число реманентных меркеров, таймеров и счетчиков; определение реманентных областей для блоков данных.

- Memory (память)

  Максимальное количество временных локальных данных в приоритетных классах (в организационных блоках); максимальный размер L-стека и число коммуникационных заданий.

- Interrupts (прерывания)

  Спецификация приоритета аппаратных прерываний, прерываний с задержкой (time-delay interrupts), асинхронных ошибок и (возможно в скором времени) коммуникационных прерываний.

- Time-of-day Interrupts (прерывания по времени суток)

  Спецификация приоритета, спецификация стартового времени и периодичности.

- Cyclic Interrupts (циклические прерывания)

  Спецификация приоритета, спецификация времени цикла и фазового смещения.

- Diagnostics/Clock (диагностические прерывания/системные часы)

  Индикация причины перехода в состояние STOP; тип и интервал синхронизации времени; коэффициент коррекции.

- Protection (параметры доступа к программам)

  Спецификация уровня защиты, задание пароля.

- Multicomputing (параметры мультипроцессорного режима)

  Определение числа CPU.

- Integrated I/O (параметры встроенных I/O)

  Активация и параметризация встроенных I/O.

При запуске CPU загружает заданные пользователем параметры вместо параметров, принятых по умолчанию. Параметры, определенные пользователем остаются в силе до тех пор, пока они не будут заменены.

3.2 Блоки

Чтобы сделать Вашу программу более легкой для чтения и понимания, Вы можете разбить ее на такое количество частей, какое Вы пожелаете. Языки программирования STEP 7 поддерживают такой подход к программированию, обеспечивая Вас необходимыми функциями. Каждая такая часть программы должна быть самодостаточной (self-contained) и должна решать технологическую или функциональную задачу.

Рассматриваемые части программы называются "блоками" ("block"). Блок - это часть программы пользователя, характеризующаяся своими собственными функциями, структурой или функциональным назначением.

3.2.1 Типы блоков (Block Types)

Язык программирования: STL использует различные типы блоков для разных задач:

- User blocks (пользовательские блоки)

  Пользовательские блоки - это блоки, содержащие пользовательскую программу и пользовательские данные.

- System blocks (системные блоки)

  Системные блоки - это блоки, содержащие системную программу и системные данные.

- Standard blocks (стандартные блоки)

  Стандартные блоки - это готовые к использованию блоки, такие, например, как драйверы для функциональных блоков FM или коммуникационных процессоров CP.

Пользовательские блоки (User blocks)

Для больших и сложных программ "структурирование" - подразделение программы на блоки рекомендуется и отчасти является необходимостью. Вы можете выбирать среди различных типов блоков те или иные, в зависимости от условий применения.

Организационные блоки OB (Organization blocks)

Этот тип блоков служит своеобразным интерфейсом между операционной системой и пользовательской программой. Операционная система CPU вызывает организационные блоки при возникновении особого события, например, аппаратного прерывания или прерывания времени суток. Главная программа находится в организационном блоке OB 1. Остальные организационные блоки имеют постоянные назначенные номера, основанные на событиях, для обработки которых они вызываются.

Функциональные блоки FB (Function blocks)

Эти блоки являются частями программы, вызов которых может быть запрограммирован с помощью параметров блока. Они обладают областью памяти для переменных (variable memory), которая расположена в блоке данных. Этот блок данных постоянно назначен функциональному блоку, или, точнее, *вызову* функционального блока. Возможно даже назначение нескольких блоков данных (с одинаковой структурой данных, но содержащих разные значения) каждому вызову функционального блока. Такой постоянно назначенный блок данных называется *экземплярным блоком данных* (instance data block), а совокупность вызова функционального блока и экземплярного блока данных называется *экземпляром вызова* (call instance) или, для краткости, "экземпляром" ("instance"). Функциональные блоки могут также хранить свои переменные в экземплярном блоке данных вызывающего функционального блока; тогда такой экземплярный блок данных называется "локальным экземпляром" ("local instance").

Функции FC (Functions)

Функции используются для программирования часто повторяющихся или сложных функций автоматики (automation functions). Функциям могут назначаться параметры. Функции могут возвращать значение (значение вызванной функции) в вызывающий блок. Причем значение функции - необязательный параметр. Кроме функционального значения функция может иметь другие выходные параметры. Функции не сохраняют информацию и не имеют назначенных блоков данных.

Блоки данных DB (Data blocks)

Эти блоки содержат данные Вашей программы. Программируя блоки данных, Вы определяете, в какой форме данные будут сохраняться (в котором блоке, в каком порядке и с каким типом данных). Существует два способа использования блоков данных: как блоки глобальных данных (global data blocks) и как экземплярные блоки данных (instance data blocks). Блоки глобальных данных в пользовательской программе являются, как говорится, "свободными" ("free") блоками данных и не назначаются кодовому блоку. Экземплярные блоки данных, однако, назначаются функциональному блоку и сохраняют часть локальных данных этого функционального блока.

Максимальное число для каждого типа блоков и размер этих блоков определяются типом CPU. Число организационных блоков и их номера фиксированы; они назначаются операционной системой CPU. Блокам других типов Вы можете самостоятельно назначать номера внутри определенных пределов. Также Вы можете выбрать для каждого блока имя (символ) в таблице символов, с тем, чтобы ссылаться на блок по символьному имени.

Системные блоки (System blocks)

Системные блоки (System blocks) являются компонентами операционной системы. Они могут содержать программы (системные функции SFC или системные функциональные блоки SFB) или данные (системные блоки данных SDB). Системные блоки предоставляют множество важных системных функций, доступных пользователю, таких, например, как функции управления внутренними часами CPU или различные коммуникационные функции.

Вы можете вызывать SFC и SFB, но Вы не можете ни изменить эти функции, ни запрограммировать их самостоятельно. Собственно системные блоки не занимают места в пользовательской памяти (user memory); только вызовы блоков и экземплярные блоки данных для SFB располагаются в пользовательской памяти.

Системные блоки данных SDB содержат информацию о таких вещах, как конфигурация автоматизированной системы или параметры модулей. Система STEP 7 самостоятельно генерирует эти блоки и управляет ими.

Тем не менее, Вы можете определять их содержимое, например, когда Вы конфигурируете станции. Как правило, системные блоки данных SDB размещаются в загрузочной (load) памяти. Пользователь не имеет доступа к ним из своей программы.

Стандартные блоки (Standard blocks)

В дополнение к функциям и функциональным блокам, которые Вы можете создавать самостоятельно, Вы можете использовать готовые для применения блоки, так называемые "стандартные блоки" ("Standard blocks").

Эти блоки могут быть получены или на различных носителях, или могут быть в составе библиотек из комплекта поставки ПО STEP 7 (например, IEC-функции или функции для S5/S7 конвертирования).

В главе 33 "Библиотеки блоков" представлен обзор стандартных блоков из состава "*библиотеки стандартных блоков*" *Standard Library*.

3.2.2 Структура блоков (Block Structure)

На нижеприведенных рисунках представлены структуры блоков для случаев "инкрементного" программирования и программирования, ориентированного на создание исходных текстов программы:

Логический блок (logic block) ("инкрементное" программирование)

Block header (заголовок блока)

Declaration (раздел объявления)

Address (адрес)	*Declaration (объявление)*	*Name (имя)*	*Type (тип)*	

Program (программа)

```
A  Input1      //Limit switch responded      (отклик конечного выключателя)
A  Input2      //Manual operation            (ручной режим)
=  Output1     //Message to operator panel (сообщение на панель оператора)
```

Блок данных (data block) ("инкрементное" программирование)

Block header (заголовок блока)

Declaration (раздел объявления)

Address (адрес)	*Name (имя)*	*Type (тип)*	*Initial Value (начальное значение)*	

Рис. 3.2 Структура блока ("инкрементное" программирование)

Логический блок (logic block) (программирование, ориентированное на создание исходных текстов программы)

```
Block type Address (Тип блока Адрес)
Block header (заголовок блока)

VAR_xxx

name : Data type := Initialization;
name : Data type := Initialization;
…

END_VAR

BEGIN

Program          (программа)

END_Block Type
```

Блок данных (data block) (программирование, ориентированное на создание исходных текстов программы)

```
DATA_BLOCK Address (Блок данных Адрес)
Block header (заголовок блока)

STRUCT

name : Data type := Initialization;
name : Data type := Initialization;
…

END_ STRUCT

BEGIN

name := Initialization;

END_DATA_BLOCK
```

Рис. 3.2 Структура блока (программирование, ориентированное на создание исходных текстов программы)

По существу кодовые блоки состоят из трех частей:

- Block header (заголовок блока), который содержит свойства (характеристики) блока, такие как имя блока.
- Declaration section (раздел объявления), в котором декларируются (т.е. определяются) локальные ("block-local" - "внутриблочные") переменные данного блока.
- Program section (раздел программы), который содержит саму программу и комментарии к ней.

Блоки данных имеют похожую структуру:

- Block header (заголовок блока), который содержит описание свойств (характеристик) блока.
- Declaration section (раздел объявления), в котором объявляются локальные ("внутриблочные") переменные; в этом случае с адресами данных указываются типы данных.
- Initialization section (раздел инициализации), в котором отдельным адресам данных назначаются начальные значения.

В случае "инкрементного" программирования раздел объявления переменных и раздел инициализации объединены. Вы определяете адреса данных, их типы данных в "declaration view" (вид "объявлений") и также Вы можете инициализировать каждый адрес данных отдельно в "data view" (вид "данных") (см. ниже).

3.2.3 Свойства блоков (Block Properties)

Свойства блоков или атрибуты содержатся в заголовке блока. Вы можете увидеть и изменить атрибуты блока в редакторе с помощью опций меню: *File -> Properties (Файл -> Свойства)* (см. рис. 3.3).

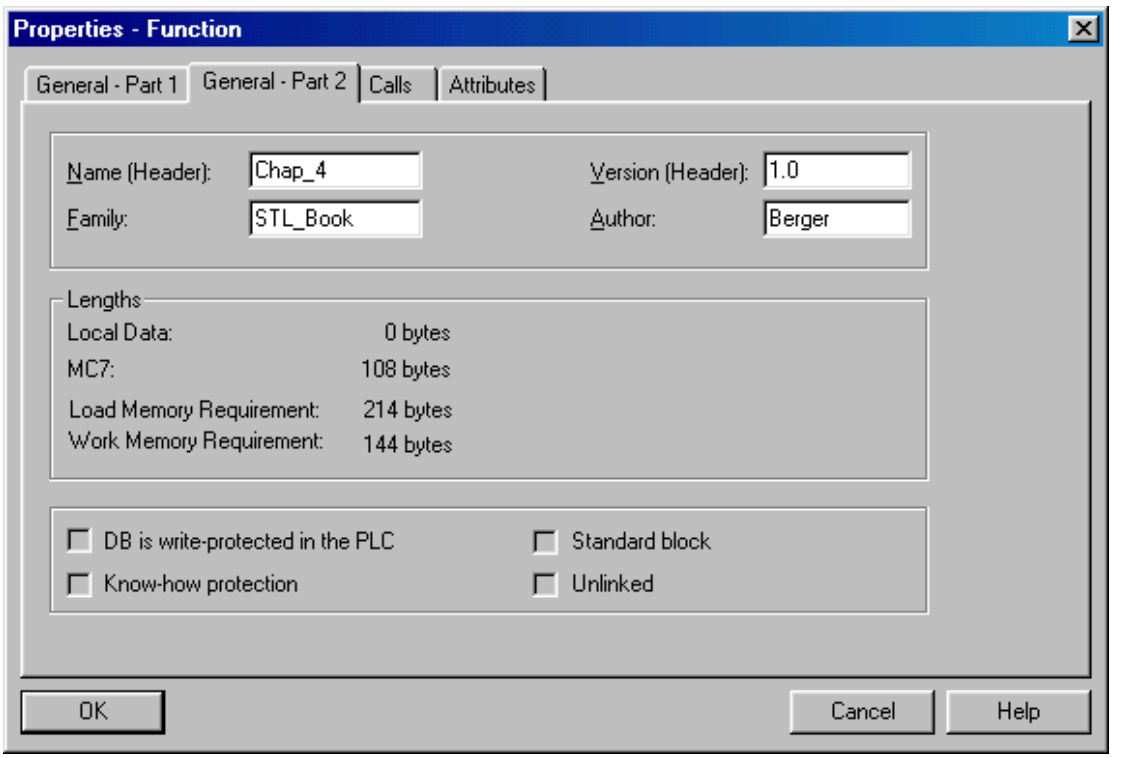


Рис. 3.3 Окно свойств блока ("Properties - *Type* block")

На вкладке "General - Part 2" ("Общие - часть 2") показано распределение памяти для блока в байтах:

- Local Data (временные локальные данные - размещение стека локальных данных).
- MC 7: размер блока (только код).
- Load memory requirement (требования к загрузочной памяти).
- Work memory requirement (требования к рабочей памяти).

Атрибут "*Know-how protection" ("защита технологии")* используется для защиты блока. Если блок защищен с помощью установки этого атрибута, то программа из этого блока не может быть отображена, распечатана или изменена. В редакторе можно будет увидеть только заголовок блока и таблицу объявления переменных (declaration table) с параметрами блока. При вводе текста в исходный файл Вы можете защитить каждый блок сами с помощью ключевого слова KNOW_HOW_PROTECT. Если Вы защитили блок, то никто (даже Вы) не сможет увидеть компилированный вариант этого блока (Вы должны обеспечить при этом безопасное место хранения для исходного файла программы!).

Заголовок блока любого стандартного блока (*standard block*), который предоставляется фирмой Siemens, содержит атрибут "S*tandard Block*".

Атрибут "*DB is write-protected in the PLC" ("DB в PLC доступен только для чтения")* используется только для блоков данных. Установка этого атрибута приведет к тому, что Вы сможете только считывать данные в Вашей программе. При попытке записи в защищенный блок данных выводится сообщение об ошибке. Такой вариант защиты (защита от записи) нельзя путать с защитой блока. Блок данных с защитой блока может быть считан и перезаписан в пользовательской программе, но его данные нельзя отобразить ни с помощью программатора PG, ни с помощью устройств наблюдения оператора.

Блок данных с установленным атрибутом "*Unlinked*" ("*Неподключенный*") будет находиться только в загрузочной (load) памяти; он не может быть запущен на выполнение ("non execution-relevant"). Вы не сможете записывать в блоки данных, находящиеся в загрузочной памяти, и Вы сможете считать данные этих блоков только с помощью системной функции SFC 20 BLKMOV.

Прочие спецификации вкладки "General - Part 2" ("Общие - часть 2") окна свойств блока:

Атрибут *Name (Имя)* идентифицирует блок; это не то же самое, что и символьный адрес: разные блоки могут иметь одинаковое имя.

Атрибут *Family (Семья - имя группы - второе имя)* позволит Вам назначить общие характеристики для группы блоков. Идентификаторы блока *Name (Имя)* и *Family (Семья - имя группы)* отображаются при вставке блоков и при выборе блоков в диалоговом окне каталога элементов программы (program elements catalog).

Атрибут *Author (Автор)* идентифицирует создателя блока.

Атрибуты *Name (Имя), Family (Семья - имя группы), Author (Автор)* могут содержать до 8 символов (здесь могут применяться следующие символы: буквы, цифры и знак подчеркивания).

Атрибут *Version (Версия)* вводится дважды двумя цифрами: от 0 до 15.

На вкладке "General - Part 1" ("Общие - часть 1") редактор записывает дату изменения блока в две отметки времени: для кодового блока и для интерфейса, т.е. для блока параметров и для статических локальных данных.

Примечание: надо отметить, что дата изменения интерфейса должна быть такой же или более ранней, нежели дата изменения программного кода вызывающего блока. Если это условие не выполняется, то при выводе на экран вызывающего блока редактор сигнализирует об ошибке "time stamp conflict" ("конфликт отметок времени").

Блоки могут быть созданы или скомпилированы в виде версии 1 или в виде версии 2. Это имеет практическое значение только для функциональных блоков. Если активировано свойство "multi-instance capability" ("несколько экземпляров DB для функционального блока"), что, кстати, является обычным случаем, то мы имеем дело с блоком версии 2. Если свойство "multi-instance capability" выключено, то Вы не сможете вызвать этот блок как локальный экземпляр, также как Вы не сможете вызвать другой функциональный блок из этого блока как локальный экземпляр. Функциональный блок версии 1 имеет преимущество, заключающееся в ограничении использования экземпляра блока данных в случае косвенной адресации (имеет значение только при STL-программировании).

На вкладке "Calls" ("Вызовы") Вы увидите список всех блоков, вызываемых в данном блоке с отметками времени для кодовых блоков и интерфейса.

На вкладке "Attributes" ("Атрибуты") показаны системные атрибуты блока. С помощью системных атрибутов осуществляется координация и управление функциями разных приложений, например, в системе управления SIMATIC PCS7.

Program length (Размер программы)

Длина пользовательской программы содержится в свойствах (Properties) в автономном каталоге *Blocks (Блоки).* Для доступа к этим данным выберите *Blocks (Блоки)* и используйте опции: *Edit -> Object Properties (Правка -> Свойства объекта).* Теперь на вкладке "Blocks" ("Блоки") Вам доступна информация "Size in work memory" (Размер в рабочей памяти) и "Size in load memory" (Размер в загрузочной памяти).

Примечание: примите во внимание, что конфигурационные данные (системные блоки данных) не учитываются при указании размера программы в загрузочной (load) памяти. Открыв каталог *Blocks (Блоки),* Вы можете увидеть требования к загрузочной (load) памяти для системных данных в деталях (представленных в табличной форме). В строке состояния утилита SIMATIC Manager указывает суммарный объем памяти для всех блоков, которые Вы выберете с нажатой клавишей Ctrl.

С помощью программатора PG, подключенного интерактивно (online), при использовании утилиты SIMATIC Manager Вы можете найти текущие назначения памяти CPU на вкладке "Memory" ("Память"), используя опции меню: *PLC -> Module Information (PLC -> Информация о модуле).*

Контрольная сумма (Checksum)

Редактор программы Program Editor генерирует контрольную сумму (Checksum) для всех блоков пользовательской программы и сохраняет ее в свойствах объекта каталога *Blocks (Блоки).* Идентичные программы имеют одинаковую контрольную сумму, контрольная сумма изменяется при любом изменении программы.

Контрольная сумма также генерируется для системных данных. Для доступа к контрольным суммам при помощи утилиты SIMATIC Manager выберите каталог Blocks (Блоки) и используйте опции: *Edit -> Object Properties (Правка -> Свойства объекта).*

3.2.4 Интерфейс блоков (Block Interface)

Таблица объявления переменных содержит интерфейс блока с остальной программой. Он состоит из параметров блока (входы, выходы и входные и выходные параметры), а также статических локальных данных (для функциональных блоков). Временные локальные данные не принадлежат интерфейсу блока. Интерфейс блока определяется в таблице объявления переменных, и эти переменные инициализируются при вызове блока (см. главу 19. "Параметры блоков").

Редактор программ Program Editor проверяет, чтобы инициализация параметров вызываемого блока соответствовала интерфейсу вызываемого блока. Для этого редактор использует метки времени: интерфейс вызываемого блока должен иметь более раннюю временную метку, чем код вызывающего блока, что означает, что последние изменения интерфейса должны быть выполнены раньше его объединения с блоком. Редактор программ Program Editor обновляет метку времени интерфейса при изменении числа параметров, или при изменении типа данных, или при изменении значений параметров, принимаемых по умолчанию.

Конфликт временных меток (Time stamp conflict)

Если интерфейс вызываемого блока имеет более позднюю временную метку, чем код вызывающего блока, возникает "конфликт временных меток" ("Time stamp conflict"). Так, Вы получите "конфликт временных меток" ("Time stamp conflict"), если вновь откроете уже скомпилированный блок. В этом случае редактор Program Editor выделит некорректный вызов блока красным цветом. Конфликт временных меток также возникнет, если Вы, например, измените интерфейсы блоков, которые уже вызывались в других блоках, или если Вы объедините блоки из разных программ в новую программу, или если Вы перекомпилируете раздел полной программы из исходного файла.

Тем не менее, конфликт интерфейса, в общем описываемый как "конфликт временных меток" ("Time stamp conflict"), может также иметь другие причины. Он может случиться, если вызванный или адресованный (referenced) блок имеет более позднюю временную метку (younger), чем вызывающий блок. Ниже представлены примеры возможных случаев "конфликта временных меток" ("Time stamp conflict"):

- Интерфейс вызываемого блока имеет более позднюю временную метку (younger), чем код вызываемого блока.
- Интерфейс инициализации не согласован с интерфейсом блока.
- Функциональный блок имеет более позднюю временную метку (younger), чем его экземплярный блок данных (экземпляр DB генерируется на основе описания интерфейса функционального блока и должен, следовательно, иметь более позднюю временную метку, чем метка функционального блока, или их метки должны быть синхронны).
- Интерфейс локального экземпляра имеет более позднюю временную метку, чем вызывающий экземпляр (касается функциональных блоков).

- Пользовательский тип данных UDT имеет более позднюю временную метку (younger), чем блок, переменные которого объявлены как UDT; это может быть любой блок, включая блок данных или другой UDT.

Корректировка неправильных вызовов блока

Редактор программ Program Editor обеспечивает возможность исправления некорректных вызовов блока или UDT-приложений при выборе команд меню: *Edit -> Block Call -> Update (Правка -> Вызов блока -> Обновить).* Для случая одинаковых имен, типов данных или местоположения редактор может найти правильные назначения в большинстве случаев. Если этого не произошло, то Вы должны выполнить корректировку вручную. В любом случае Вы должны проверить правильность выполненной корректировки.

Check Block Consistency (Проверка блока на консистентность)

Редактор программ Program Editor лишь информирует о наличии "конфликта временных меток" ("Time stamp conflict"), если Вы открываете блок, содержащий этот самый "конфликт временных меток". Если необходимо проверить программу целиком, Вы можете использовать функцию проверки консистентности блока "Check Block Consistency". Эта функция снимает большинство конфликтов интерфейса и указывает на места в программе, требующие редактирования.

Для выполнения проверки на консистентность данных выберите каталог *Blocks (Блоки)* и затем опции меню: *Edit -> Check Block Consistency (Правка -> Проверки консистентности блока).* Редактор программ Program Editor генерирует данные, требующиеся для такой проверки, начиная с системы STEP 7 V5.0 SP3. Если пользовательская программа скомпилирована в системе STEP 7 более ранней версии или если программа содержит блоки, скомпилированные в системе STEP 7 более ранней версии (Вам необходимо будет проверить это, если соответствующая информация не отображается в окне функции "Check Block Consistency"), выбрав в этом окне: *Program -> Compile (Программа -> Компилирование)*.

Редактор программ Program Editor отобразит процесс выполнения задачи и результат проверки на консистентность в окне результата ("1:Compile"). Такая проверка консистентности не может быть использована для программ, находящихся в библиотеках.

Отношения в случае вызванных или адресованных блоков отображаются в форме древовидной диаграммы (рис. 3.4).

Вы можете выбирать между двумя представлениями, указанными ниже.

"Дерево ссылок" (*reference tree*) представляет связи аналогично отображению структуры программы: слева расположены вызывающие блоки, а правее расположены вызванные ими блоки. Пример: экземпляр DB 20 / FB 20 вызван в OB 1, а локальные экземпляры FB 21 и FB 22 вызваны в FB 20.

"Дерево подчиненности" (*dependency tree*) представляет связи, начиная от всех вызванных или адресованных блоков. Эти блоки расположены в левом ряду, а правее расположены вызывающие их блоки. Пример: FB 22 хранит свои данные в экземпляре DB 20 / FB 20, который вызывается в OB 1. Он также имеет свой собственный DB 29, и он вызывается как локальный экземпляр в FB 20.

Для обоих представлений случай появления знака восклицания (!) будет означать необходимость исправления и компиляции соответствующего блока.

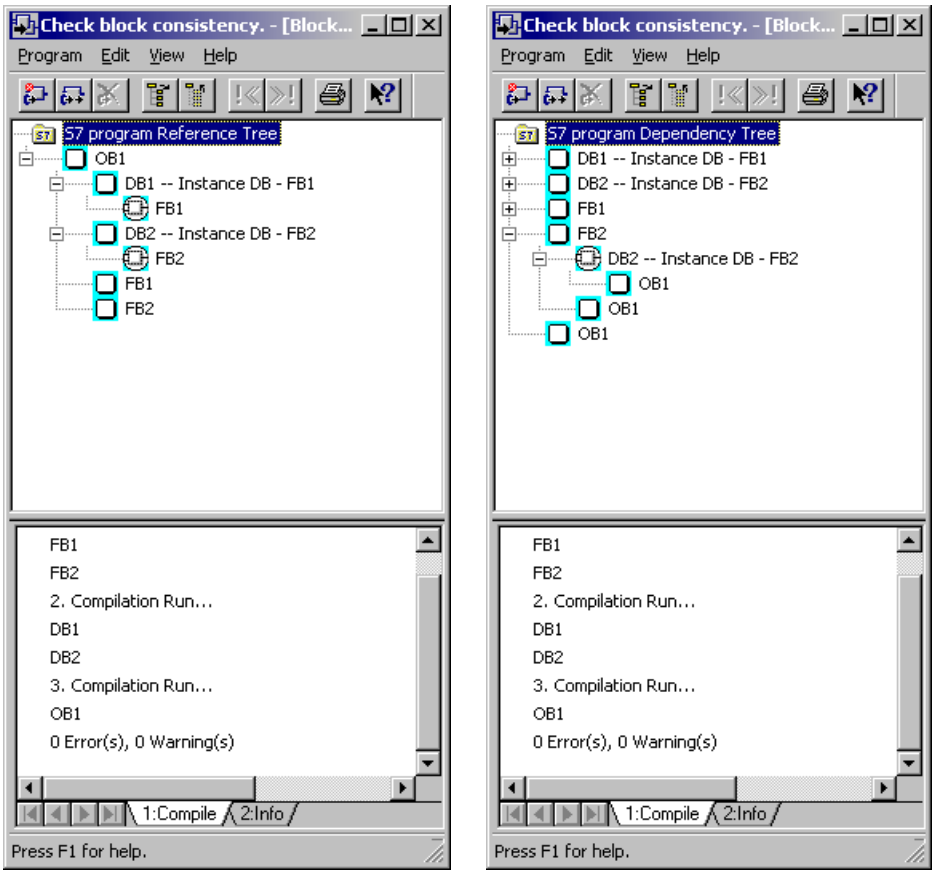


Рис. 3.4 Пример представления структуры отношений блоков по результатам проверки консистентности данных с помощью функции Check Block Consistency

Если Вы выбрали блок в древовидной схеме или в открытом окне, Вы сможете отредактировать его, выбрав соответствующие опции меню: *Edit -> Open Block (Правка -> Открыть блок)*, т.е. Вы можете исправить некорректный вызов.

3.3 Адресация переменных (Addressing Variables)

При адресации переменных Вы можете выбирать способ адресации из двух основных вариантов: абсолютная адресация (absolute addressing) или символьная адресация (symbol addressing). При абсолютной адресации используются численные адреса, начиная с нулевого (0) адреса для каждой адресной области. При символьной адресации используются символьные (состоящие из букв и цифр) имена, которые Вы сами задаете в таблице символов (Symbol Table) для глобальных адресов или в разделе объявления переменных (declaration section) для внутриблочной адресации. Расширением абсолютной адресации является косвенная адресация (indirect addressing), при которой адреса (местоположение) в памяти высчитываются во время выполнения программы.

3.3.1 Абсолютная адресация переменных

Переменные простых типов могут быть адресованы с использованием абсолютной адресации (absolute addressing).

Абсолютные адреса входов и выходов рассчитываются, исходя из начального адреса модуля, который Вы установили или должны установить в таблице конфигурации (configuration table), и типа сигнала, подключаемого к модулю. Подключаться могут как дискретные, так и аналоговые сигналы.

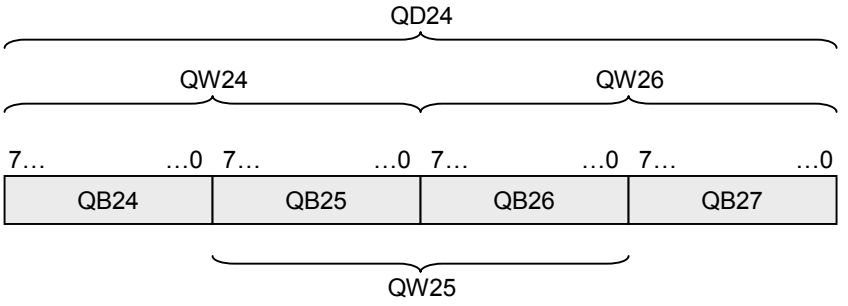


Рис. 3.5 Биты и байты в машинных словах и в двойном слове

Дискретные (binary) сигналы

Дискретный сигнал содержит один бит информации. Примерами дискретных сигналов являются входные сигналы от конечных выключателей, кнопок и т.п., которые поступают на дискретные входные модули, и выходные сигналы, управляющие лампами, контакторами и т.п., которые поступают на дискретные выходные модули.

Аналоговые сигналы

Аналоговый сигнал содержит 16 бит информации. Аналоговый сигнал соответствует "каналу" ("channel"), который занимает в контроллере машинное слово (word), т.е. 2 байта (см. ниже). Аналоговые входные сигналы (например, напряжения от терморезисторов) поступают на аналоговые входные модули, оцифровываются и после этого становятся доступными для обработки в контроллере в виде 16-разрядного сигнала (16 информационных битов). С другой стороны, 16-разрядный сигнал может управлять аналоговым индикатором посредством преобразования в аналоговом выходном модуле в аналоговый сигнал (например, ток). Динамическому диапазону изменения ("information width" - "информационный диапазон") сигнала соответствует динамический диапазон изменения ("information width") переменной, в виде которой сигнал сохраняется и обрабатывается. Динамический диапазон изменения сигнала и интерпретация этого сигнала (например, относительное положение), взятые вместе, определяют тип данных (*data tape*) для соответствующей переменной.

Дискретные сигналы сохраняются в переменных типа BOOL (булева переменная), аналоговые сигналы - в переменных типа INT (целая переменная).

Определяющим фактором для адресации переменной является ее тип, от которого зависит требуемая величина области памяти для размещения переменной.

В системе STEP 7 существуют 4 типа данных для абсолютной адресации:

- 1 бит Тип данных BOOL,
- 8 битов Тип данных BYTE или другой 8-битовый тип данных,
- 16 битов Тип данных WORD или другой 16-битовый тип данных,
- 32 бита Тип данных DWORD или другой 32-битовый тип данных.

На переменные типа BOOL ссылка производится посредством идентификатора адреса, номера байта и отделенного десятичной точкой номера бита. Нумерация байтов начинается с нуля (0) в каждой адресной области. Верхнее предельное значение номера байта определяется типом CPU. Биты внутри байтов нумеруются от 0 до 7.

Примеры:

I 1.0 входной бит с номером 0 в байте номер1

Q 16.4 выходной бит с номером 4 в байте номер16

Для переменных типа BYTE в качестве абсолютного адреса используется идентификатор адреса и номер байта, в котором содержится собственно значение переменной. Идентификатор адреса дополнен символом B.

Примеры:

IB 2 входной байт номер 2

QB 18 выходной байт номер 18

Переменные типа WORD состоят из двух байтов (слово). В качестве абсолютного адреса используется идентификатор адреса и номер младшего байта машинного слова, в котором содержится собственно значение переменной. Идентификатор адреса дополнен символом W.

Примеры:

IW 4 входное слово номер 4; содержит байты 4 и 5

QW 20 выходное слово номер 20; содержит байты 20 и 21

Переменные типа DWORD состоят из четырех байтов (двойное слово). В качестве абсолютного адреса используется идентификатор адреса и номер младшего байта двойного слова, в котором содержится собственно значение переменной. Идентификатор адреса дополнен символом D.

Примеры:

ID 8 входное двойное слово номер 8; содержит байты 8, 9, 10 и 11

QD 24 выходное двойное слово номер 24; содержит байты 24, 25, 26 и 27

Адресация области данных в блоке данных.

Примеры:

DB 10.DBX 2.0 бит данных 2.0 в блоке данных DB 10

DB 11.DBB 14 байт данных 14 в блоке данных DB 11

DB 20.DBW 20 слово данных 20 в блоке данных DB 20

DB 22.DBD 10 двойное слово данных 10 в блоке данных DB 22

Дополнительную информацию по адресации областей данных Вы найдете в разделе 18.2.2 "Адресация данных".

3.3.2 Косвенная адресация

Косвенная адресация (indirect addressing) позволяет рассчитывать адреса в области данных во время выполнения программы. Языки программирования STL и SCL используют различные методы для косвенной адресации. В STL различают следующие виды адресации:

- Косвенная адресация посредством памяти
  ("Memory-indirect-addressing")
  Например, IW [MD 200]
  означает, что адрес находится двойном слове памяти.
- Косвенная внутризонная адресация посредством регистра
  ("Register-indirect area-internal addressing")
  Например, IW [AR1, P#2,0]
  означает, что адрес, находящийся в адресном регистре AR1, получает приращение на величину смещения (offset) P#2,0 при выполнении оператора.
- Косвенная межзонная адресация посредством регистра
  ("Register-indirect area-crossing addressing")
  Например, W [AR1, P#0,0]
  означает, что адрес (включая адресную область), находящийся в адресном регистре AR1, получает приращение на величину смещения (offset) P#0,0 при выполнении оператора.

Двойные слова адресной области для данных (DBD и DID), меркеров (MD) и временных локальных данных (LD) могут использоваться для хранения адресов при косвенной адресации посредством памяти. Косвенную адресацию посредством регистра можно применять с использованием двух адресных регистров: AR1 и AR2.

Косвенная адресация подробно описана в главе 25 "Косвенная адресация".

При использовании языка программирования SCL адресные области состоят из поля, элементы которого доступны косвенно и отдельно. Например, MW[*index*] - это обращение к слову памяти, адрес которого размещен в переменной *index.* Переменная *index* может быть изменена в процессе выполнения программы. Более подробно косвенная адресация при использовании SCL рассматривается в разделе 27.2.3 "Косвенная адресация при использовании SCL".

3.3.3 Символьная адресация переменных

Символьная адресация (symbolic addressing) использует имена (символы) вместо абсолютных адресов. Вы сами можете выбирать эти имена. Такое имя должно начинаться с буквы и может содержать до 24 символов. В STL не разрешено использовать ключевые слова в качестве имен (символов). Для того, чтобы использовать ключевые слова в качестве имен (символов) в SCL, вставьте символ решетки "#" перед таким именем.

При присвоении имен входам учитывается регистр написания символа (имеет значение, какой регистр применяется: верхний или нижний). Для имен выходов редактор использует регистр и нотацию (форму записи), которые были применены при объявлении (declaration) символа.

При символьной адресации абсолютному адресу должно быть назначено имя (символ).

Символы различаются по месту назначения: глобальные символы действительны во всей программе, тогда как локальные символы действительны только в блоке, в разделе объявления переменных которого они описаны.

Глобальные символы

Вы можете назначить имена в таблице символов (symbol table) следующим объектам:

- Блоки данных и кодовые блоки
- Входы, выходы, периферийные входы и периферийные выходы
- Меркеры, таймеры и счетчики
- Пользовательские типы данных
- Таблицы переменных

Глобальный символ может также содержать пробелы, специальные символы и национальные символы, такие как умляут. Исключения составляют символы 00_{hex}, FF_{hex} и кавычки ("). При использовании в программе имен со специальными символами Вы должны заключать имена в кавычки. В скомпилированном блоке программный редактор всегда отображает все глобальные символы в кавычках.

Вы можете использовать глобальный символ во всей программе; каждый такой символ должен быть уникальным (однозначно принадлежать одному адресу) в этой программе.

Редактирование, импортирование и экспортирование глобальных символов описано в разделе 2.5.2 "Таблица символов".

Локальные символы

Имена локальных данных определяются в разделе объявления переменных соответствующего блока. Эти имена могут содержать только буквы, цифры и знак подчеркивания.

Локальные символы являются действующими только внутри блока, в котором они описаны. Такие же символы (такие же имена переменных) могут быть применены в ином контексте (для обозначения совершенно иных объектов) в другом блоке. Редактор отображает локальные символы (имена) с впереди стоящим символом "#". Если редактор не может отличить локальный символ от адреса Вы должны вводить этот символ с впереди стоящим символом "#".

Локальные символы доступны только в базе данных программатора PG (в автономном [offline] каталоге *Blocks* [*Блоки*]). Если эта информация отсутствует при декомпиляции, то редактор вставляет символы замены (substitute symbol).

Использование символьных имен

Если Вы используете символьные имена во время инкрементного программирования, то эти имена должны уже к этому времени быть присвоены абсолютным адресам. Вы можете ввести новые символические имена в таблицу символов во время инкрементного программирования и можете в дальнейшем использовать в программе.

Если вы программируете исходный текстовый файл программы, то полностью закончить процесс назначения символических имен абсолютным адресам необходимо лишь к моменту компиляции программы.

В случае использования массивов доступ к отдельным элементам массивов обеспечивается использованием имени массива с индексом, например, имя MSERIES[1] принадлежит первому элементу массива MSERIES. В случае программирования на STL индекс должен быть константой (INT). В случае программирования на SCL индекс может быть как целой переменной (INT), так и целым выражением (INT).

В структурах каждый элемент имени ("subname") отделяется от остальных элементов десятичной точкой, например, FRAME.HEADER.CNUM.

Компоненты пользовательских типов данных адресуются точно также как и компоненты структур.

Подробная информация изложена в главе 24 "Типы данных".

Адресация данных

Символьная адресация данных предполагает использование полного адреса. включая блока данных. Например, блок данных с символьным адресом MVALUES содержит переменные MVALUE1, MVALUE2 и MTIME. Эти переменные могут быть адресованы следующим образом:

"MVALUES". MVALUE1

"MVALUES". MVALUE2

"MVALUES". MTIME

Для получения дополнительной информации по использованию адресов для доступа к данным обратитесь к разделам 18.2.2 "Адреса для доступа к данным" (STL) и 27.2.2 "Символьная адресация" (SCL).

3.4 Программирование кодовых блоков на STL

3.4.1 Структура STL-выражения

STL-программа состоит из ряда отдельных выражений (statement). Выражение - это наименьшая самостоятельная единица пользовательской программы. Выражение содержит описание работы для CPU. На рисунке 3.6 показана общая структура STL-выражения.

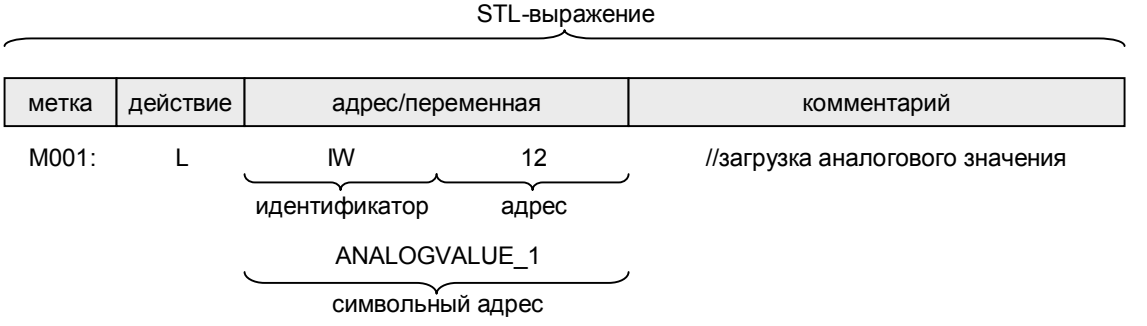


Рис. 3.6 Структура STL-выражения

Ниже перечислены компоненты STL-выражения:

- Метка (не обязательный элемент) содержит до 4 символов, заканчивается двоеточием ":" (см. раздел "Функции перехода").
- Описание задания для CPU (такие задания, например, как load [загрузить], scan [считать], compare [сравнить] и т.д.).
- Адрес - информация, необходимая для выполнения действия (например, абсолютный адрес IW12, символьный адрес некоторой переменной ANALOGVALUE_1 или некоторой константы W#16#F001 и т. д.). Отдельные операторы не требуют задания адреса.
- Комментарий (не обязательный элемент) должен начинаться двумя косыми чертами "//" и может продолжаться до конца строки.

При вводе в исходный файл Вы должны заканчивать каждое выражение (до начала комментария, если он есть) символом "точка с запятой (";"). В STL строка может содержать не более 200 символов, а длина комментария не может быть больше 160 символов.

3.4.2 Инкрементное программирование кодовых блоков на STL

В разделе 2.5 "Создание S7-программ" Вы можете найти введение в основы создания S7-программы и использование программного редактора.

Создание блоков

Процесс программирования блока начинается с его открытия одним из двух способов: либо двойным щелчком на блоке в окне проекта SIMATIC Manager, либо в редакторе с помощью выбора опций меню: *File -> Open (Файл -> Открыть).* Если блок пока не существует, Вы должны его сначала создать одним из следующих путей:

- В левой половине окна проекта SIMATIC Manager выберите объект *Blocks (Блоки),* создайте новый блок с помощью опций меню: *Insert->S7 Block->… (Вставка -> S7 Block -> …)*. В окне свойств (Properties) блока на вкладке "General - Part 1" ("Общие - часть 1") выберите номер блока и язык программирования "STL".
- Находясь в редакторе, с помощью опций меню: *File -> New (Файл -> Создать)* вызовите окно диалога с заголовком блока (номер блока, язык программирования, атрибуты блока). После закрытия диалогового окна Вы можете вводить программу этого блока.

Вы можете ввести информацию заголовка блока либо при создании блока, либо позднее, активизировав редактор, затем открыв блок и выбрав опции меню: *File -> Properties (Файл -> Свойства)*.

В редакторе программ язык программирования устанавливается на вкладке "Create Block" ("Создать блок") в диалоговом окне, открытом с помощью опций меню: *Options -> Customize (Опции -> Установки пользователя)*.

Окно блока

На нижеприведенном рисунке представлен пример открытого блока STL-программы (см. рис. 3.7).

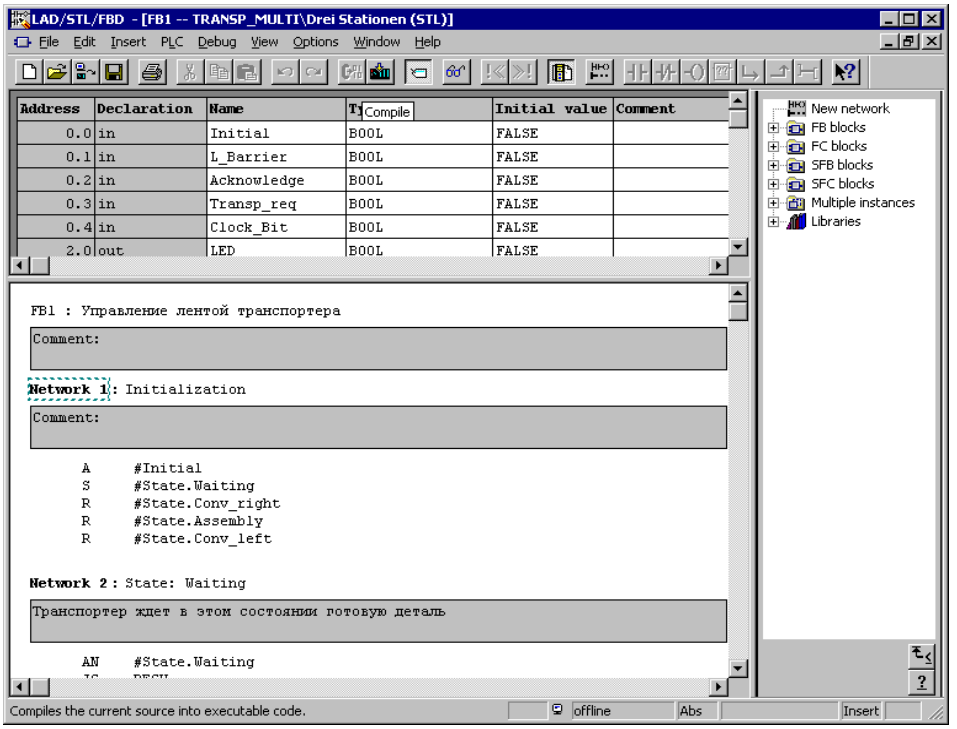


Рис.3.7 Пример открытого STL-блока

Когда открывается кодовый блок, появляется окно блока, которое само по себе состоит из трех частей:

- В верхней части окна блока отображается окно таблицы объявления переменных. Именно здесь Вы определяете внутриблочные (локальные) переменные данного блока.
- Ниже таблицы объявления переменных отображается окно программы. В этом окне Вы вводите текст Вашей программы для блока.
- В правой части окна блока отображается окно каталога элементов программы. В STL этот каталог содержит только блоки, которые находятся в автономном (offline) каталоге *Blocks (Блоки)*, а также уже запрограммированные экземпляры мультиэкземплярных FB и доступные библиотеки.

Таблица объявления переменных

Таблица объявления переменных располагается выше окна программы. Если она не видна, то поместите курсор на верхней разделительной с окном программы линии, щелкните левой кнопкой мыши, когда курсор изменит свою форму, и "потащите" курсор. Вы увидите окно таблицы объявления переменных, в которой Вы должны дать описание для локальных переменных блока (см. таблицу 3.2). Существуют типы переменных, которые не могут использоваться в некоторых типах кодовых блоков. Если Вы не используете данный тип переменных, соответствующая строка должна остаться незаполненной.

Таблица 3.2 Типы переменных в разделе объявления переменных

Variable type (Тип переменной)	Declaration (Описание)	Тип переменной возможен в блоках типов		
Input parameters (Входные параметры)	in	-	FC	FB
Output parameters (Выходные параметры)	out	-	FC	FB
In-out parameters (Вх/вых параметры)	in_out	-	FC	FB
Static local data (Статические локальные данные)	stat	-	-	FB
Temporary local data (Временные локальные данные)	temp	OB	FC	FB

Описание переменных состоит из имени, типа данных, значения по умолчанию (если есть) и комментария (не обязательный элемент). Не всем переменным может быть назначено значение по умолчанию (например, значения по умолчанию не могут присваиваться временным локальным данным). Более подробно значения, назначаемые по умолчанию для функций и функциональных блоков, описаны в главе 19 "Параметры блока".

Порядок описаний в кодовом блоке фиксирован (как показано в таблице выше), в то же время порядок следования типов переменных произволен. Вы можете экономно расходовать память, если будете группировать двоичные переменные в блоки по 8 или 16 штук, а переменные типа BYTE - парами. Редактор сохраняет новые переменные типов BOOL или BYTE, выделяя им байтовые участки памяти. Для всех остальных типов переменных выделяются участки памяти размером в 1 слово (начиная с байта с четным адресом).

Окно программы

В окне программы Вы увидите (в зависимости от установок редактора, принятых по умолчанию) поля для заголовка и комментария блока и, если это первый сегмент, то поля для заголовка и комментария сегмента, а также поле для ввода программы. В разделе программы кодового блока Вы можете управлять отображением комментариев и символов (имен) с помощью опций меню: *View -> Comment (Вид -> Комментарий)*, *View -> Symbolic Representation (Вид -> Представление символов)*, *View -> Symbol Information (Вид -> Информация о символе)*. Также Вы можете менять размер отображения с помощью опций меню: *View -> Zoom In (Вид -> Увеличить масштаб)*, *View -> Zoom Out (Вид -> Уменьшить масштаб) View -> Zoom Factor (Вид -> Коэффициент масштабирования)*.

Вы можете разделить STL-программу на сегменты. Редактор нумерует сегменты автоматически, начиная с 1. Вы можете дать каждому сегменту заголовок сегмента (*network title*) и комментарий сегмента (*network comment*). Во время редактирования Вы можете выбирать каждый сегмент непосредственно с помощью опций меню: *Edit -> Go To -> ...(Правка -> Перейти к ->…)*. Сегментирование не является обязательным приемом.

Для начала ввода программного кода щелкните один раз кнопкой мыши ниже поля для комментария сегмента или, если Вы установили режим отображения "Display with Comments" ("Отображать с комментариями"), тогда щелкните один раз кнопкой мыши ниже выделенного тенью поля для комментария. Перед Вами пустая рамка окна. Здесь Вы можете вводить программу в любом месте внутри этого окна. Обратитесь к разделу 3.4.1 "Структура STL-выражения" для получения информации о структуре STL-выражения.

Отделите оператор (OP-code [operator]) от адреса (операнд [operand]) одним или несколькими пробелами или одним шагом табуляции. После адреса в этой же строке Вы можете ввести две косых черты, после которых введите комментарий к выражению. Закончите выражение, нажав клавишу <Enter>. Вы можете также ввести целую строку комментария, начав новую строку с двойной косой черты "//".

Новый сегмент можно запрограммировать, выбрав опции меню: *Insert -> Network (Вставка -> Сегмент).* При этом редактор вставляет пустой сегмент после выбранного сегмента.

Если необходимо использовать символьные имена при инкрементном программировании, эти имена к моменту их использования в программе должны быть уже назначены абсолютным адресам.

Вы можете вызывать таблицу символов для выбора из нее символьных имен с помощью опций меню: *Insert -> Symbol (Вставка -> Символ)*. После вызова таблицы символов требуемый символ переносится в программу после щелчка на нем кнопкой мыши.

При инкрементном программировании Вы также можете вносить новые символьные имена в таблицу символов или корректировать имена, ранее в нее внесенные. Вы можете вызывать таблицу символов целиком с помощью опций меню: *Option -> Symbol Table (Опции -> Таблица символов)*, также Вы можете вызывать одну строку из таблицы символов для редактирования с помощью опций меню: *Edit -> Symbol (Правка -> Символ).* После редактирования или ввода нового символьного имени Вы можете использовать его, продолжив ввод своей программы.

Нет необходимости завершать блок специальным выражением. Тем не менее, Вы можете запрограммировать последний "пустой" сегмент с заголовком "Block End" ("Конец блока"), облегчая тем самым зрительное восприятие программы (это может быть полезно в случае особо длинных блоков).

Если с помощью редактора открывается ранее скомпилированный блок, этот блок "декомпилируется", т.е. для него генерируется STL-код. Для этого редактор использует разделы программы в базе данных программатора PG, которые не совсем соответствуют программе, например, в плане символов, комментариев и меток перехода. Если нужная информация в базе данных программатора PG отсутствует во время декомпиляции программы, редактор использует подставленные символы (substitute symbols).

Вы можете в редакторе создать новые блоки или открыть и отредактировать существующие без необходимости перехода в утилиту SIMATIC Manager.

Каталог элементов программы

Если каталог элементов программы не видим, активизируйте его с помощью опций меню: *View -> Catalog (Вид -> Каталог)*.

Каталог элементов программы располагается в своем собственном окне в правой части окна редактора. Каталог можно убрать с экрана, если дважды щелкнуть кнопкой манипулятора "мышь" на заголовке окна каталога.

Каталог элементов программы поддерживается при программировании на языках LAD и FBD, обеспечивая доступные графические элементы. При программировании на языке STL этот каталог показывает только блоки, которые находятся в автономном (offline) каталоге *Blocks (Блоки)*, а также уже запрограммированные экземпляры мультиэкземплярных FB и доступные библиотеки.

3.4.3 Программирование кодовых блоков на STL, ориентированное на создание исходных файлов

В разделе 2.5 "Создание S7-программ" Вы можете найти введение в основы создания S7-программы и использование программного редактора. Процесс программирования блока, ориентированный на создание исходных файлов, начинается с создания пустого файла исходной программы в SIMATIC Manager (см. раздел 2.5.3 "Редактор STL-программ [STL Program Editor] под подзаголовком "Программирование, ориентированное на создание исходных файлов").

Теперь Вы можете запустить редактор, открыв исходный файл, и можете немедленно начать вводить программу, например, с помощью ключевого слова для функционального блока.

В таблице 3.3 приведены ключевые слова, требующиеся при программировании блоков, а также порядок их использования.

Заголовок блока

Вы должны запрограммировать свойства блока в его заголовке после указания типа блока и перед разделом объявления переменных. Вся информация в заголовке блока опциональна (optional); Вы можете пропустить или отдельные характеристики, или все характеристики без исключения. Для получения информации по описанию и назначению свойств блоков обратитесь к разделу 3.2.3 "Свойства блоков".

С помощью ключевого слова "TITLE =" сразу же после строки с указанием типа блока Вы можете ввести заголовок блока длиной до 64 символов. Вы можете вслед за тем добавить комментарий в одной или нескольких строках, начинающихся двойной косой чертой. Комментарий блока может иметь размер до 18 Кбайт.

Описание переменных

Раздел объявления переменных содержит определения всех внутриблочных переменных, т.е. таких переменных, которые Вы применяете только в этом блоке. Вы не можете использовать любые типы переменных в любом блоке (см. таблицу 3.3). Если Вы не используете какие-либо типы переменных, пропустите соответствующие описания, включая ключевые слова.

Описание переменной состоит из имени, типа данных, значения по умолчанию (если есть) и комментария переменной (необязательный элемент).

Например:

Quantity : INT := +500; //Units per batch (единиц на пакет)

Таблица 3.3 Ключевые слова для программирования кодовых блоков на STL

Блок	Организационный блок	Функциональный блок	Функция
Тип блока	ORGANIZATION_BLOCK	FUNCTION_BLOCK	FUNCTION : *значение*
Заголовок [Header]	TITLE = *Заголовок блока* *//Комментарий блока* KNOW_HOW_PROTECT NAME : *Имя блока* FAMILY : *Второе имя* AUTHOR : *Автор* VERSION : *Версия*	TITLE = *Заголовок блока* *//Комментарий блока* CODE_VERSION1 KNOW_HOW_PROTECT NAME : *Имя блока* FAMILY : *Второе имя* AUTHOR : *Автор* VERSION : *Версия*	TITLE = *Заголовок блока* *//Комментарий блока* KNOW_HOW_PROTECT NAME : *Имя блока* FAMILY : *Второе имя* AUTHOR : *Автор* VERSION : *Версия*
Описание [Declaration]		VAR_INPUT *Входные параметры* END_VAR	VAR_INPUT *Входные параметры* END_VAR
		VAR_OUTPUT *Выходные параметры* END_VAR	VAR_OUTPUT *Выходные параметры* END_VAR
		VAR_IN_OUT *Вх/Вых параметры* END_VAR	VAR_IN_OUT *Вх/Вых параметры* END_VAR
		VAR *Статические локальные данные* END_VAR	
	VAR_TEMP *Временные локальные данные* END_VAR	VAR_TEMP *Временные локальные данные* END_VAR	VAR_TEMP *Временные локальные данные* END_VAR
Программа [Program]	BEGIN NETWORK TITLE = *Заголовок сегмента* *//Комментарий* *//сегмента* *…STL-операторы* *//Комментарий строки* NETWORK *и т.д.*	BEGIN NETWORK TITLE = *Заголовок сегмента* *//Комментарий* *//сегмента* *…STL-операторы* *//Комментарий строки* NETWORK *и т.д.*	BEGIN NETWORK TITLE = *Заголовок сегмента* *//Комментарий* *//сегмента* *…STL-операторы* *//Комментарий строки* NETWORK *и т.д.*
Конец блока [Block end]	END_ORGANIZATION_BLOCK	END_FUNCTION_BLOCK	END_FUNCTION

Не всем переменным могут быть назначены значения по умолчанию (не могут значения по умолчанию назначаться, например, для временных локальных данных). Назначения по умолчанию для функций и функциональных блоков детально описаны в главе 19 "Параметры блоков".

Порядок описаний в кодовых блоках является регламентированным (см. таблицу 3.3), в то же время порядок следования типов переменных в разделе описания переменных произволен, при этом порядок следования типов переменных влияет (с учетом типов данных) на расходование памяти. В главе 24 "Типы данных" показано, как оптимизировать распределение памяти, правильно планируя порядок размещения данных.

Раздел программы

Раздел программы кодового блока начинается с ключевого слова BEGIN и заканчивается ключевым словом END_xxx, в котором xxx заменяется, в зависимости от типа блока, на ORGANIZATION_BLOCK, FUNCTION_BLOCK или FUNCTION. Ключевое слово END_xxx заменяет Block End BE.

И в ключевых словах, и в тексте программы редактор различает, какой регистр используется (верхний или нижний). Подробнее о синтаксисе выражений Вы можете прочитать в разделе 3.4.1 "Структура STL-выражения". OP-код (оператор) может быть отделен от адреса (операнд) одним или несколькими пробелами или шагами табуляции. Для улучшения читаемости исходного текста программы Вы можете оставлять один или несколько пробелов (и/или шагов табуляции) между словами. Вы должны заканчивать каждое выражение точкой с запятой ";". После точки с запятой Вы можете записать комментарий, который должен начинаться с двойной косой черты "//". Комментарий может продолжаться до конца строки. Вы можете помещать несколько выражений в одной строке, разделяя их точкой с запятой ";". Вы также можете записывать комментарии с начала строки, помещая в начале строки двойную косую черту "//". Строка комментария не может содержать более 160 символов; она не может содержать символов табуляции и непечатаемых символов.

Для улучшения читаемости и логики программы Вы можете разбить программу блока на сегменты (*network*). В графических языках (с графической интерпретацией) такое разбиение обязательно, в языке STL - необязательно. Сегменты в STL не имеют функционального назначения; они используются здесь просто для разбиения программы на большее количество логически связанных частей и для улучшения ее читаемости, а также чтобы упростить и сделать более эффективным написание комментариев. В очень больших программах сегментирование программы дает преимущество, заключающееся в том, что возможен прямой доступ к сегментам в скомпилированном блоке, что способствует быстрому доступу к отдельным точкам в программе посредством опций: *Edit -> Go To -> ... (Правка -> Перейти к->…).* Так, Вы можете задавать номер сегмента или номер строки, относящийся к началу сегмента.

Сегменты начинаются с ключевого слова NETWORK; ключевое слово "TITLE=" в следующей строке позволяет задавать сегменту заголовок длиной до 64 символов. Строка комментариев, следующая сразу за заголовком, образует комментарий сегмента, вмещающий до 18 Кбайт информации. Редактор STL нумерует сегменты автоматически, начиная с номера 1. В каждом блоке может находиться до 999 сегментов. Всего пользователю доступно 64 Кбайта памяти для комментариев блока и сегмента в каждом блоке.

Порядок блоков при программировании, ориентированном на создание исходных файлов программы

Для вызова блока редактор требует информацию из заголовка блока, заданные параметры блока, заявленный тип, а также тип данных параметров блока. Это значит, что Вы должны сначала запрограммировать вызываемые функции и функциональные блоки, то есть Вы должны начать программирование программы с блоков "самого нижнего уровня" (имеется в виду положение блоков относительно начала программы в исходном файле).

Однако, достаточно запрограммировать заголовки блоков и задать их параметры (то есть задать описание интерфейса ["interface description"]), чтобы не было ошибки при вызове блока. В дальнейшем Вы можете снабдить этот "интерфейс" программой, Однако необходимо сохранять без изменения интерфейс уже вызванного блока! Иначе редактор выдаст сообщение о конфликте временных меток при вызове блока.

В случае большой пользовательской программы Вы будете, очевидно, разбивать исходный файл программы на отдельные легкоуправляемые файлы, например, на "стандартные подпрограммы", которые можно использовать неоднократно по всей программе, технологически- или функционально-законченные подпрограммы и главную программу (main program), которая содержит, как правило, организационные блоки. При создании отдельных исходных файлов, Вы должны следить за порядком компилирования, для соблюдения правил вызовов блоков, приведенных выше. Рекомендуется использовать следующий порядок компилирования:

- Пользовательские типы данных UDT
- Блоки глобальных данных
- Функции и функциональные блоки, начиная с блоков самого низкого уровня вызовов
- Экземплярные блоки данных (эти блоки могут также быть расположены непосредственно за назначенным функциональным блоком)
- Организационные блоки

Пример функционального блока с экземплярным блоком данных

На рисунке 3.8 показан пример функционального блока со статическими локальными данными, за которым следует экземплярный блок данных, связанный с этим функциональным блоком.

3.5 Программирование кодовых блоков на SCL

3.5.1 Структура SCL-выражения

SCL-программа состоит из ряда отдельных выражений (statement). Выражение - это наименьшая самостоятельная единица программы пользователя. Выражение содержит описание работы для CPU. На рисунке 3.9 показаны несколько примеров SCL-выражений.

В составе SCL-выражения можно выделить следующие компоненты:

- Метка перехода (необязательный элемент), содержащая до 24 символов и заканчивающаяся двоеточием ":". Метки перехода должны быть описаны.
- Инструкция, описывающая задание для CPU (например, присвоение значений, оператор управления и т.д.)
- Комментарий (необязательный элемент), начинающийся двойной косой чертой "//", может продолжаться до конца строки и содержать только печатаемые символы (кроме табуляции).

Каждое SCL-выражение должно завершаться точкой с запятой ";" (перед комментарием). SCL-выражение может содержать до 126 символов.

```
FUNCTION_BLOCK V_Memory
TITLE = Intermediate buffer  for 4 values
//В заголовке: промежуточный буфер на 4 значения
//Пример блока FB с параметрами и статическими локальными данными на STL
AUTHOR   : Berger
FAMILY   : STL_Book
NAME     : Memory
VERSION  : 01.00
VAR_INPUT
 Transfer    : BOOL := FALSE;//Пересылка положительного фронта сигнала
 Input_value : REAL := 0.0;  //в формате данных REAL  (дробное число)
END_VAR
VAR_OUTPUT
 Output_value : REAL := 0.0; //в формате данных REAL  (дробное число)
END_VAR
VAR
 Value1 : REAL := 0.0; //первое сохраненное значение в формате данных REAL
 Value2 : REAL := 0.0; //второе значение
 Value3 : REAL := 0.0; //третье значение
 Value4 : REAL := 0.0; //четвертое значение
 Edge_memory_bit : BOOL := FALSE;//меркер фронта передаваемого сигнала
END_VAR
BEGIN
NETWORK
TITLE = Program for transfer and output
//Передача и вывод имеют место, если в Transfer положительный фронт сигнала
  A     Transfer;        // если Transfer устанавливается в значение "1"
  FP    Edge_memory_bit; // RLO устанавливается в "1" вслед за FP
  JCN   End;             // переход, если нет положительного фронта
//Передача значений, начиная с последнего
  L     Value4;
  T     Output_value;    //Передача последнего значения
  L     Value3;
  T     Value4;
  L     Value2;
  T     Value3;
  L     Value1;
  T     Value2;
  L     Input_value;     //Передача входного значения
  T     Value1;
End: BE;
END FUNCTION BLOCK



DATA_BLOCK Values1
TITLE = Instance data block for "V_Memory"
//Пример экземплярного блока данных для FB "V_Memory"
AUTHOR    : Berger
FAMILY    : STL_Book
NAME      : V_MEM_DB1
VERSION   : 01.00
  V_Memory          //экземпляр для FB "V_Memory"
BEGIN
  Value1  := 1.0;   //Индивидуальные присвоения для
  Value2  := 1.3;   //отдельных значений
END_DATA_BLOCK
```

Рис. 3.8 Пример программирования функционального STL-блока со связанным экземпляром блока данных

```
Value Assignments      (присвоение значений)

Power      := Voltage * Current;
TooLarge   := Volt_Act > Volt_Set;
Switch_on  := Manual_on OR Auto_on;

Control Statements     (операторы управления)

IF Input_value > Maximum
  THEN Delimiter    := Maximum;
  ELSIF Input_value  < Minimum
    THEN Delimiter  := Minimum;
  ELSE Delimiter    := Input_value;
END_IF;
FOR i := 1 TO 32 DO
   Measure_value[i] := 0;
END_FOR;

Function Calls         (вызовы функций)

Result  := Delimiter(
   Input_value:= Actual_value,
   Minimum    := Lower_limit,
   Maximum    := Upper_limit)
```

Рис. 3.9 Примеры STL-выражений

3.5.2 Программирование кодовых SCL-блоков

В разделе 2.5 "Создание S7-программ" Вы можете найти введение в основы создания S7-программы и использование программного редактора.

Процесс программирования блока начинается с создания пустого файла исходной программы в SIMATIC Manager (см. раздел 2.5.4 "Редактор SCL-программ [SCL Program Editor] под подзаголовком "Программирование исходного SCL-файла").

Теперь Вы можете запустить редактор, открыв исходный файл, и можете немедленно начать вводить программу, например, с помощью ключевого слова для функционального блока.

В таблице 3.4 приведены ключевые слова, требующиеся при программировании блоков, а также порядок их использования.

Заголовок блока

Вы должны запрограммировать свойства блока в его заголовке после указания типа блока и перед разделом объявления переменных. Вся информация в заголовке блока опциональна (optional); Вы можете пропустить или отдельные характеристики, или все характеристики без исключения. Для получения информации по описанию и назначению свойств блоков обратитесь к разделу 3.2.3 "Свойства блоков".

С помощью ключевого слова "TITLE =" сразу же после строки с указанием типа блока Вы можете ввести заголовок блока длиной до 64 символов. Вы можете вслед за тем добавить комментарий в одной или нескольких строках, начинающихся двойной косой чертой "//". Комментарий блока может иметь размер до 18 Кбайт.

Таблица 3.4 Ключевые слова для программирования кодовых блоков на SCL

Блок	Организационный блок	Функциональный блок	Функция
Тип блока	ORGANIZATION_BLOCK	FUNCTION_BLOCK PROGRAM[3]	FUNCTION : *значение функции*
Заголовок [Header]	TITLE = *Заголовок блока* *//Комментарий блока* KNOW_HOW_PROTECT NAME : *Имя блока* FAMILY : *Второе имя* AUTHOR : *Автор* VERSION : *Версия*	TITLE = *Заголовок блока* *//Комментарий блока* CODE_VERSION1 KNOW_HOW_PROTECT NAME : *Имя блока* FAMILY : *Второе имя* AUTHOR : *Автор* VERSION : *Версия*	TITLE = *Заголовок блока* *//Комментарий блока* KNOW_HOW_PROTECT NAME : *Имя блока* FAMILY : *Второе имя* AUTHOR : *Автор* VERSION : *Версия*
Описание [Declaration]		VAR_INPUT *Входные параметры* END_VAR	VAR_INPUT *Входные параметры* END_VAR
		VAR_OUTPUT *Выходные параметры* END_VAR	VAR_OUTPUT *Выходные параметры* END_VAR
		VAR_IN_OUT *Вх/Вых параметры* END_VAR	VAR_IN_OUT *Вх/Вых параметры* END_VAR
		VAR *Статические локальные данные* END_VAR	VAR[1] *Временные локальные данные* END_VAR
	VAR_TEMP *Временные локальные данные* END_VAR	VAR_TEMP *Временные локальные данные* END_VAR	VAR_TEMP *Временные локальные данные* END_VAR
	CONST *Константы* END_CONST	CONST *Константы* END_CONST	CONST *Константы* END_CONST
	LABEL *Метки перехода* END_LABEL	LABEL *Метки перехода* END_LABEL	LABEL *Метки перехода* END_LABEL
Программа [Program]	BEGIN[2] *…SCL-операторы* *//Комментарий строки* (**Комментарий блока …* *…Комментарий блока*)* *… и т.д.*	BEGIN[2] *…SCL-операторы* *//Комментарий строки* (**Комментарий блока …* *…Комментарий блока*)* *… и т.д.*	BEGIN[2] *…SCL-операторы* *//Комментарий строки* (**Комментарий блока …* *…Комментарий блока*)* *… и т.д.*
Конец блока [Block end]	END_ORGANIZATION_ BLOCK	END_FUNCTION_BLOCK END_PROGRAM[3]	END_FUNCTION

[1] Локальные данные под ключевым словом VAR в SCL-функции FC используются как временные локальные данные (VAR_TEMP).

[2] Не требуется в SCL.

[3] Альтернативный вариант функциональному блоку: FUNCTION_BLOCK и END_FUNCTION_BLOCK.

Описание переменных

Раздел объявления переменных содержит определения всех внутриблочных переменных, т.е. таких переменных, которые Вы применяете только в этом блоке. Вы не можете использовать любые типы переменных в любом блоке (см. таблицу 3.4). Если Вы не используете какие-либо типы переменных, пропустите соответствующие описания, включая ключевые слова.

Описание переменной состоит из имени, типа данных, значения по умолчанию (если есть) и комментария переменной (необязательный элемент).

Например:

Quantity : INT := +500; //Units per batch (единиц на пакет)

При описании переменных допускается группировать переменные одного типа в одной строке, например:

Value1, Value2, Value3, Value4 : INT;

Не всем переменным могут быть назначены значения по умолчанию (не могут значения по умолчанию назначаться, например, для временных локальных данных). Назначения по умолчанию для функций и функциональных блоков детально описаны в главе 19 "Параметры блоков".

Порядок описаний в кодовых блоках является регламентированным (см. таблицу 3.4), в то же время порядок следования типов переменных в разделе описания переменных произволен, при этом порядок следования типов переменных влияет (с учетом типов данных) на расходование памяти. В главе 24 "Типы данных" показано, как оптимизировать распределение памяти, правильно планируя порядок размещения данных.

При программировании на SCL Вы можете объявлять константы, то есть можно назначать символическое имя фиксированным значениям.

Если в SCL-программе Вы используете метки перехода, то Вы также должны объявить их.

Раздел программы

Раздел программы кодового SCL-блока может начинаться с ключевого слова BEGIN (опционально) и заканчивается ключевым словом END_xxx, в котором xxx заменяется, в зависимости от типа блока, на ORGANIZATION_BLOCK, FUNCTION_BLOCK или FUNCTION. Ключевое слово END_xxx заменяет Block End (BE).

И в ключевых словах, и в тексте программы редактор различает, какой регистр используется (верхний или нижний). Подробнее о синтаксисе выражений Вы можете прочитать в разделе 3.5.1 "Структура SCL-выражения". OP-код (оператор) может быть отделен от адреса (операнд) одним или несколькими пробелами или шагами табуляции. Для улучшения читаемости исходного текста программы Вы можете оставлять один или несколько пробелов (и/или шагов табуляции) между словами.

Вы должны заканчивать каждое выражение точкой с запятой ";". После точки с запятой Вы можете записать комментарий, который должен начинаться с двойной косой черты "//". Комментарий может продолжаться до конца строки. Вы можете помещать несколько выражений в одной строке, разделяя их точкой с запятой ";".

SCL-блок должен содержать по крайней мере одно SCL-выражение (один знак точки с запятой ";"). SCL-программа не имеет сегментов в отличие от STL-программы.

Вы также можете записывать комментарии с начала строки, помещая в начале строки двойную косую черту "//". Строка комментария не может содержать более 160 символов; она не может содержать символов табуляции и непечатаемых символов.

В SCL Вы можете создавать комментарий блока, который может занимать несколько строк. Он начинается открывающей скобкой и звездочкой "(*" и заканчивается звездочкой и закрывающей скобкой "*)". Комментарий блока может также помещаться внутри SCL-выражения; однако он не может "разрывать" символических имен или констант (исключение: строка символов).

Порядок блоков при программировании, ориентированном на создание исходных файлов программы

Для вызова блока редактор требует информацию из заголовка блока, заданные параметры блока, заявленный тип, а также тип данных параметров блока. Это значит, что Вы должны сначала запрограммировать вызываемые функции и функциональные блоки, то есть Вы должны начать программирование программы с блоков "самого нижнего уровня" (имеется в виду положение блоков относительно начала программы в исходном файле).

Однако, достаточно запрограммировать заголовки блоков и задать их параметры (то есть задать описание интерфейса ["interface description"]), чтобы не было ошибки при вызове блока. В дальнейшем Вы можете снабдить этот "интерфейс" программой, Однако необходимо сохранять без изменения интерфейс уже вызванного блока! Иначе редактор выдаст сообщение о конфликте временных меток при вызове блока.

В случае большой пользовательской программы Вы будете, очевидно, разбивать исходный файл программы на отдельные легкоуправляемые файлы, например, на "стандартные подпрограммы", которые можно использовать неоднократно по всей программе, технологически- или функционально-законченные подпрограммы и главную программу (main program), которая содержит, как правило, организационные блоки. При создании отдельных исходных файлов, Вы должны следить за порядком компилирования, для соблюдения правил вызовов блоков, приведенных выше.

Рекомендуется использовать следующий порядок компилирования:

- Пользовательские типы данных UDT
- Блоки глобальных данных
- Функции и функциональные блоки, начиная с блоков самого низкого уровня вызовов
- Экземплярные блоки данных (эти блоки могут также быть расположены непосредственно за назначенным функциональным блоком)
- Организационные блоки

Пример функционального блока с экземплярным блоком данных

На рисунке 3.10 показан пример функционального блока со статическими локальными данными, за которым следует экземплярный блок данных, связанный с этим функциональным блоком.

```
FUNCTION_BLOCK V_Memory
TITLE = 'Intermediate buffer for 4 values'
//В заголовке: промежуточный буфер на 4 значения
//Пример блока FB с параметрами и статическими локальными данными на SCL
AUTHOR   : Berger
FAMILY   : SCL_Book
NAME     : Memory
VERSION  : 01.00

VAR_INPUT
 Transfer    : BOOL := FALSE;//Пересылка положительного фронта сигнала
 Input_value : REAL := 0.0;  //в формате данных REAL  (дробное число)
END_VAR

VAR_OUTPUT
 Output_value : REAL := 0.0; //в формате данных REAL  (дробное число)
END_VAR

VAR
 Value1 : REAL := 0.0; //первое сохраненное значение в формате данных REAL
 Value2 : REAL := 0.0; //второе значение
 Value3 : REAL := 0.0; //третье значение
 Value4 : REAL := 0.0; //четвертое значение
 Edge_memory_bit : BOOL := FALSE;//меркер фронта передаваемого сигнала
END_VAR

BEGIN
//Передача и вывод имеют место, если в Transfer положительный фронт

IF Transfer = 1 AND Edge_memory_bit = 0
THEN Output_value  := Value4;
     //Передача начинается с последнего значения
     Value4  := Value3;
     Value3  := Value2;
     Value2  := Value1;
     Value1  := Input_value;
     Edge_memory_bit  := Transfer;   //обновление меркера, даже
ELSE Edge_memory_bit  := Transfer;   //если нет фронта
END_IF;

END FUNCTION BLOCK


DATA_BLOCK Values1
TITLE = 'Instance data block for  "V_Memory"
//Пример экземплярного блока данных для FB "V_Memory"

AUTHOR    : Berger
FAMILY    : SCL_Book
NAME      : V_MEM_DB1
VERSION   : 01.00
  V_Memory          //экземпляр для FB "V_Memory"
BEGIN
  Value1  := 1.0;   //Индивидуальные присвоения для
  Value2  := 1.3;   //отдельных значений
END_DATA_BLOCK
```

Рис. 3.10 Пример программирования функционального SCL-блока со связанным экземплярным блоком данных

3.6 Программирование блоков данных

В разделе 2.5 "Создание S7-программ" Вы можете найти введение в основы создания S7-программы и использование программного редактора.

Блоки данных программируются на языках программирования STL и SCL таким же образом, что и кодовые блоки. Для инкрементного программирования Вы будете использовать редактор STL-программ. Для программирования, ориентированного на создание исходных текстовых файлов программ используются как редактор STL-программ, так и редактор SCL-программ.

3.6.1 Инкрементное программирование блоков данных

Создание блоков

Процесс программирования блока начинается с его открытия одним из двух способов: либо двойным щелчком на блоке в окне проекта SIMATIC Manager, либо в редакторе STL-программ с помощью выбора опций меню: *File -> Open (Файл -> Открыть).* Если блок пока не существует, Вы должны его сначала создать одним из следующих путей:

- В левой половине окна проекта SIMATIC Manager выберите объект *Blocks (Блоки),* создайте новый блок данных с помощью опций меню: *Insert -> S7 Block -> Data Block (Вставка -> S7 Block -> Блок данных)*. В окне свойств (Properties) блока на вкладке "General - Part 1" ("Общие - часть 1") выберите номер блока. Язык программирования (creation language) установлен как "DB". Вы можете ввести остальные свойства блока позже.
- Находясь в редакторе, с помощью опций меню: *File -> New (Файл -> Создать)* вызовите окно диалога, в котором Вы можете задать требуемый блок в окне "Object name" ("Имя объекта"). После закрытия диалогового окна Вы можете вводить программу этого блока.

Вы можете ввести информацию заголовка блока либо при создании блока, либо позднее, активизировав редактор, затем открыв блок и выбрав опции меню: *File -> Properties (Файл -> Свойства)*.

Типы блоков данных

Когда Вы впервые открываете новый блок данных, Вы увидите окно "New Data Block" ("Новый блок данных"); теперь Вы должны решить, какой тип назначать для создаваемого блока.

Вы можете назначить блоку данных один из трех следующих типов, щелкнув кнопкой манипулятора "мышь" по одному из них:

- "Data Block" ("Блок данных")
  При выборе данной опции создается блок глобальных данных; в данном случае Вы должны объявить назначенные адреса данных при программировании блока.
- "Data block with assigned user-defined data type" ("Блок данных пользовательского типа")
  При выборе данной опции создается блок данных пользовательского типа; в данном случае Вы должны объявить структуру данных пользовательского типа UDT.

- "Data block with assigned function block" ("Блок данных с назначением функциональному блоку")
  При выборе данной опции создается экземплярный блок данных; в данном случае Вы должны объявить структуру данных для пересылки в соответствующий функциональный блок.

Окно блока

На нижеприведенном рисунке представлен пример открытого блока данных (см. рис. 3.11).

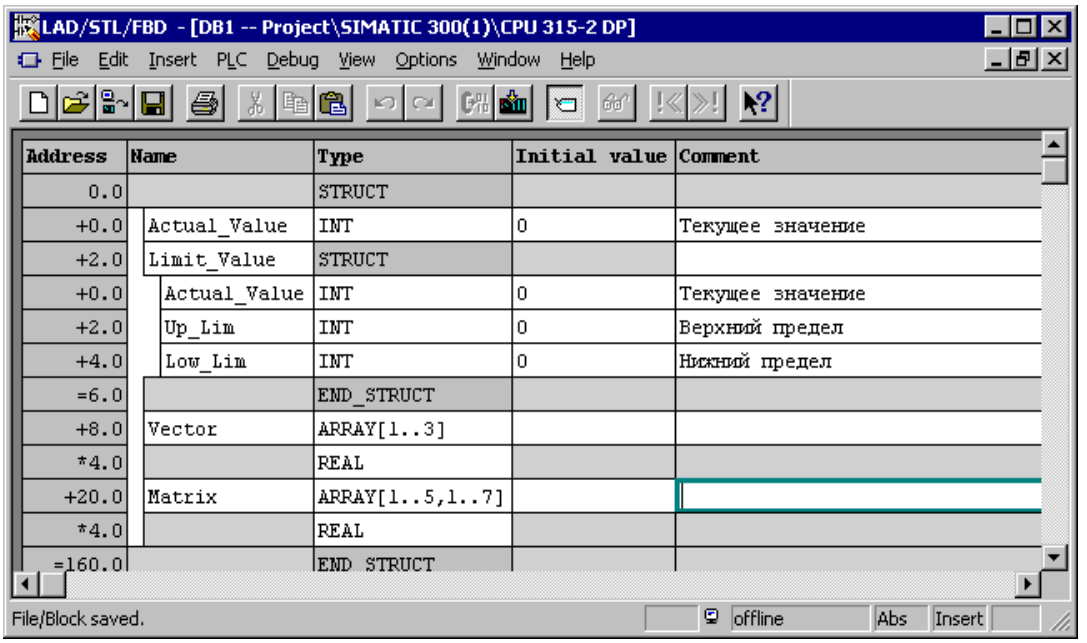


Рис.3.11 Пример открытого блока данных (Declaration View [вид объявления данных])

Вы можете выбирать один из двух видов окна:

- Declaration View (вид объявления данных)
  При выборе данной опции Вы вводите назначенные адреса данных, задаете тип данных и определяете начальные значения.
- Data View (вид фактических значений данных)
  При выборе данной опции редактор отображает фактические значения данных, которые Вы можете редактировать.

При программировании блока глобальных данных Вы можете задать для каждого адреса начальное значение. Переменные имеют стандартное значение, принимаемое по умолчанию, равное нулю (0), наименьшему значению или пробелу (blank), в зависимости от типа данных.

Экземплярный блок данных, генерирующийся для функционального блока, принимает в качестве начальных значений данных значения, принятые по умолчанию, из раздела объявления переменных этого FB.

Блок, создаваемый из данных пользовательского типа UDT, принимает в качестве начальных значений данных значения, принятые по умолчанию, в UDT.

Редактор может отображать блок данных в двух видах:

Declaration View (вид объявления данных: *View -> Declaration View [Вид -> Вид объявления данных]*)
При использовании данного вида Вы можете определять адреса данных, кроме того Вы можете наблюдать переменные в том виде, который Вы определили для них, например, поле или пользовательский тип данных как одну переменную.

Data View (вид фактических значений данных: *View -> Data View [Вид -> Вид фактических значений данных]*)
При использовании данного вида редактор отображает каждую переменную и каждый компонент поля или структуры индивидуально. Данный вид обеспечивает дополнительно отображение столбца "Actual value" ("Фактическое значение"). Фактическое значение адреса данных - это то значение, которое это адрес имеет или будет иметь в основной (main) памяти CPU. Редактор использует здесь начальное значение в качестве значения по умолчанию.

Вы можете модифицировать фактические значения отдельно для каждого адреса данных.

Пример: допустим Вы создаете несколько экземплярных блоков данных для функционального блока. Тем не менее, Вам необходимо для каждого вызова функционального блока (для каждой пары FB/DB) иметь слегка отличающийся в предустановках отдельный экземплярный блок данных. Вы можете редактировать каждый блок данных, выбрав опции: *View -> Data View (Вид -> Вид фактических значений данных)* и затем задавая значения, действительные только для данного блока данных, в столбце "Actual value" ("Фактическое значение"). С помощью опций меню: *Edit -> Initialize Data Block (Правка -> Инициализировать блок данных)* Вы можете вновь вернуть начальные значения данным этого блока.

3.6.2 Программирование блоков данных, ориентированное на создание исходных файлов

При создании исходных файлов блоков данных Вы должны придерживаться структуры или порядка, показанного в таблице 3.5. Данная таблица регламентирует использование ключевых слов при программировании блоков данных. Это касается как исходных файлов STL-программ, так и исходных файлов SCL-программ.

Заголовок блока

Вы должны запрограммировать свойства блока в его заголовке после указания типа блока и перед разделом объявления переменных. Вся информация в заголовке блока опциональна (optional); Вы можете пропустить или отдельные характеристики, или все характеристики без исключения. Для получения информации по описанию и назначению свойств блоков обратитесь к разделу 3.2.3 "Свойства блоков".

С помощью ключевого слова "TITLE =" сразу же после строки с указанием типа блока Вы можете ввести заголовок блока длиной до 64 символов. Вы можете вслед за тем добавить комментарий в одной или нескольких строках, начинающихся двойной косой чертой. Комментарий блока может иметь размер до 18 Кбайт.

Таблица 3.5 Ключевые слова для программирования блоков данных

Блок	Блок глобальных данных [Global Data Block]	Блок глобальных данных из UDT [Global Data Block from UDT]	Экземплярный блок данных [Instance Data Block]
Тип блока	DATA_BLOCK	DATA_BLOCK	DATA_BLOCK
Заголовок [Header]	TITLE = *Заголовок блока* *//Комментарий блока* KNOW_HOW_PROTECT NAME : *Имя блока* FAMILY : *Второе имя* AUTHOR : *Автор* VERSION : *Версия* READ_ONLY UNLINKED	TITLE = *Заголовок блока* *//Комментарий блока* KNOW_HOW_PROTECT NAME : *Имя блока* FAMILY : *Второе имя* AUTHOR : *Автор* VERSION : *Версия* READ_ONLY UNLINKED	TITLE = *Заголовок блока* *//Комментарий блока* KNOW_HOW_PROTECT NAME : *Имя блока* FAMILY : *Второе имя* AUTHOR : *Автор* VERSION : *Версия*
Описание [Declaration]	STRUCT *Name : Type := Default* *(Имя : Тип := Значение по умолчанию)* END_STRUCT	*UDTname (имя UDT)*	*FBname (имя FB)*
Инициа-лизация [Initialization]	BEGIN *Name := Default* *(Имя := Значение по умолчанию)* *… и т.д.*	BEGIN KOMPname := Default (имя := Значение по умолчанию) *… и т.д.*	BEGIN KOMPname := Default (имя := Значение по умолчанию) *… и т.д.*
Конец блока [Block end]	END_DATA_BLOCK	END_DATA_BLOCK	END_DATA_BLOCK

Описание переменных

Раздел объявления переменных содержит определения всех внутриблочных переменных, т.е. таких переменных, которые Вы применяете только в этом блоке. Вы можете объявить блок данных как блок глобальных данных с отдельными ("individual") переменными, как блок глобальных данных с UDT и как экземплярный блок данных.

Описание переменной в блоке глобальных данных состоит из имени, типа данных, значения по умолчанию (если есть) и комментария переменной (необязательный элемент).

Например:

Quantity : INT := +500; //Units per batch (единиц на пакет)

Всем переменным могут быть назначены значения по умолчанию. Порядок следования переменных в разделе описания переменных произволен, при этом порядок следования типов переменных влияет (с учетом типов данных) на расходование памяти. В главе 24 "Типы данных" представлены требования к памяти для разных типов переменных. Из раздела 26.2 "Хранение данных в переменных" Вы можете получить информацию о том, как переменные хранятся в блоках данных. Вы можете оптимизировать распределение памяти, правильно планируя порядок размещения данных.

Если Вы не назначите других значений по умолчанию для переменных, редактор запишет в них нулевые значения (0), минимальные значения или заполнит их пробелами (blank), в зависимости от типа данных.

Блок, создаваемый из данных пользовательского типа UDT, состоит только из UDT. Вы можете использовать абсолютные адреса (например, UDT 51) или символьные адреса (например, "Frame header").

Экземплярный блок данных, генерирующийся для функционального блока, принимает в качестве начальных значений данных только значения из раздела объявления переменных этого FB или с абсолютной, или с символьной адресацией.

Инициализация блока данных

Раздел инициализации блока данных начинается с ключевого слова BEGIN и заканчивается ключевым словом END_DATA_BLOCK. Даже если Вы не назначаете значения по умолчанию для переменных в разделе инициализации, Вы должны ввести эти ключевые слова.

Значения, которые Вы определили в разделе инициализации блока данных соответствуют фактическим (actual) значениям при инкрементном программировании. При компилировании блока данных значения, принятые по умолчанию (default) для переменных в разделе инициализации блока, становятся начальными (initial) значениями этих переменных, а начальные значения становятся фактическими (actual) значениями. Если блок данных загружается в CPU, начальные (initial) значения пересылаются в загрузочную (load) память, а фактические (actual) значения пересылаются в рабочую (work) память CPU (см. раздел 2.6.5 "Обработка блоков" пункт "Блоки данных в автономном (offline)/в интерактивном (online) режимах").

Если Вы не определили начальные значения для переменных, редактор будет использовать начальные значения как фактические. Если Вы используете пользовательские типы данных со значениями, принимаемыми по умолчанию, в разделе объявления переменных, Вы можете переписать (заменить - "overwrite") эти значения, принимаемые по умолчанию, в разделе инициализации.

Это же касается экземплярных блоков данных, которые назначены функциональным блокам (со своими значениями, принимаемыми по умолчанию) как структуры данных. Здесь Вы можете сформировать фактические (actual) значения по отдельности для каждого экземпляра (для вызова функционального блока с конкретным блоком данных).

3.7 Переменные и константы

3.7.1 Общие замечания по поводу переменных

Переменная - это величина особого формата (см. рис. 3.12). Простые переменные состоят из адреса (например, адрес входа input 5.2) и типа данных (например, BOOL для дискретного [двоичного] значения). Адрес, в свою очередь, содержит идентификатор адреса (такой как I для входа input) и указание абсолютного месторасположения (такое, например, как 5.2, что означает 2-ой бит 5-ого байта). Вы можете также использовать для доступа к адресу или переменной символьное имя, назначив для этого адреса имя (символ) в таблице символов (Symbol Table).

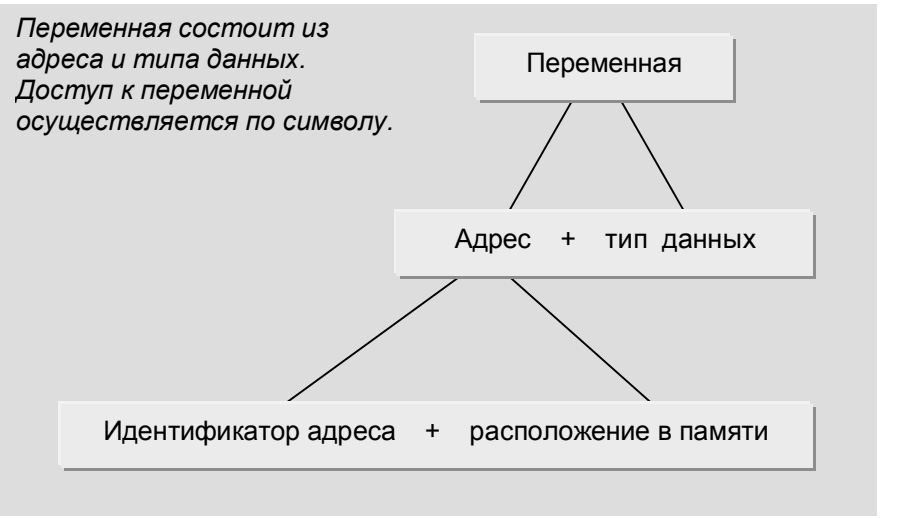


Рис. 3.12 Структура переменной

Бит данных типа BOOL адресуется как *двоичный адрес* (*binary address*) или *двоичный операнд* (*binary operand*). Адреса, содержащие один, два или четыре бита или переменные соответствующих типов называются *численными операндами (digital operand)*.

Переменные, объявленные внутри блока, являются внутриблочными (local - локальными) переменными. К ним относятся параметры блока, статические и временные локальные данные и даже адреса данных в блоках глобальных данных. Если локальные переменные являются переменными простого типа, они также могут быть доступны как операнды (например, статические локальные данные - как DI операнды, временные локальные данные - как L операнды, и данные в блоках глобальных данных - как DB операнды).

Тем не менее, локальные переменные могут быть сложных типов (например, такие переменные как структура или массив). Переменные этих типов требуют для размещения памяти больше, чем 32 бита, так что они не могут быть, например, загружены в аккумулятор. И по этой же причине они не могут быть адресованы с помощью обычного ("normal") STL-выражения. Существуют специальные функции для обработки таких переменных, такие как IEC-функции, которые поставляются в виде стандартной библиотеки с ПО STEP 7 (Вы можете создавать переменные сложных типов в параметрах блоков такого же типа).

Если переменные сложного типа содержат компоненты простого типа, то эти компоненты могут расматриваться, как если бы они были отдельными переменными (например, Вы можете загрузить в аккумулятор элемент массива, состоящего из 30 значений целого (INT) типа, и в дальнейшем обрабатывать его).

Константы используются для задания переменным фиксированных значений. Константа имеет особый префикс в зависимости от типа данных, к которому она принадлежит.

3.7.2 Общие замечания по поводу типов данных

Тип данных обуславливает характеристики данных, особенно это касается представления содержания переменной и диапазона допустимых для нее значений. STEP 7 предусматривает предопределение типов данных. Вы можете комбинировать типы данных, формируя пользовательске типы данных (User Data Type - UDT). Типы данных доступны в базовом пакете (Global Basis) и могут использоваться в каждом блоке.

В данном разделе приводится обзор всех типов данных. В разделе также содержится краткое введение в простые типы данных. Эти знания помогут Вам программировать PLC.

Ниже в таблице 3.6 приведен обзор типов данных в STEP 7.

Таблица 3.6 Классификация типов данных

Elementary Data Types (Простые типы данных)	Complex Data Types (Сложные типы данных)	User Data Types (Пользовательские типы данных)	Parametr Data Types (Типы данных для параметров)
BOOL, BYTE, CHAR, WORD, INT, DATE, DWORD, DINT, REAL, S5TIME, TIME, TOD	DT, STRING, ARRAY, STRUCT	UDT, блоки глобальных данных, экземпляры (Instances)	TIMER, COUNTER, BLOCK_DB, BLOCK_SDB, BLOCK_FC, BLOCK_FB, POINTER, ANY
Типы данных с объемом не более одного двойного слова (32 бита)	Типы, которые могут содержать данные объемом более одного двойного слова (DT, STRING) или состоящие из нескольких компонентов	Структуры или области данных, которые могут быть адресованы по имени	Параметры блока
Могут распределяться для операндов абсолютной или символьной адресацией	Могут распределяться только для переменных с символьной адресацией		Могут распределяться только для параметров блоков (только с символьной адресацией)
Разрешенные во всех адресных областях	Разрешенные в блоках данных (как глобальные данные и экземплярные данные), как временные локальные данные и как параметры блоков		Разрешенные в связи с параметрами блоков

3.7.3 Простые типы данных

Переменные простых типов данных могут быть отредактированы непосредственно в редакторе STL-программ, так как их значения могут быть занимать от одного до 32 битов в памяти. При присвоении значений в программе переменные простых типов данных также могут быть отредактированы непосредственно в SCL-редакторе.

Переменные простых типов могут быть предопределены фиксированными значениями (константами) на этапе объявления переменных.

Здесь записи в STL-программе (см. табл. 3.7) и записи в SCL-программе (см. табл. 3.8) отличаются друг от друга. Для многих типов данных существует больше одной формы для записи константы и все эти записи одинаково правомочны в употреблении (например, можно использовать форму записи TIME# или форму записи T#).

Запись константы в STL

Язык STL не накладывает ограничений на обработку (на операторы) типов данных (за исключением различий для двоичных [binary] и численных [digital] операндов). Функции сравнения, такие например, как сравнение содержимого аккумуляторов, не зависят от типа переменных, содержащихся в аккумуляторах.

Запись константы в SCL

При использовании языка SCL Вы можете обрабатывать переменные только разрешенных типов данных. Константы в SCL не принимают своего типа данных, пока они не будут поставлены в соответствие оператору.

Пример: в SCL константа 12345 относится к классу типов ANY_NUM, так как в зависимости от применения она будет иметь тип INT или DINT, или REAL. С помощью "определяющих тип" ("type-defined") записей Вы можете назначить отдельный тип данных непосредственно константе, например, с помощью определения DINT#12345 вы задаете для константы тип DINT.

3.7.4 Сложные типы данных

Вы можете использовать сложные типы данных (табл. 3.9) для работы с переменными в блоках данных или в L-стеке, а также для работы с параметрами блока.

Переменные сложных ("complex") типов, которые назначаются параметрам блоков, могут быть только полными ("complete") переменными; отдельные части переменных сложных типов не могут быть обработаны с помощью обычных ("normal") операторов. Тем не менее с помощью прямого доступа к переменным ("direct variable access") и с помощью косвенной адресации STL обеспечивает способ управления переменными, если известна их внутренняя структура.

Кроме того существуют IEC-функции, с помощью которых можно обрабатывать переменные DT и STRING (например, объединение двух символьных строк в одну). IEC-функции являются составной частью системы STEP 7; Вы можете найти их в стандартной библиотеке "Standard Library" в разделе "IEC Function Blocks". IEC-функции могут быть использованы в любом языке программирования. Длина переменной DT является фиксированной; Вы можете сами задавать длину переменных STRING, ARRAY и STRUCT при их определении.

Строка символов STRING может содержать до 254 символов и резервирует в памяти на 2 байта больше, чем число символов в строке.

Массив может иметь до 65536 элементов для каждого из индексов (т.е. от -32768 до 32767).

Таблица 3.7 Обзор простых типов данных при записи констант на языке STL

Data Type (width) (Тип данных, размер)	Description (Описание)	Пример записи STL-констант	
		минимальное значение	максимальное значение
BOOL (1 бит)	Bit (двоичное число)	FALSE (ЛОЖЬ)	TRUE (ИСТИНА)
BYTE (8 битов)	8-bit hexadecimal number (8-разрядное шестнадцатеричное число)	B#16#00, 16#00	B#16#FF, 16#FF
CHAR (8 битов)	One character (ASCII) (Один символ ASCII)	Печатный символ, например, 'A'	Печатный символ, например, 'A'
WORD (16 битов)	16-bit hexadecimal number (16-разрядное шестнадцатеричное число)	W#16#0000, 16#0000	W#16#FFFF, 16#FFFF
	16-bit binary number (16-разрядное двоичное число)	2#0000_0000_0000_0000	2#1111_1111_1111_1111
	Count value, 3 decades BCD (Значение счетчика, 3 декады формата BCD)	C#000	C#999
	Two 8-bit unsigned decimal numbers (Два 8-разрядных десятичных числа без знака)	B(0,0)	B(255,255)
DWORD (32 бита)	32-bit hexadecimal number (32-разрядное шестнадцатеричное число)	DW#16#0000_0000, 16#0000_0000	DW#16#FFFF_FFFF, 16#FFFF_FFFF
	32-bit binary number (32-разрядное двоичное число)	2#0000_0000...0000_0000	2#1111_1111...1111_1111
	Four 8-bit unsigned decimal numbers (Четыре 8-разрядных десятичных числа без знака)	B(0,0,0,0)	B(255,255,255,255)
INT (16 битов)	Fixed-point number (Число с фиксированной запятой)	-32 768	+32 767
DINT (32 бита)	Fixed-point number (Число с фиксированной запятой)	L# -2 147 483 648 [1]	L#+2 147 483 647 [1]
REAL (32 бита)	Floating-point number (Число с плавающей запятой)	в экспоненциальном представлении: +1.234567E+02 [2]	
		как десятичное число: 123.4567 [2]	
S5TIME (16 битов)	Time value in SIMATIC format (Значение времени в SIMATIC - формате)	S5T#0ms, S5TIME#0ms	S5T#2h46m30s, S5TIME#2h46m30s
TIME (32 бита)	Time value in IEC format (Значение времени в IEC-формате)	T#-24d20h31m23s647ms, TIME#-24d20h31m23s647ms	T#24d20h31m23s647ms, TIME#24d20h31m23s647ms
		T#-24.855134d, TIME#-24.855134d	T#24.855134d, TIME#24.855134d
DATE (16 битов)	Date (Дата)	D#1990-01-01, DATE# 1990-01-01	D#2168-12-31, DATE#2168-12-31
TIME_OF_DAY (32 бита)	Time of day (Суточное время)	TOD#00:00:00, TIME_OF_DAY#00:00:00	TOD#23:59:59.999, TIME_OF_DAY#23:59:59.999

[1] "L#" может быть опущено, если число за пределами диапазона целых чисел INT

[2] информация о диапазоне значений в разделе 24.1.3, "Представление чисел"

Таблица 3.8 Обзор простых типов данных при записи констант на языке SCL

Data Type (width) (Тип данных, размер)	Description (Описание)	Пример записи SCL-констант
BOOL (1 бит)	Bit (бит, двоичное число)	FALSE (ЛОЖЬ), TRUE (ИСТИНА), BOOL#FALSE, BOOL'TRUE. 2#0. 2#1. BOOL#0. BOOL#1
BYTE (8 битов)	8-bit decimal number 8-bit hexadecimal number 8-bit octal number 8-bit binary number Соответственно, 8-разрядные: десятичное, шестнадцатеричное, восьмеричное и двоичное числа	0, B#127, BYTE#255 16#0, B#16#7F, BYTE#16#FF 8#0, B#8#177, BYTE#8#377 2#0, B#2#0111_1111, BYTE#2#0111_1111
CHAR (8 битов)	One printable character (ASCII) (Печатный символ ASCII)	' ', CHAR#' ', CHAR#20 'z', CHAR#'z', CHAR#122
WORD (16 битов)	16-bit decimal number 16-bit hexadecimal number 16-bit octal number 16-bit binary number Соответственно, 16-разрядные: десятичное, шестнадцатеричное, восьмеричное и двоичное числа	0, W#32767, WORD#65535 16#0, W# 16#7FFF, WORD#16#FFFF 8#0, W#8#7J7777, WORD#8#17_7777 2#0, W#2#0111_1111_ …, WORD#2#1111_1111_…
DWORD (32 бита)	32-bit decimal number 32-bit hexadecimal number 32-bit octal number 32-bit binary number Соответственно, 32-разрядные: десятичное, шестнадцатеричное, восьмеричное и двоичное числа	0, DW#2147483647, DWORD#4294967295 16#0, DW#16#7FFF_FFFF, DWORD#16#FFFF_FFFF 8#0, DW#8#177_7777_7777, DWORD#8#377_7777_ ... 2#0, DW#2#0111_1111_... , DWORD#2#1111_1111_...
INT (16 битов)	16-bit decimal number 16-bit hexadecimal number 16-bit octal number 16-bit binary number Соответственно, 16-разрядные: десятичное, шестнадцатеричное, восьмеричное и двоичное числа	-32_768, 0, +32_767 INT#16#0, INT#16#7FFF, INT#16#FFFF INT#8#0, INT#8#7_7777, INT#8#17_7777 INT#2#0, INT#2#0111_1111_ ..., INT#2#1111_1111_...
DINT (32 бита)	32-bit decimal number 32-bit hexadecimal number 32-bit octal number 32-bit binary number Соответственно, 32-разрядные: десятичное, шестнадцатеричное, восьмеричное и двоичное числа	-2_147_483_648, 0, +2_147_483_647 DINT#16#0, DINT#16#7FFF_FFFF, DINT#16#FFFF_... DINT#8#0, DINT#8#177_7777_7777, DINT#8#377_... DINT#2#0, DINT#2#0111_1111_... , DINT#2#1111_...
REAL (32 бита)	Floating-point number (Число с плавающей запятой)	в экспоненциальном представлении: +1.234567E+02 [1)] как десятичное число: -123.4567 [1)] как целое число: +1234567 [1)]
S5TIME (16 битов)	Time value for SIMATIC timer functions (Значение времени в SIMATIC -формате)	T#0ms, TIME#2h46m30s T#0.0s, TIME#24.855134d
TIME (32 бита)	Time value in IEC format (Значение времени в IEC-формате)	T#-24d20h31m23s647ms, T#0ms, TIME#24d20h31m23s647ms T#-24.855134d, T#0.0ms, TIME#24.855134d
DATE (16 битов)	Date (Дата)	D# 1990-01-01, D#2168-12-31 DATE# 1990-01-01, DATE#2168-12-31
TIME_OF_DAY (32 бита)	Time of day (Суточное время)	TOD#00:00:00, TOD#23:59:59:999 TIME_OF_DAY#00:00:00, TIME_OF_DAY#23:59:59:999

[1)] информация о диапазоне значений в разделе 24.1.3, "Представление чисел"

Таблица 3.9 Обзор сложных типов данных

Тип данных	Описание		Пример
DATE_AND_TIME	Дата и время	64 бита	DT#1990-01-01-00:00:00.000 DATE_AND_TIME#2168-12-31:23:59:59.999
STRING	Строка символов	Переменная	Набор ASCII символов, например, "String 1"
ARRAY	Массив	Переменная	Набор компонентов одного типа данных; возможно до 6 размерностей
STRUCT	Структура	Переменная	Набор компонентов разного типа данных; возможно до 6 уровней вложения

3.7.5 Параметрические типы

Параметрические типы - это типы данных для параметров блоков (см. табл. 3.10). Размеры данных в таблице соответствуют области памяти, требуемой для параметров блока (для случая функциональных блоков). Используйте также TIMER и COUNTER в таблице символов как типы данных для таймеров и счетчиков.

Таблица 3.10 Обзор параметрических типов

Тип параметра	Описание		Пример фактических адресов
TIMER	Таймер	16 битов	T 15 или символ
COUNTER	Счетчик	16 битов	C 16 или символ
BLOCK_FC	Функция	16 битов	FC 17 или символ
BLOCK_FB	Функциональный блок	16 битов	FB 18 или символ
BLOCK_DB	Блок данных	16 битов	DB 19 или символ
BLOCK_SDB	Системный блок данных	16 битов	(до сих пор не используется)
POINTER	DB указатель	48 битов	Как указатель: P#M10.0 или P#DB20.DBX22.2 Как адрес: MW 20 или I 1.0 или символ
ANY	ANY указатель	80 битов	Как область: P#DB10.DBX0.0 WORD 20 или любая целая переменная

Базовые функции

В данной части книги описываются функции языка программирования STL, представляющие собой "базовые функции" ("basic functionality") STEP 7. Эти функции позволят Вам запрограммировать PLC как систему управления, построенную на контакторах и реле.

Двоичные логические операции (**binary logic operations**) используются для моделирования параллельных и последовательных цепей в схемах или для выполнения функций AND (И) и OR (ИЛИ) в электронных переключающих системах. Использование приема вложения ("комбинирование") функций делает возможным выполнение сложных двоичных логических операций.

Операции с памятью (**memory functions**) используются для сохранения результата логической операции (RLO - result of logic operation). Таким образом, этот результат может быть в дальнейшем проверен и обработан в другом месте программы.

Функции пересылки (**transfer functions**) используются для обработки дискретных значений. Эти функции также используются, например, для передачи таймеру значения времени.

Таймеры (**timer**) используются для программирования PLC на выполнение функций включения/выключения, управляемых по времени, и для обеспечения функции задержки в переключающих электронных системах "electronic switching systems". Таймеры, встроенные в CPU, позволяют Вам программировать такие параметры как время ожидания или время контроля (monitoring time).

Счетчики (**counter**) двух видов - прямого и обратного счета - используются для счета в пределах от 0 до 999.

В данной части книги описываются функции, использующие адресные области входов, выходов и меркеров. Входы и выходы связаны с процессом или установкой. Меркеры играют роль дополнительных реле, сохраняющих двоичные состояния. В последующих разделах книги рассматриваются другие адресные области, которые могут использоваться в двоичных логических операциях. Одними из наиболее важных являются биты данных в блоках глобальных данных, а также биты данных во временных и статических локальных данных.

Глава 5 "Операции с памятью" (memory functions) содержит примеры программирования двоичных логических операций и операций с памятью (memory functions); глава 8 "Функции счетчиков" предоставляет Вашему вниманию примеры использования таймеров и счетчиков. Каждый пример содержится в FC функции без параметров.

4 Двоичные логические операции (binary logic operations)

Функции AND (И), OR (ИЛИ), Exclusive OR (Исключающее ИЛИ); считывание состояния сигнала "1" и "0"; выполнение двоичных логических операций; вложение (комбинирование) функций.

5 Операции с памятью (memory functions)

Функции Assign (присвоение), Set (установка) и Reset (сброс); RS-триггер; проверка наличия фронта сигнала; пример управления ленточным транспортером.

6 Функции пересылки (transfer functions)

Функции Load (загрузка) и Transfer (выгрузка); функции аккумулятора.

7 Таймеры (timer)

Запуск 5 различных типов таймеров; сброс, разблокировка и считывание значения таймера; значение времени.

8 Счетчики (counter)

Установка счетчика; прямой и обратный счет; сброс разблокировка и считывание значения счетчика; значение счетчика; пример счетчика числа деталей.

4 Двоичные логические операции

В этой главе обсуждаются функции AND (И), OR (ИЛИ), Exclusive OR (Исключающее ИЛИ), также как и их комбинации для языка программирования STL. Функции AND (И), OR (ИЛИ), Exclusive OR (Исключающее ИЛИ) используются для проверки состояния сигналов двоичных разрядов и связи их друг с другом.

Двоичный разряд может быть проверен (просканирован) на наличие сигнала "1" или "0". С помощью инвертирования результата логической операции и использования комбинации логических операций Вы можете запрограммировать сложные логические операции без необходимости хранения промежуточного результата.

Примеры, показанные в данном разделе, представлены на дискете, приложенной к данной книге, в библиотеке "STL_Book" в разделе "Basic Functions" в функциональном блоке FB 104 и в исходном файле Chap_4.

4.1 Структура программируемого контроллера

На рис. 4.1 показана общая схема выполнения двоичной логической операции. Входной модуль выбирает датчик посредством адреса, например, датчик на входе I 1.2. CPU проверяет состояние сигнала ("status" - статус) датчика связывает его на входе блока логической операции с результатом логической операции, сохраненным после выполнения логической операции в предыдущий раз. Результат текущей логической операции (RLO) запоминается и хранится как "новый результат логической операции". Затем CPU обрабатывает следующее выражение программы, например, обеспечивающее сохранение результата логической операции в специальной ячейке памяти.

Состояние сигнала (статус)

Состояние (статус) бита эквивалентно состоянию (статусу) его сигнала и может иметь значения "0" или "1". В SIMATIC S7 состояние сигнала имеет значение "1" при наличии напряжения на входе, например, ~ 230 В или = 24 В (в зависимости от модуля); с другой стороны, если напряжение на входе отсутствует, то состояние сигнала входа соответствует "0".

Выражение, содержащее оператор проверки (check statement), инициирует проверку состояния бита. В то же время это выражение содержит правило логической операции - алгоритм, согласно которому результат проверки состояния бита будет сравниваться с сохраненным в процессоре результатом логической операции. Например выражение:

```
A   I 17.1
```

проверяет вход I 17.1 на состояние "1" и связывает с RLO по логике AND (И).

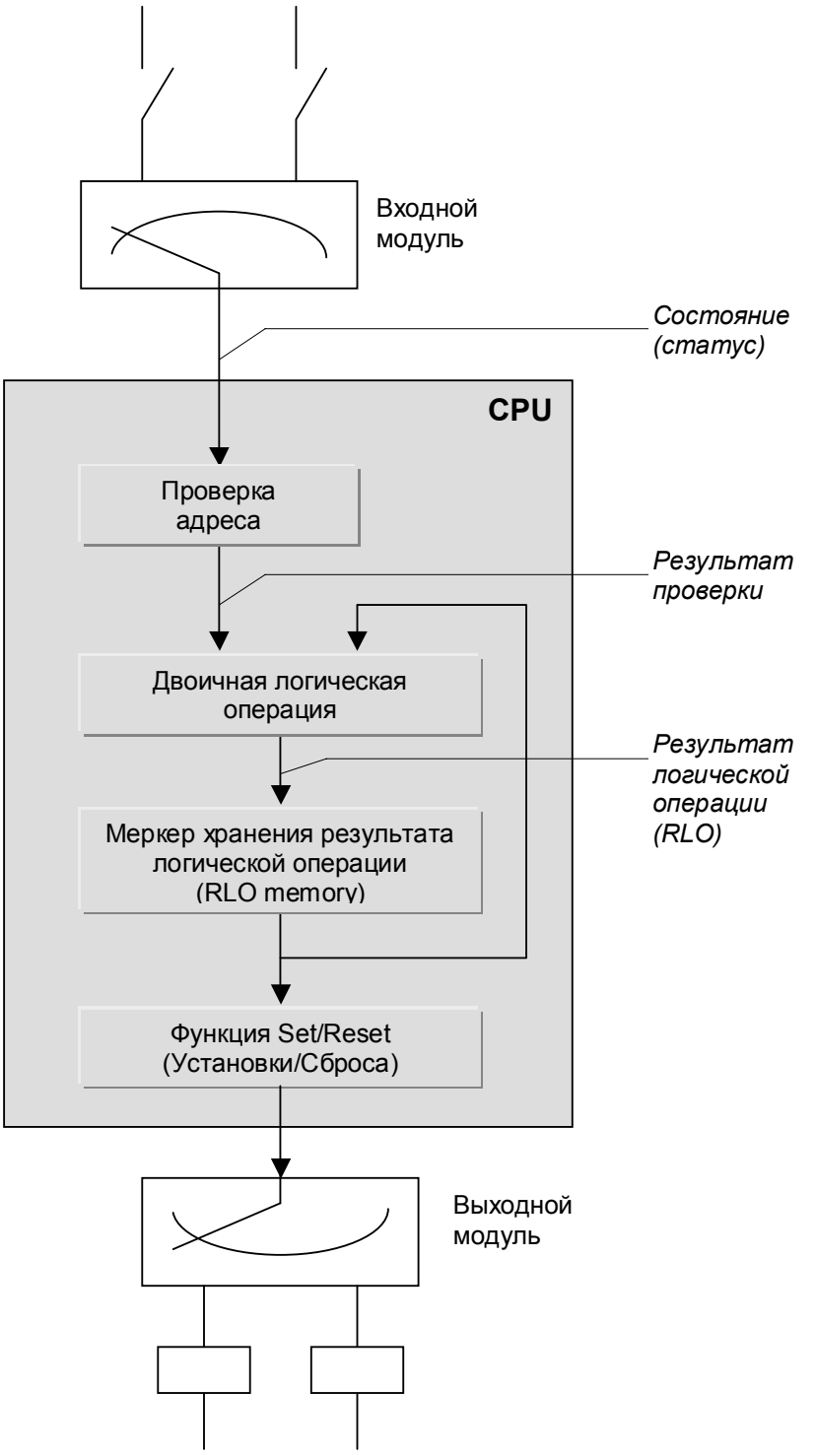


Рис. 4.1 Схема выполнения двоичной логической операции

Выражение:

ON M 20.5

проверяет меркер M 20.5 на состояние "0" и связывает (links) с RLO по правилу логической операции OR (ИЛИ).

Результат проверки

Строго говоря, CPU не связывает (link) состояние сигнала проверенного бита, а скорее формирует результат проверки: в случаях проверки на состояние сигнала "1" результат проверки идентичен состоянию сигнала проверяемого бита; в случаях проверки на состояние сигнала "0" результат проверки инвертируется относительно состояния сигнала проверяемого бита.

Результат логической операции

Результат логической операции (RLO) - это состояние сигнала в CPU, которое CPU в дальнейшем использует для обработки двоичных сигналов. Результат логической операции формируется и модифицируется с помощью операторов проверки (check statement). Если RLO имеет значение "1", то это означает, что условие двоичной логической операции выполнено, если RLO имеет значение "0", то это означает, что условие двоичной логической операции не выполнено. Биты устанавливаются или сбрасываются в соответствии с результатом логической операции.

Логический шаг (logic step)

Так же как можно выделить последовательный шаг в последовательной системе управления (sequential control system), так и в логической системе управления (logic control system) можно выделить логический шаг (logic step). На каждом таком логическом шаге формируется и оценивается (т.е. подвергается последующей обработке) результат логической операции. Логический шаг состоит из выражений с операторами проверки (также называемыми операторами сканирования - "scan statements") и выражений с условными операторами. Первый оператор проверки, следующий за условным оператором, называется "первичным опросом" ("first check").

В нижеследующей схеме логической системы управления выделен логический шаг:

```
…                …
=  Q 4.0        условный оператор (conditional statement)
A  I 2.0        первичный опрос (first check)
A  I 2.1        оператор проверки (check statement)
…               …
A  I 1.7        оператор проверки (check statement)
=  Q 5.1        условный оператор (conditional statement)
…               …
=  Q 4.3        условный оператор (conditional statement)
O  I 2.6        первая проверка (first check)
O  I 2.5        оператор проверки (check statement)
…               …
```

Первичный опрос (First check)

Первый оператор проверки, следующий за условным оператором, называется "первичным опросом" ("first check"). Он имеет особенное значение, потому что CPU имеет прямой доступ к результату этого оператора, как результату логической операции. Таким образом "старое значение" ("old") результата логической операции RLO теряется. Первичный опрос всегда соответствует началу логической операции. Правило (алгоритм) первичного опроса (AND (И), OR (ИЛИ), Exclusive OR (Исключающее ИЛИ)) не играет при этом никакой роли.

Оператор проверки (Check statement)

Результат логической операции RLO формируется с помощью операторов проверки. Эти операторы проверяют сигнал бита на состояние "1" или "0" и связывают (link) его в соответствии с правилом AND (И), OR (ИЛИ), Exclusive OR (Исключающее ИЛИ). Затем CPU сохраняет результат данной логической операции как новый результат логической операции RLO.

На рис. 4.2 показано, как программируется проверка сигнала бита на состояние "1" и "0". В случаях проверки на состояние сигнала "1" результат проверки идентичен состоянию сигнала проверяемого бита. В случаях проверки на состояние сигнала "0" результат проверки инвертируется относительно состояния сигнала проверяемого бита.

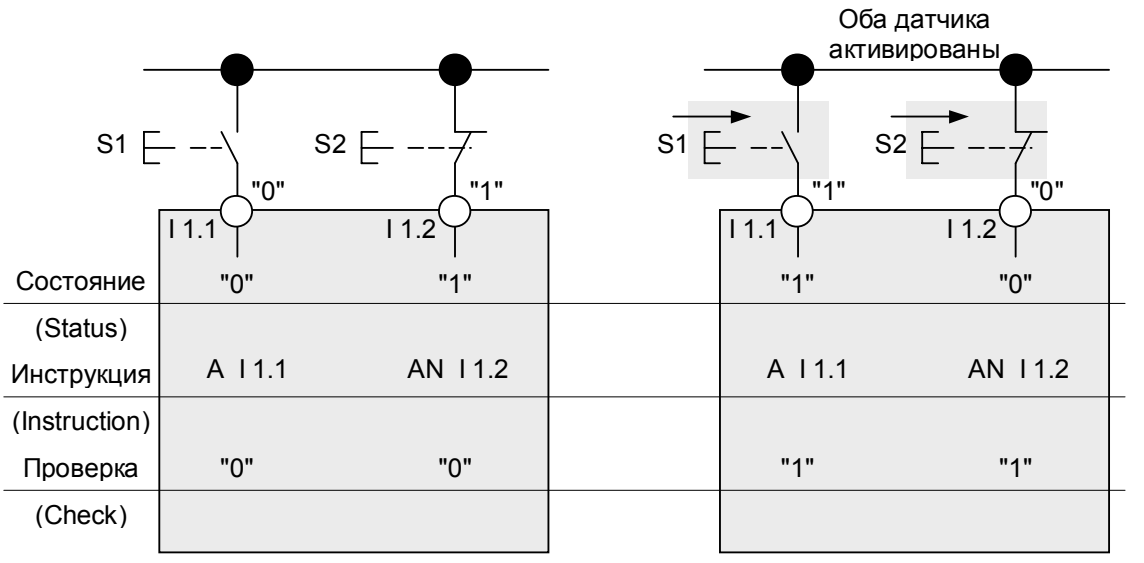


Рис. 4.2 Проверка состояния сигнала "1" и "0"

Условные операторы (Condition statement)

Условные операторы (condition statement) - это такие операторы, выполнение которых зависит от результата логической операции RLO. Эти операторы включают операции присвоения (назначения [assign]), установки (set) и сброса (reset) двоичных разрядов, а также запуск таймеров и счетчиков и т.п.

Условные операторы (за редким исключением) выполняются, когда результат логической операции RLO имеет состояние "1", и не выполняются, когда RLO имеет состояние "0". Условные операторы (за редким исключением) не влияют на результат логической операции RLO, и, таким образом, RLO на протяжении последовательного выполнения нескольких операторов остается неизменным.

Правильный подход в программировании

Правило (алгоритм) первичного опроса не имеет значения, так как результат этой проверки берется непосредственно как результат логической операции. В целях грамотного программирования правило (алгоритм) первичного опроса должно быть идентично требуемой функции. Например, последовательность операторов

```
…
=   Q 15.3
O   I 18.5      первая AND функция
A   I 21.7
=   Q 15.4

A   I 18.4      вторая AND функция
A   I 21.6
=   Q 15.5
…
```

представляет собой две AND-функции, из которых вторая функция (в которой обе проверки имеют логику AND) запрограммирована более правильным и более предпочтительным способом.

В случае отдельных операторов проверки, как например, в следующем фрагменте:

```
…
=   Q 10.0
A   I 20.1      присвоение
=   Q 10.1      I 20.1 -> Q 10.1
…
```

функция AND более предпочтительна.

4.2 Элементарные двоичные логические операции

В языке программирования STL используются элементарные двоичные функции AND (И), OR (ИЛИ), Exclusive OR (Исключающее ИЛИ). Эти функции связаны с проверкой сигнала на состояние "1" и "0".

A *адрес бита*
проверка сигнала на состояние "1" и
комбинирование в соответствии с функцией AND (И)

AN *адрес бита*
проверка сигнала на состояние "0" и
комбинирование в соответствии с функцией AND (И)

O *адрес бита*
проверка сигнала на состояние "1" и
комбинирование в соответствии с функцией OR (ИЛИ)

ON *адрес бита*
проверка сигнала на состояние "0" и
комбинирование в соответствии с функцией OR (ИЛИ)

X *адрес бита*
проверка сигнала на состояние "1" и
комбинирование в соответствии с Exclusive OR (Исключающее ИЛИ)

XN *адрес бита*
проверка сигнала на состояние "0" и
комбинирование в соответствии с Exclusive OR (Исключающее ИЛИ)

Проверка сигнала на состояние "1" дает результат проверки "1", если состояние сигнала бита соответствует "1". Проверка сигнала на состояние "0" дает результат проверки "1", если состояние сигнала бита соответствует "0".

CPU комбинирует результат проверки бита с текущим результатом логической операции RLO в соответствии с определенной функцией и формирует новое значение RLO. Если двоичная логическая операция следует сразу же за операцией с памятью, результат проверки вводится в RLO-буфер без выполнения логической операции.

Число двоичных функций и диапазон их действия теоретически произвольны. На практике, однако, ограничения на использование этих функций накладываются из-за конечных размеров блока и рабочей памяти CPU.

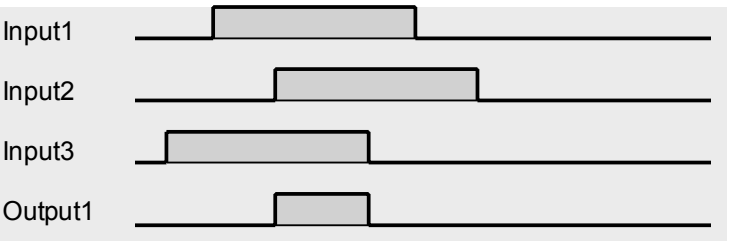
4.2.1 Функция AND (И)

Функция AND (И) связывает два двоичных сигнала и возвращает результат RLO, равный "1", если оба эти сигнала (оба результата проверки) равны "1". Если функция AND (И) встречается несколько раз подряд, то для того, чтобы общий результат RLO был равен "1", нужно, чтобы все промежуточные RLO были равны "1", иначе общий результат будет равен "0". На рис. 4.3 показан

Функция AND (И)

Network 1 (Сегмент 1)

```
A   Input1;
A   Input2;
A   Input3;
=   Output1;
```



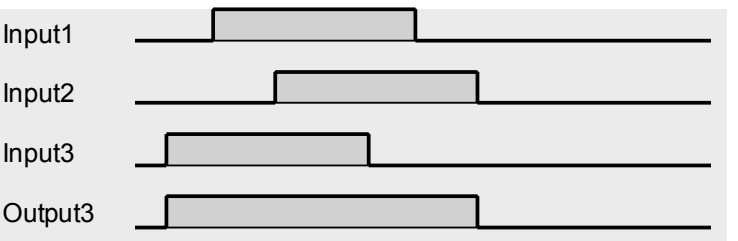
Network 2 (Сегмент 2)

```
A   Input4;
AN  Input5;
=   Output2;
```

Функция OR (ИЛИ)

Network 3 (Сегмент 3)

```
O   Input1;
O   Input2;
O   Input3;
=   Output3;
```



Network 4 (Сегмент 4)

```
O   Input4;
ON  Input5;
=   Output4;
```

Функция Exclusive OR (Исключающее ИЛИ)

Network 5 (Сегмент 5)

```
X   Input1;
X   Input2;
=   Output5;
```

Network 6 (Сегмент 6)

```
X   Input4;
XN  Input5;
=   Output6;
```

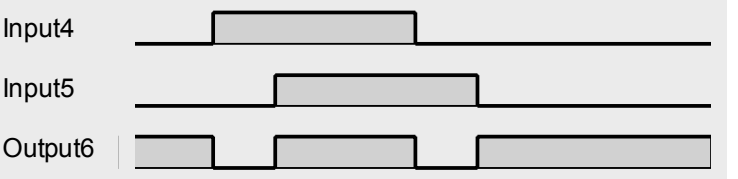


Рис. 4.3 Элементарные двоичные функции

пример действия функции AND (И). В сегменте 1 функция AND (И) имеет три входа (Input1 ... Input3) и один выход (Output1); за этими идентификаторами могут стоять произвольные адреса битов. На всех трех входах сигнал проверяется на состояние "1", так что сигналы этих битов связываются непосредственно по схеме AND (И). Когда все три входных бита дают при проверке состояние сигнала, равное "1", оператор присваивания устанавливает бит Output1 в состояние "1". Во всех других случаях условие AND (И) не выполняется, и бит Output1 находится в состоянии "0".

В сегменте 2 функция AND (И) имеет два входа, один из которых (Input5) инвертирующий, и один выход (Output2). Суть инвертирования для входа Input5 заключается в том, что сигнал на этом входе проверяется на состояние "0", т.е. результат проверки равен "1" при сигнале, равном "0". Поэтому условие AND (И) выполняется, когда бит Input4 находится в состоянии "1", а бит Input5 находится в состоянии "0".

4.2.2 Функция OR (ИЛИ)

Функция OR (ИЛИ) связывает два двоичных сигнала и возвращает результат RLO, равный "1", если хоть один из этих сигналов (один из результатов проверки) равен "1". Если функция OR (ИЛИ) встречается несколько раз подряд, то для того, чтобы общий результат RLO был равен "1", нужно, чтобы хоть один промежуточный RLO был равен "1". Если все промежуточные RLO равны "0", то общий результат будет равен "0". На рис. 4.3 показан пример действия функции OR (ИЛИ). В сегменте 3 функция OR (ИЛИ) имеет три входа (Input1 ... Input3) и один выход (Output3); за этими идентификаторами могут стоять произвольные адреса битов. На всех трех входах сигнал проверяется на состояние "1", так что сигналы этих битов связываются непосредственно по схеме OR (ИЛИ). Если хоть один из входных битов дает при проверке состояние сигнала, равное "1", следующий оператор присваивания устанавливает бит Output3 в состояние "1". Если все промежуточные RLO равны "0", то условие OR (ИЛИ) не выполняется, и бит Output3 устанавливается в состояние "0".

В сегменте 4 функция OR (ИЛИ) имеет два входа, один из которых (Input5) инвертирующий, и один выход (Output4). Суть инвертирования для входа Input5 заключается в том, что сигнал на этом входе проверяется на состояние "0", т.е. результат проверки равен "1" при сигнале, равном "0". Поэтому условие OR (ИЛИ) выполняется, когда бит Input4 находится в состоянии "1", или когда бит Input5 находится в состоянии "0".

4.2.3 Функция Exclusive OR (Исключающее ИЛИ)

Функция Exclusive OR (Исключающее ИЛИ) связывает два двоичных сигнала и возвращает результат RLO, равный "1", если оба эти сигнала (оба результата проверки) имеют разные значения; с другой стороны, RLO равен "0", если оба эти сигнала имеют одинаковое значение.

На рис. 4.3 показан пример действия функции Exclusive OR (Исключающее ИЛИ). В сегменте 5 функция Exclusive OR (Исключающее ИЛИ) имеет два входа (Input1 и Input2) и один выход (Output5); за этими именами могут стоять

произвольные адреса битов. На обоих входах сигнал проверяется на состояние "1", так что сигналы этих битов связываются непосредственно по схеме Exclusive OR (Исключающее ИЛИ). Если только один из входных битов дает при проверке состояние сигнала, равное "1", бит Output5 устанавливается в состояние "1". Если оба входных бита дают при проверке состояния сигнала значения, равные "1", или оба дают значения, равные "0", то бит Output5 устанавливается в состояние "0".

В сегменте 6 функция Exclusive OR (Исключающее ИЛИ) имеет два входа, один из которых (Input5) инвертирующий, и один выход (Output6). Суть инвертирования для входа Input5 заключается в том, что сигнал на этом входе проверяется на состояние "0", т.е. результат проверки равен "1" при сигнале, равном "0". Поэтому условие Exclusive OR (Исключающее ИЛИ) выполняется, когда оба входных бита находятся в одинаковом состоянии, т.е. имеют одинаковое состояние сигнала.

Если функция Exclusive OR (Исключающее ИЛИ) встречается несколько раз подряд, то для того, чтобы общий результат RLO был равен "1", нужно, чтобы нечетное число проверяемых входных битов дало результат проверки, равный "1".

4.2.4 Допущения, принимаемые по отношению к типам датчиков

Элементарные двоичные функции AND (И), OR (ИЛИ), Exclusive OR (Исключающее ИЛИ), описываемые в предыдущих разделах данной главы, рассматривались как входные модули с нормальноразомкнутыми контактами на входах (т.е. датчик имеет нормальноразомкнутые контакты, которые замыкаются при активации датчика, и при этом датчик возвращает значение сигнала, равное "1"). При выполнении функции управления, однако, не всегда возможно использовать только датчики, имеющие нормальноразомкнутые контакты. Во многих случаях, например, в случае "закороченных" цепей (имеющих короткое замыкание), использование нормальнозамкнутых контактов совершенно необходимо (нормальнозамкнутые контакты датчика обеспечивают возвращение сигнала, равного "0", при активации датчика).

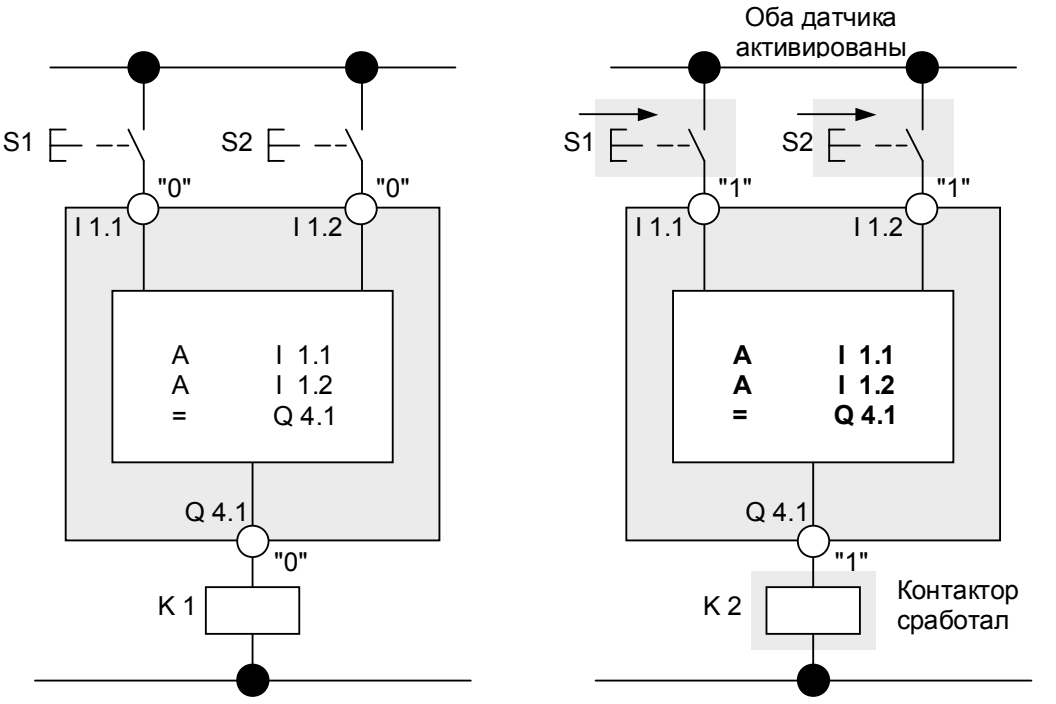
Если датчик, подключенный ко входу, имеет нормальноразомкнутые контакты, то при активации датчика на входе возникает значение сигнала, равное "1". Если датчик, подключенный ко входу, имеет нормальнозамкнутые контакты, то в неактивном состоянии датчика на входе сохраняется значение сигнала, равное "1". CPU "не знает" к какому типу относится датчик на том или ином входе (нормальнозамкнутый или нормальноразомкнутый). Он может только различить состояния сигналов "1" или "0".

При разработке программы, следовательно, необходимо учитывать тип используемого датчика.

Перед вводом программы Вы должны определиться с типом используемого датчика (с нормальнозамкнутыми или с нормальноразомкнутыми контактами). Это необходимо, потому что в отдельных частях программы учитывается состояние датчиков ("Sensor activated" [Датчик активирован], "Sensor not activated" [Датчик не активен]), а, следовательно, Вы должны проверять вход на состояние сигнала "1" или на состояние "0", в зависимости от типа используемого датчика.

На рис. 4.4 показаны варианты программы в зависимости от типа датчика.

Случай 1: оба датчика имеют нормальноразомкнутые контакты:



Случай 2: один нормальноразомкнутый и один нормальнозамкнутый датчики:

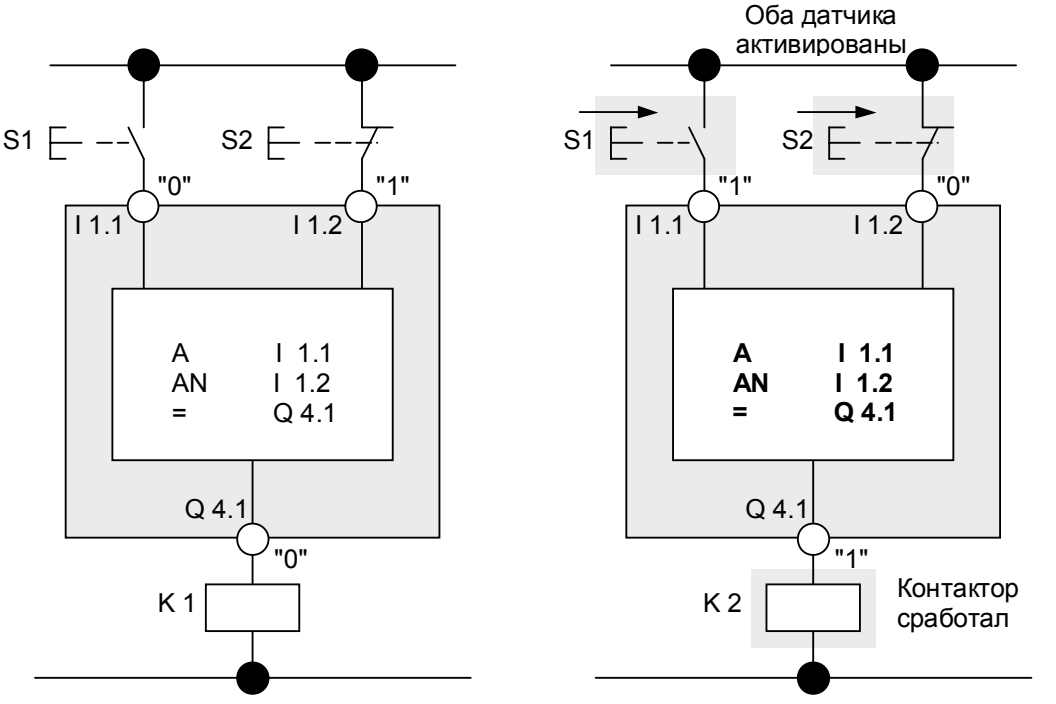


Рис. 4.4 Применение различных типов датчиков

Из вышеприведенного рисунка следует, что благодаря программам, учитывающим тип датчика, и в первом случае (когда оба подключенных к PLC датчика имеют нормальноразомкнутые контакты), и во втором случае (когда к PLC подключены нормальноразомкнутый и нормальнозамкнутый датчики) контакторы, подключенные к выходам, срабатывают при активации двух датчиков на входах.

Если оба подключенных ко входам датчика с нормальноразомкнутыми контактами активируются, сигналы на входах принимают значение "1". Для выполнения условия функции AND (И) с результатом проверки "1", входы функции проверяются на состояние "1". Если активируется датчик с нормальнозамкнутыми контактами, сигнал на входе принимает значение "0". Для выполнения в данном случае условия функции AND (И) с результатом проверки "1", такой вход функции должен проверяться на состояние "0".

4.3 Инвертирование результата логической операции

Оператор NOT (НЕ) инвертирует результат логической операции. Вы можете использовать оператор NOT (НЕ), например, для инвертирования результата выполнения функции AND (И) (см. рис.4.5. Сегмент 7. Функция NAND [НЕ-И]). На этом же рисунке в сегменте 8 показано инвертирование функции OR (ИЛИ), вызванное функцией NOR (НЕ-ИЛИ).

Вы можете найти дополнительные примеры использования оператора NOT (НЕ) в разделе 4.4.6 "Инвертирование во вложенных операторах".

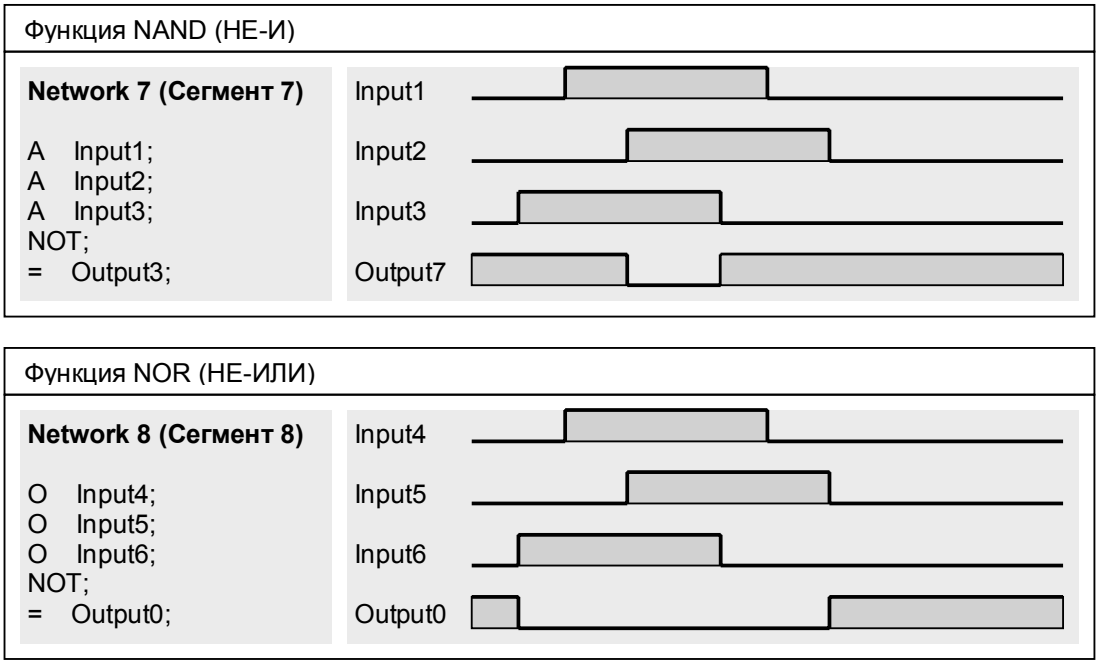


Рис. 4.5 Примеры применения функции NOT (НЕ)

4.4 Сложные двоичные логические операции

Двоичные логические функции могут быть объединены, например, функции AND (И) и OR (ИЛИ) могут стоять в программе в любом порядке. Если такие функции стоят в программе в произвольном порядке, нелегко разобраться в обработке их CPU. Поэтому лучше использовать для иллюстрации решения задачи, например, программирование с помощью функциональных блок-схем, чем программирование на STL.

При программировании сложных двоичных логических операций STL одинаково рассматривает операторы OR (ИЛИ) и Exclusive OR (Исключающее ИЛИ) как операторы с одинаковым приоритетом. Оператор AND (И) имеет более высокий приоритет и выполняется "перед" операторами OR (ИЛИ) и Exclusive OR (Исключающее ИЛИ).

Чтобы функции выполнялись в требуемом порядке CPU иногда должен временно сохранять значение функции (результат логической операции RLO, рассчитанный в некоторой точке программы). Для этой цели (для временного хранения результата) могут использоваться вложенные выражения (вложенные операторы). Как и в случае используемых в булевой алгебре записей вложенные операторы обеспечивают выполнение одних функций раньше других. Вложенные выражения (операторы) могут также включать в себя функцию OR (ИЛИ).

Язык программирования STL позволяет использование следующих двоичных вложенных выражений (вложенных операторов):

O	функция OR (ИЛИ) для функций AND (И)
A(	открывающая скобка с функцией AND (И)
O(	открывающая скобка с функцией OR (ИЛИ)
X(	открывающая скобка с функцией Exclusive OR (Исключающее ИЛИ)
AN(	открывающая скобка с функцией NOT-AND (НЕ-И)
ON(	открывающая скобка с функцией NOT-OR (НЕ-ИЛИ)
XN(	открывающая скобка с функцией NOT-Exclusive OR (НЕ-Исключающее ИЛИ)
)	закрывающая скобка.

Правило логики для выражения с открывающей скобкой показывает, как результат вложенного выражения должен быть связан с текущим значением RLO в момент обработки закрывающей скобки. До этой логической операции результат выполнения вложенного выражения инвертируется, если присутствует символ операции инвертирования.

4.4.1 Обработка вложенных выражений (вложенных операторов)

В языке программирования STL двоичные вложенные выражения используются для определения порядка выполнения двоичных логических операций. В процессе выполнения программы CPU первыми обрабатывает выражения, заключенные в скобки, то есть до выполнения выражений, находящихся за скобками.

Когда CPU встречает открывающую скобку, запоминает текущее значение RLO и затем обрабатывает выражение в скобках (вложенное выражение), когда CPU встречает закрывающую скобку, он связывает значение RLO для вложенного выражения с ранее запомненным значением RLO в соответствии с функцией, определенной при открывающей скобке (Рис. 4.6).

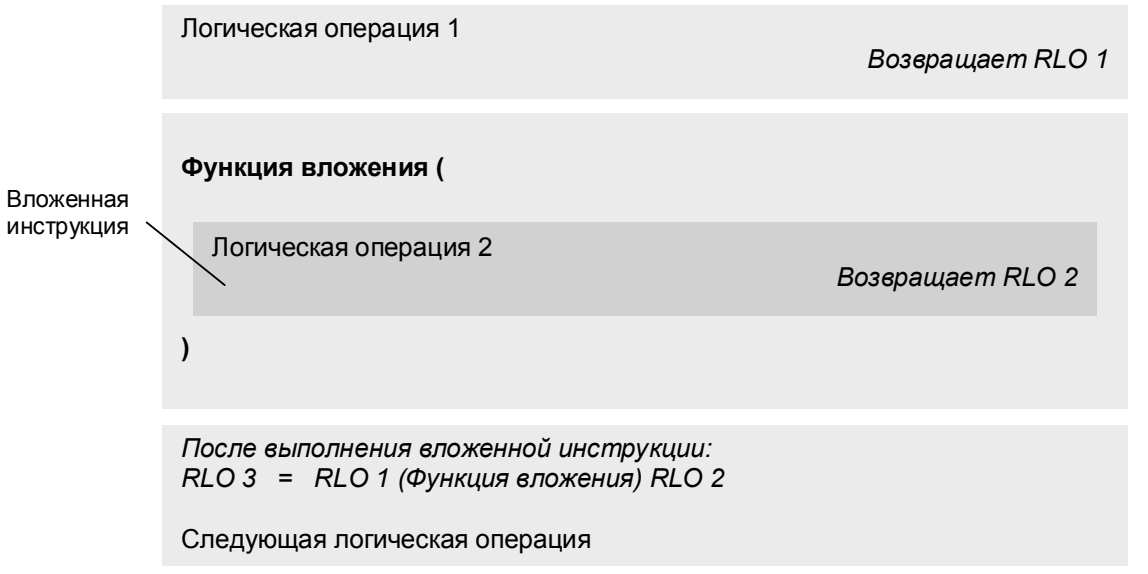


Рис. 4.6 Обработка вложенных выражений

Оператор проверки, следующий за открытой скобкой, всегда является первичным опросом, т.к. CPU всегда создает новый результат логической операции RLO для вложенного выражения. Оператор проверки, следующий за закрытой скобкой, никогда не является первичным опросом, т.к. первой инструкцией является вложенное выражение. CPU обрабатывает значение RLO для вложенного выражения как результат первичного опроса.

Вложенные выражения могут включать в себя другие вложенные выражения (см. рис. 4.7). Глубина вложения равна 7, что означает, что Вы можете 7 раз включать новые вложенные выражения в выражения, которые уже сами по себе являются вложенными, не завершая последних. Обработка внутренних скобок ведется, как описано выше.

Сохранение промежуточных результатов с помощью стека вложения (nesting stack)

При обработке вложенных выражений в CPU заполняется "стек вложения" ("nesting stack") в порядке обработки вложенных функций. В данном стеке хранится следующая информация:

- результат логической операции (RLO) предыдущих скобок;
- двоичный результат (BR "binary result") предыдущих скобок;
- бит состояния (OR) (показывающий, было ли уже выполнено условие функции OR [ИЛИ]);
- функция вложения (для запоминания функции, с которой необходимо связать результат вложенного выражения).

```
X(   ;              // функция XOR (6)
A(   ;              // функция AND (4)
O   Input0;         // функция OR  (1)
O   Input1;
)    ;
A(   ;
O   Input2;         // функция OR  (2)
O   Input3;
)    ;
A   Input4;
)    ;
X(   ;              // функция XOR (6)
A   Input5;         // функция AND (5)
A(   ;
O   Input6;         // функция OR  (3)
O   Input7;
)    ;
)    ;
=   Output20;
```

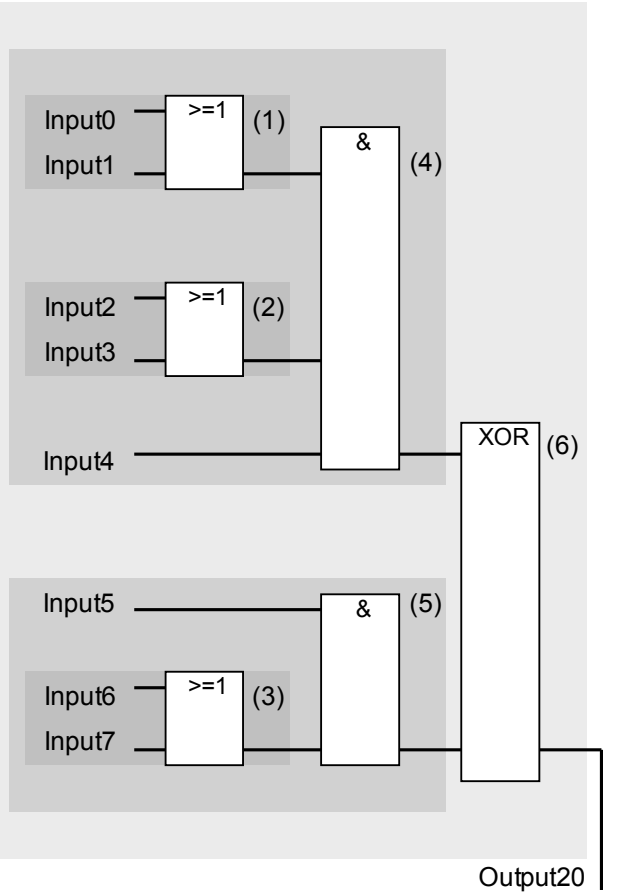


Рис. 4.7 Пример реализации вложенных выражений (операторов)

CPU устанавливает двоичный результат (BR) после закрывающей скобки в состояние, которое имело место к началу обработки вложенного выражения. Во вложенных выражениях Вы можете использовать не только двоичные логические операторы, но и все остальные выражения языка программирования STL. Однако, от программиста требуется внимание при завершении сложного вложенного выражения закрывающей скобкой. Так, возможно, например, во вложенном выражении запрограммировать несколько логических шагов, или операций с памятью, или функций сравнения.

4.4.2 Объединение AND-функций (И) в операторе OR (ИЛИ)

Эти логические операции, представляющие собой комбинации функций OR (ИЛИ) и AND (И), могут быть записаны в булевой алгебре без использования скобок. Здесь действует правило, согласно которому функции AND (И) выполняются в первую очередь. Затем результаты, если есть функция OR (ИЛИ), обрабатываются (link - связываются) этой функцией.

Пример:

```
A     Input0;
A     Input1;
O     ;
A     Input2;
A     Input3;
=     Output8;
```

В данном примере единственный оператор O (соответствующий функции OR (ИЛИ)) находится между двумя функциями AND (И).

В примере Output8 устанавливается, если { Input0 И *Input1}* ИЛИ *{ Input2* И *Input3}* установлены (имеют значение "1").

4.4.3 Объединение OR (ИЛИ) и Exclusive OR (Исключающее ИЛИ) в операторе AND (И)

Эти логические операции, представляющие собой комбинации функций OR (ИЛИ) и AND (И), должны быть записаны в булевой алгебре с использованием скобок, с помощью которых указывается, что функции OR (ИЛИ) выполняются в первую очередь, раньше функции AND (И).

Пример:

```
A(    ;
O     Input0;
O     Input1;
)     ;
A(    ;
O     Input2;
O     Input3;
)     ;
=     Output10;
```

В данном примере оператор открытой скобки объединен с функцией AND (И). Функция OR (ИЛИ) находится во вложенном выражении. Закрывающая скобка в данном случае связывает (link) результат функции OR (ИЛИ) (результат логической операции, которая заключена в скобки) с дополнительными проверками, если они есть, по логике функции AND (И).

В примере Output10 устанавливается, если { Input0 ИЛИ *Input1}* И *{ Input2* ИЛИ *Input3}* установлены (имеют значение "1").

Обработка функций Exclusive OR (Исключающее ИЛИ) в операторе AND (И) производится и записывается точно таким же образом. В вышеуказанном примере функции OR (ИЛИ) должны быть заменены на функции Exclusive OR (Исключающее ИЛИ), так как они имеют одинаковый приоритет.

4.4.4 Объединение функций AND (И) в операторе Exclusive OR (Исключающее ИЛИ)

Функции AND (И), выполняющиеся перед функцией Exclusive OR (Исключающее ИЛИ), должны быть записаны в скобках. С помощью скобок CPU сохраняет результат выполнения AND (И) функции и затем их комбинирует, возможно с дополнительными проверками, в соответствии с правилами функции Exclusive OR (Исключающее ИЛИ).

Пример:

```
X(   ;
A    Input0;
A    Input1;
)    ;
X(   ;
A    Input2;
A    Input3;
)    ;
=    Output12;
```

В данном примере первая функция AND (И) не нуждается в скобках, так как функция AND (И) имеет более высокий приоритет, чем функция Exclusive OR (Исключающее ИЛИ). Тем не менее, при наличии скобок программа лучше читается.

В примере Output12 устанавливается, если только в одной из скобок {Input0 И Input1} и { Input2 И Input3} выполнено условие функции AND (И).

4.4.5 Объединение функций OR (ИЛИ) в операторе Exclusive OR (Исключающее ИЛИ)

Функции OR (ИЛИ), выполняющиеся перед функцией Exclusive OR (Исключающее ИЛИ), должны быть записаны в скобках. С помощью скобок CPU сохраняет результат выполнения OR (ИЛИ) функции и затем их комбинирует, возможно с дополнительными проверками, в соответствии с правилами функции Exclusive OR (Исключающее ИЛИ).

Пример:

```
X(   ;
O    Input0;
O    Input1;
)    ;
X(   ;
O    Input2;
O    Input3;
)    ;
=    Output14;
```

В примере Output14 устанавливается, если только в одной из скобок {Input0 ИЛИ Input1} и { Input2 ИЛИ Input3} выполнено условие функции OR (ИЛИ).

Обработка функций Exclusive OR (Исключающее ИЛИ) в операторе OR (ИЛИ) производится и записывается точно таким же образом. Для такого случая в вышеуказанном примере функции OR (ИЛИ) должны быть заменены на функции Exclusive OR (Исключающее ИЛИ) и наоборот, так как они имеют одинаковый приоритет.

4.4.6 Инвертирование вложенных выражений

Точно также как Вы можете проверить бит на состояние сигнала "0" (по сути инвертировать состояние бита), Вы также можете инвертировать вложенное выражение (точнее - инвертировать результата этого выражения). Это означает, что CPU после вычисления вложенного выражения должен "взять" его результат в инвертированной форме. Признаком инвертирования результата операции является наличие дополнительного символа N в выражении открывающей скобки.

Пример:

```
AN(   ;
O    Input0;
O    Input1;
)    ;
AN(   ;
X    Input2;
X    Input3;
)    ;
=    Output16;
```

В примере Output16 устанавливается, если ни в одной из скобок - ни в {Input0 ИЛИ Input1}, ни в { Input2 Исключающее ИЛИ Input3} - не выполняется условие вложенных функций.

Второй способ инвертирования вложенных выражений возможен с помощью использования выражения (оператора) NOT. Оператор NOT, записанный перед закрывающей скобкой, инвертирует результат вложенного выражения перед последующей обработкой.

Пример:

```
A(    ;
O    Input0;
O    Input1;
NOT ;
)    ;
```

```
A(    ;
X    Input2;
X    Input3;
NOT ;
)    ;
=    Output17;
```

В данном примере имеет место такая же логическая операция, как и в предыдущем примере. Инвертирование вложенных выражений обеспечивается с помощью использования выражения (оператора) NOT внутри вложенных выражений.

5 Операции с памятью (memory functions)

В данной главе рассматриваются операции с памятью, применяемые в языке программирования STL. К операциям с памятью относятся функции Assign (Присвоение) для "динамичного" управления битами, а также функции Set (Установка бита) и Reset (Сброс бита) для "статического" управления битами. Также к операциям с памятью относится функция проверки наличия фронта сигнала.

Операции с памятью используются в сочетании с двоичными логическими операциями, чтобы воздействовать на значения сигналов (состояния) битов с использованием "результата логической операции" RLO, который генерируется CPU.

Вы можете использовать функции работы с памятью для управления всеми битовыми адресами: адресами области отображения входов/выходов процесса, адресами меркеров, адресами глобальных данных, адресами статических и временных локальных данных.

Примеры, рассмотренные в данной главе, содержатся на дискете, прилагающейся к данной книге в библиотеке STL_Book в разделе "Basic functions" ("Базовые функции") в функциональном блоке FB 105 и исходном файле Chap_5.

5.1 Функция Assign (Присвоение)

Синтаксис:

= *Bit*

Функция присваивает биту результат логической операции

Выражение, содержащее символ присвоения "=", назначает результат логической операции RLO, содержащийся в CPU, биту, указанному в выражении. Если результат логической операции имеет значение "1", то бит устанавливается; если результат логической операции имеет значение "0", то бит сбрасывается (см. рис. 5.1. Сегмент 1).

Если необходимо установить бит, в то время когда RLO имеет значение "0", то значение RLO должно быть сначала инвертировано с помощью оператора NOT до момента выполнения функции присвоения (см. рис. 5.1. Сегмент 2).

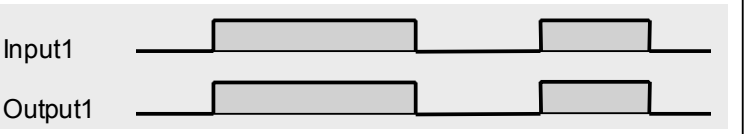
Вы можете найти дополнительные примеры применения функции присвоения в главе 4 "Двоичные логические операции".

Одновременное выполнение нескольких функций присвоения

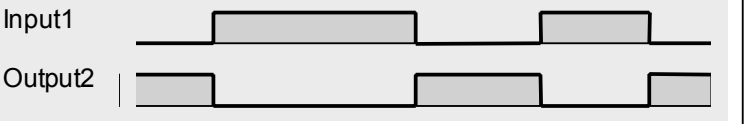
Также допускается выполнять одновременное присваивание результата логической операции различным битам, запрограммировав подряд несколько операторов присваивания соответствующим битам (см. рис. 5.1. Сегмент 3).

Функция Assign (Присвоение значения биту)

Network 1 (Сегмент 1)
A Input1;
= Output1;

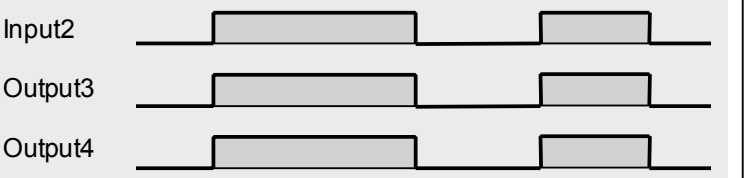


Network 2 (Сегмент 2)
A Input1;
NOT;
= Output2;



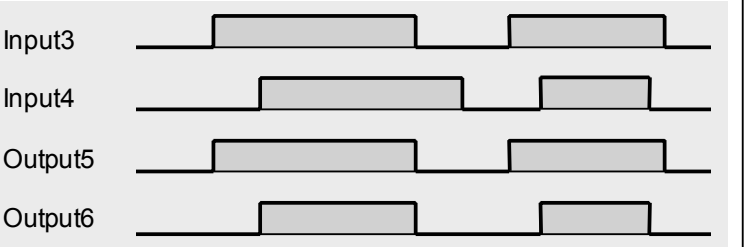
Network 3 (Сегмент 3)

A Input2;
= Output3;
= Output4;



Network 4 (Сегмент 4)

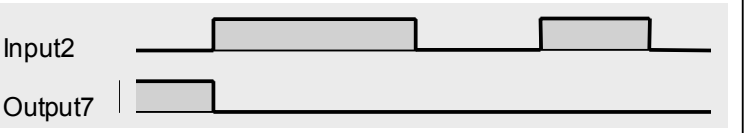
A Input3;
= Output5;
A Output5;
A Input4;
= Output6;



Функции Set и Reset (Установка и сброс бита)

Network 5 (Сегмент 5)
A Input1;
S Output7;

Network 6 (Сегмент 6)
A Input2;
R Output7;



Network 7 (Сегмент 7)

A Input3;
A Input4;
S Output8;
R Output9;
S Output10;
= Output11;

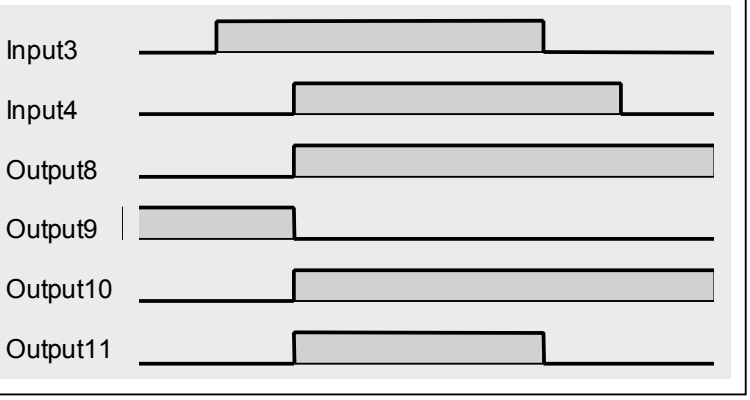
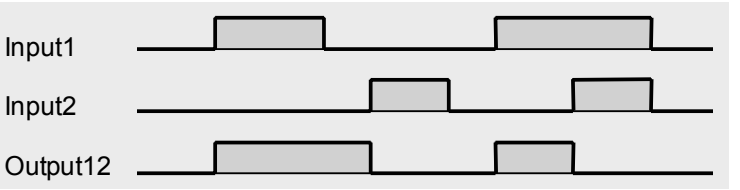


Рис. 5.1 Функции Assign (Присвоение), Set (Установка) и Reset (Сброс)

Функции Set и Reset (Установка и сброс бита)

Network 8 (Сегмент 8)
A Input1;
S Output12;
A Input2;
R Output12;



Network 9 (Сегмент 9)
A Input3;
R Output13;
A Input4;
S Output13;

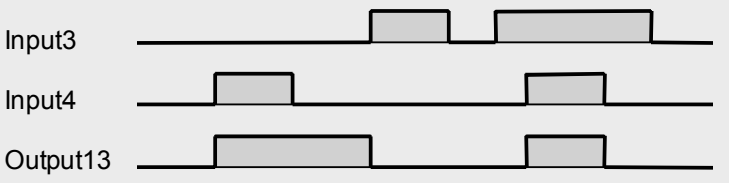


Рис. 5.1 (*продолжение*) Функции Set (Установка) и Reset (Сброс)

Все биты, определенные в функции присвоения Assign, реагируют на эту функцию одинаково, так как инструкции, используемые для управления битами, не влияют на RLO. CPU не генерирует новый результат логической операции RLO до появления следующего оператора проверки.

Если необходимо проверить состояние сигнала выхода в другой логической операции просто проверьте этот выход с помощью соответствующей функции проверки (см. рис. 5.1. Сегмент 4).

5.2 Функции Set (Установка бита) и Reset (Сброс бита)

Синтаксис:

S *Bit*

Функция устанавливает бит, в случае если результат логической операции равен "1".

R *Bit*

Функция сбрасывает бит, в случае если результат логической операции равен "1".

Функция Set (Установка бита) устанавливает бит, а функция Reset (Сброс бита) сбрасывает бит только в случае, если результат логической операции RLO имеет значение "1". Если результат логической операции имеет значение "0", то инструкции Set и Reset не изменяют состояния бита, определенного как операнд в этих инструкциях (см. рис. 5.1. Сегменты 5 и 6).

Одновременное выполнение нескольких операций с памятью

Также допускается использование нескольких операций с памятью одновременно и в любой комбинации, а также совместно с функциями присвоения Assign с использованием одного и того же результата логической

операции. Просто запишите в программе подряд несколько операторов установки или сброса (Set/Reset) для соответствующих битов (см. рис. 5.1. Сегмент 7). Во время действия функции присвоения Assign, установки Set и сброса Reset результат логической операции не изменяет своего значения. При этом CPU не генерирует новый результат логической операции RLO до появления следующего оператора проверки. Также Вы можете использовать оператор NOT (НЕ) для инвертирования значения RLO в одном ряду с операциями с памятью.

Для большей ясности и лучшей читаемости программы рекомендуется использовать операторы Set (Установка бита) или Reset (Сброс бита) парами для определенного бита и только один раз.

5.3 Функции RS Flipflop (RS-триггер)

Функция RS Flipflop (RS-триггер) состоит из одного выражения с оператором Set (Установка бита) и одного выражения, содержащего оператор Reset (Сброс бита); данная функция не имеет специального идентификатора в языке программирования STL. Функция RS Flipflop (RS-триггер) реализуется при последовательной записи выражения с оператором Set (Установка бита) и выражения с оператором Reset (Сброс бита) с общим для этих операторов операндом - одним и тем же битом. С точки зрения функциональных свойств триггера очень важным является порядок, в котором записываются вышеуказанные выражения.

Надо отметить, что биты, используемые в операциях с памятью, обычно сбрасываются при запуске (при полном перезапуске). В особых случаях состояния битов, используемых в операциях с памятью, сохраняют свои значения; это может зависеть от типа запуска (например, "теплый" перезапуск), от используемого бита (например, бит расположен в области статических локальных данных) и от установок CPU (имеются в виду установки свойства реманентности).

5.3.1 Операции с памятью при установленном приоритете функции Reset (Сброс бита)

Приоритет функции Reset (Сброс бита) здесь означает, что при одновременном управлении битом с помощью инструкций Set (Установка бита) и Reset (Сброс бита) и при одновременной выработке этими инструкциями управляющего сигнала "1", указанный бит будет сброшен (значение его сигнала будет равно "0"). Таким образом, здесь имеет место случай, когда инструкция Reset (Сброс бита) имеет приоритет над инструкцией Set (Установка бита) (см. рис. 5.1. Сегмент 8).

Так как выражения обрабатываются последовательно, CPU сначала устанавливает бит, первой выполняя инструкцию Set (Установка бита), затем сбрасывает этот бит, выполняя инструкцию Reset (Сброс бита). После этого до конца цикла сканирования программы выход остается в сброшенном состоянии.

Если рассматриваемый бит является выходом, то столь короткое по времени

его пребывание в установленном состоянии имеет место лишь в области отображения процесса, на этом промежутке времени состояние соответствующего связанного выхода (внешнего) выходного блока не изменяется. До конца цикла сканирования программы CPU не пересылает в выходные модули таблицу выходов образа процесса.

Фактически приоритет функции Reset (Сброс бита) является "стандартной" формой использования данной функции, так как состояние сброса (состояние "0") является, как правило, более безопасным.

5.3.2 Операции с памятью при установленном приоритете функции Set (Установка бита)

Приоритет функции Set (Установка бита) здесь означает, что при одновременном управлении битом с помощью инструкций Set (Установка бита) и Reset (Сброс бита) и при одновременной выработке этими инструкциями управляющего сигнала "1", указанный бит будет установлен (значение его сигнала будет равно "1"). Таким образом, здесь имеет место случай, когда инструкция Set (Установка бита) имеет приоритет над инструкцией Reset (Сброс бита) (см. рис. 5.1. Сегмент 9).

Так как выражения обрабатываются последовательно, CPU сначала сбрасывает этот бит, первой выполняя инструкцию Reset (Сброс бита), выполняя инструкцию Set (Установка бита), затем устанавливает этот бит. После этого до конца цикла сканирования программы выход остается в установленном состоянии.

Если рассматриваемый бит является выходом, то его пребывание в установленном состоянии имеет место в области отображения процесса до конца цикла сканирования программы, на этом промежутке времени состояние соответствующего связанного выхода (внешнего) выходного блока не изменяется. До конца цикла сканирования программы CPU не пересылает в выходные модули таблицу выходов образа процесса.

Фактически приоритет функции Set (Установка бита) является скорее исключением, чем правилом, как форма использования данной функции. Обычно приоритет функции Set (Установка бита) используется, например, для выставления сигнала отказа, в случае когда несмотря на подтверждение на входе Reset (Сброс), текущий сигнал отказа должен поддерживать определенный бит в установленном состоянии с использованием операций с памятью.

5.3.3 Операции с памятью в сочетании с двоичными логическими функциями

В языке программирования STL Вы можете свободно использовать операции с памятью. С их помощью можно сохранять результат логической операции RLO в любом месте программы, чтобы в дальнейшем можно было его использовать.

На рис. 5.2 показано, как можно вместо использования вложенных выражений использовать временное хранение результата логической операции RLO.

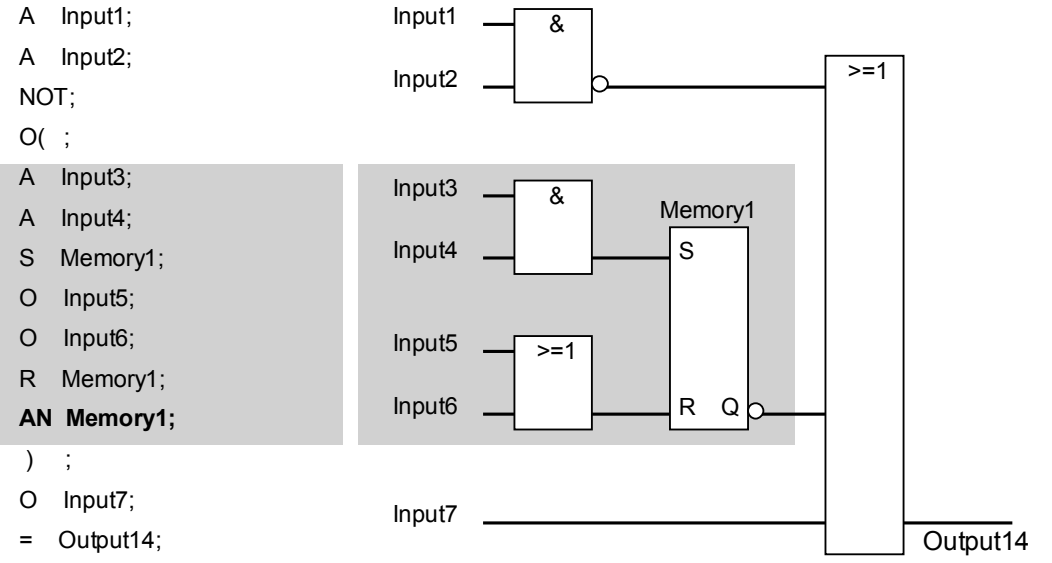


Рис. 5.2 Пример: промежуточный буфер для хранения значения бита

Здесь внутри вложенного выражения используется RS-триггер.

Состояние его сигнала используется в дальнейшем. Поэтому необходимо опросить состояние RS-триггера в конце вложенного выражения для получения состояния сигнала RS-триггера перед оператором закрытия скобки. Если этот оператор отсутствует, то состояние сигнала логической операции перед входом сброса будет в этом случае обрабатываться в дальнейшем.

Внутри скобок Вы можете запрограммировать любое STL-выражение; однако, перед закрытием скобок Вы должны убедиться в том, что Вы получили во вложенном выражении требуемое значение результата логической операции RLO.

Промежуточные двоичные результаты

Почти все биты могут использоваться для временного хранения двоичных результатов, в том числе:

- Биты из области временных локальных данных, являющиеся наиболее подходящим вариантом, если Вам необходимо сохранять промежуточные результаты только внутри блока. Все кодовые блоки имеют области временных локальных данных.
- Биты из области статических локальных данных, доступные только в функциональных блоках и при этом сохраняющие заданные состояния вплоть до нового их определения.

- Биты из области меркеров, предоставляющие повсеместный доступ. Количество меркеров определяется типом CPU. Для четкости работы программы избегайте "множественного" использования меркеров (т.е. избегайте использования одних и тех же меркеров для разных целей).
- Биты в блоках глобальных данных, также предоставляющие повсеместный доступ в программе. Но для того, чтобы их можно было использовать, необходимо сначала открыть соответствующий блок данных (даже если для этого используется полная адресация).

Примечание:

Вы можете заменить "сверхоперативную память" ("scratch-pad memory"), используемую в STEP 5, на область временных локальных данных, которая предоставляется в каждом блоке.

5.4 Функция Edge Evaluation (Проверка наличия фронта сигнала)

Синтаксис:

FP *Bit*

Функция проверки наличия положительного или переднего (возрастающего) фронта сигнала

FN *Bit*

Функция проверки наличия отрицательного или заднего (убывающего) фронта сигнала

Функция проверки наличия переднего или заднего фронта сигнала есть функция для определения наличия изменений в состоянии сигнала (изменения его уровня). Наличие переднего фронта сигнала свидетельствует о переходе сигнала от уровня "0" к уровню "1". Соответственно, наличие заднего фронта сигнала свидетельствует о переходе сигнала от уровня "1" к уровню "0".

В логических переключающих схемах эквивалентом функции проверки наличия фронта сигнала является "контактный формирователь импульса" ("pulse contact element"). При включении реле этот формирователь импульса генерирует импульс, свидетельствующий о наличии возрастающего фронта сигнала. При выключении реле этот формирователь импульса генерирует импульс, свидетельствующий о наличии убывающего фронта сигнала.

К биту, указанному как операнд в функции проверки наличия фронта сигнала, обращаются как к "меркеру фронта" ("edge memory bit") (хотя фактически этот бит может быть и не из области меркеров). Тем не менее, он должен быть таким битом, состояние сигнала которого может быть проверено в любой момент в последующих циклах сканирования программы, и который не используется в программе каким-либо другим образом. Битом в функции проверки наличия фронта сигнала может быть меркер, бит из блока глобальных данных или бит из статических локальных данных в функциональных блоках. (В дальнейшем рассматриваемый бит именуется для определенности "меркером фронта").

Итак, вышеупомянутый меркер фронта сохраняет "старое" значение RLO, которое CPU использовал при последней проверке наличия фронта сигнала.

При каждой новой проверке наличия фронта сигнала CPU сравнивает текущее значение RLO с состоянием меркера фронта. Фронт сигнала будет обнаружен, если эти два сигнала будут иметь различные состояния. В случае обнаружения фронта сигнала CPU обновляет состояние меркера фронта, посредством назначения последнему текущего значения RLO, и устанавливает для RLO значение "1" после положительного или отрицательного фронта сигнала, в зависимости от инструкции функции проверки наличия фронта сигнала. В том случае, если CPU не определяет присутствия фронта сигнала, он устанавливает для RLO значение "0".

Таким образом, состояние бита, равное "1", означает факт обнаружения фронта сигнала. Это состояние бита сохраняется короткое время, как правило, в течение одного цикла сканирования программы. Это происходит из-за того, что CPU не обнаруживает фронта сигнала при следующей проверке наличия фронта сигнала (если значение проверяемого бита не изменяется). Поэтому CPU вновь возвращает RLO значение "0" при следующей проверке наличия фронта.

Вы можете использовать RLO сразу после выполнения проверки наличия фронта сигнала или можете сохранить его значение в бите, называемом "меркер импульса" ("pulse memory bit"). Используйте для сохранения RLO меркер импульса, если Вы должны обработать значение RLO в другом месте программы; это удобный буфер хранения сигнала о наличии фронта сигнала. В качестве "меркера импульса" также может использоваться собственно меркер, бит из блока глобальных данных, а также бит из временных или бит из статических локальных данных.

Непосредственно после выполнения функции обнаружения фронта сигнала Вы можете использовать полученное значение RLO с помощью функций проверки AND (И), OR (ИЛИ) или Exclusive OR (Исключающее ИЛИ).

Проверьте реакцию функции обнаружения фронта сигнала после включения CPU. Для обнаружения фронта сигнала нужно, чтобы RLO до выполнения функции обнаружения фронта и состояние меркера фронта были одинаковыми. При определенных обстоятельствах меркер фронта должен быть сброшен при запуске (в зависимости от требуемой реакции функции проверки наличия фронта, а также от используемого бита).

В следующих примерах показано, как выполняется функция проверки наличия фронта сигнала. На упрощенной схеме показаны вход, начиная с момента времени до начала выполнения проверки наличия фронта синала и меркер импульса ("pulse memory bit"). Понятно, что функции проверки наличия фронта сигнала могут предшествовать, а также после нее могут выполняться двоичные логические операции.

5.4.1 Положительный фронт сигнала

CPU определяет положительный (возрастающий) фронт сигнал, когда до начала выполнения функции проверки наличия фронта результат логической операции изменяется от уровня "0" к уровню "1". Процесс обработки сигналов показан на верхней половине рис. 5.3.

На рисунке 5.3 меркеры импульса имеют имена *PulseMerker*X, а меркеры фронта, соответственно, имеют имена *FrontMerker*X.

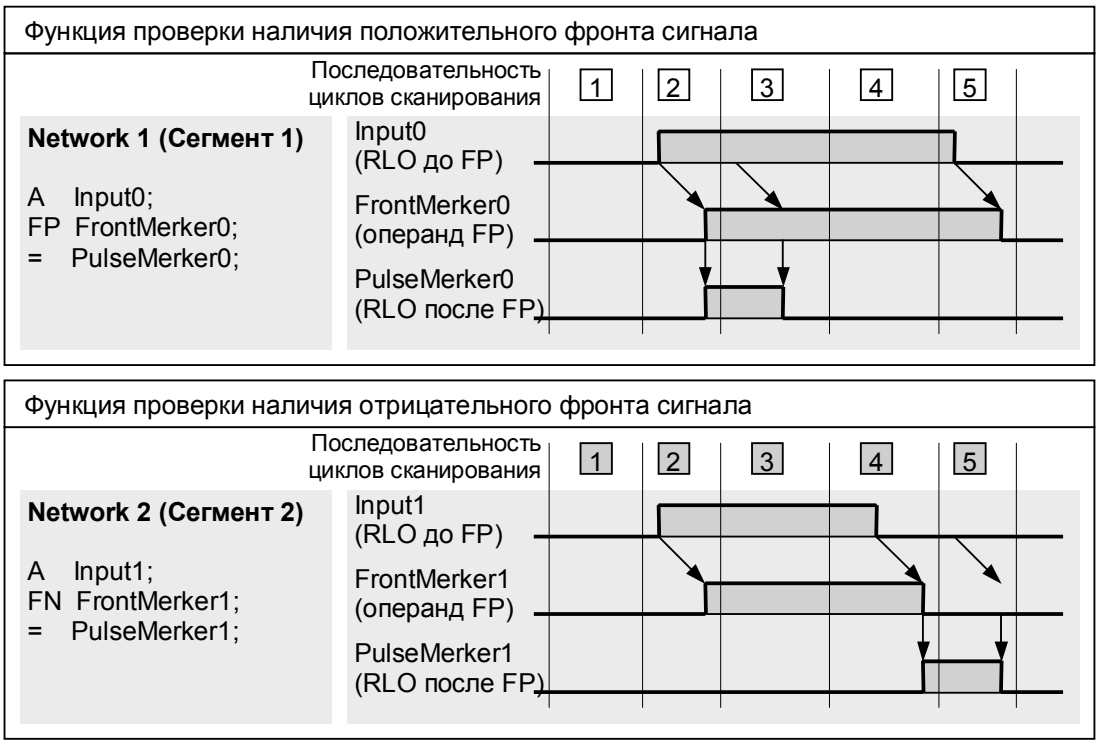


Рис. 5.3 Функция проверки наличия фронта сигнала

На рис. 5.3 показана следующая последовательность циклов сканирования:

1 Сначала состояния и входа Input0, и меркера фронта FrontMerker0 соответствуют уровню "0". Меркер импульса PulseMerker0 также сброшен, т.е. уровень его сигнала равен "0".

2 На 2-ом цикле сканирования состояние входа Input0 изменяется с "0" на "1". CPU обнаруживает это изменение при сравнении текущего значения RLO с состоянием меркера фронта FrontMerker0. Если Input0 равен "1", а меркер фронта FrontMerker0 равен "0", то значение меркера фронта устанавливается (становится равным "1"). Текущее значение PulseMerker0 также равен "1".

3 На 3-ем цикле сканирования при сравнении текущего значения RLO с состоянием меркера фронта FrontMerker0 CPU обнаруживает, что они имеют один уровень. Поэтому он устанавливает для PulseMerker0 значение "0".

4 На 4-ом цикле при оставшихся с предыдущего цикла сканирования значениях текущего RLO и сигнала меркера фронта FrontMerker0 CPU оставляет для PulseMerker0 значение "0", а для меркера фронта FrontMerker0 значение "1".

5 На 5-ом цикле сканирования состояние входа Input0 изменяется с "1" на "0". CPU обнаруживает это изменение и изменяет состояние меркера фронта FrontMerker0 с "1" на "0". При этом не изменяется значение PulseMerker0 (PulseMerker0 остается равным "0"). Таким образом, восстановлено исходное состояние рассматриваемых битов.

5.4.2 Отрицательный фронт сигнала

CPU определяет отрицательный (убывающий) фронт сигнал, когда до начала выполнения функции проверки наличия фронта результат логической операции изменяется от уровня "1" к уровню "0". Процесс обработки сигналов показан на рис. 5.3.

На на нижней половине рис. 5.3 показана следующая последовательность циклов:

1 Сначала состояния входа Input1 и меркера фронта FrontMerker1 соответствуют уровню "0". Меркер импульса PulseMerker1 также сброшен, т.е. уровень его сигнала равен "0".

2 На 2-ом цикле сканирования состояние входа Input1 изменяется с "0" на "1". CPU обнаруживает это изменение при сравнении текущего значения RLO с состоянием меркера фронта FrontMerker1. Если Input1 равен "1", а меркер фронта равен "0", то меркер фронта устанавливается (становится равным "1"). PulseMerker1 остается равным "0".

3 На 3-ем цикле сканирования пока нет разницы между значениями уровней Input1 и FrontMerker1, PulseMerker1 сохраняет значение "0", а меркер фронта FrontMerker1 имеет состояние "1".

4 На 4-ом цикле состояние входа Input1 изменяется с "1" на "0". Обнаружив это, CPU изменяет состояние меркера фронта FrontMerker1 с "1" на "0" и устанавливает для PulseMerker1 значение "1".

5 На 5-ом цикле сканирования не изменяется состояние Input1 и FrontMerker1. CPU устанавливает значение "0" для PulseMerker1. Таким образом, восстановлено исходное состояние битов.

5.4.3 Проверка меркера импульса

Наблюдение за состоянием меркеров импульсов (pulse memory bits) с помощью средств тестирования (тест-функций) программатора PG является трудной задачей, так как эти меркеры остаются в установленном состоянии (состояние сигнала "1") в течение только одного цикла сканирования программы.

Выход также не подходит для такой роли как меркер импульса, так как усилители сигнала выходного модуля или приводы не способны достаточно быстро реагировать на изменения входного сигнала.

С помощью "схемы быстрого перезапуска" ("flying restart circuit"), тем не менее, Вы можете записывать чрезвычайно короткие изменения сигналов меркеров импульсов, используя RS-триггер. Меркер импульса устанавливает RS-триггер, тем самым фиксируя факт прихода фронта сигнала, т.е. RS-триггер запоминает сигнал "Фронт сигнала обнаружен". После проверки этого сигнала Вы можете "сбросить" триггер.

```
O   Pmembit0;
O   Pmembit1;
S   Flipflop2;
```

```
A   Input2;
R   Flipflop2;
```

Таким образом, после проверки "сохраненного фронта" Вы можете вновь "сбросить" триггер.

5.4.4 Проверка наличия фронта в двоичной логической операции

Проверка наличия фронта в двоичной логической операции может служить для решения практической задачи, если Вы используете сигнал, полученный в результате проверки ("импульс") для управления таймером, счетчиком или операцией с памятью. Двоичные операции проверки могут располагаться между операцией проверки наличия фронта и запускаемой функцией.

```
O   Input3;
O   Input4;
FP  EMembit2;
A   Input5;
S   Output15;
A   Input6;
FN  EMembit3;
R   Output15;
```

В примере выход *Output15* устанавливается в момент, когда выполняется OR- (ИЛИ-) условие (когда бит в OR- [ИЛИ-] выражении переходит от состояния "0" к состоянию "1") и вход *Input5* установлен (равен "1"). Выход *Output15* сбрасывается в момент, когда приходит отрицательный фронт на вход *Input6.*

Функция проверки наличия фронта сигнала является "первичным опросом" ("first check"), так как результат логической операции RLO, который генерируется функцией, может быть использован для последующей обработки. Это также означает, что логическая операция до момента начала проверки наличия фронта считается "завершенной" ("completed") (выполненное OR- [ИЛИ-] условие не сохранено). Функция проверки наличия фронта сигнала не влияет на обработку вложенных инструкций.

5.4.5 Двоичный делитель (Binary Scaler)

Двоичный делитель (Binary Scaler) имеет один вход и один выход. Если сигнал на входе двоичного делителя меняет свое состояние, например, с состояния "0" на состояние "1", то выходной сигнал также будет менять свое состояние в соответствии с рис. 5.4.

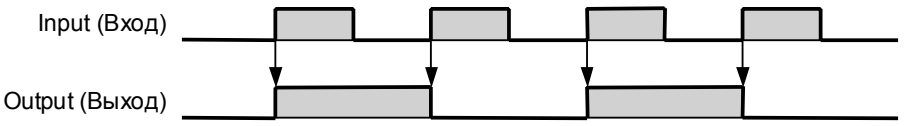


Рис. 5.4 Схема, поясняющая принцип работы двоичного делителя

В нашем примере новое состояние выходной сигнал сохраняет до появления нового положительного фронта сигнала на входе делителя. Только при этом условии выходной сигнал делителя изменяет свое состояние. Это означает, что в этом случае частота выходного сигнала составляет половину частоты переменного входного сигнала.

Существуют различные методы решения такой задачи, два из которых будут рассмотрены ниже.

В первом варианте используется меркер импульса, который устанавливает выход (задает уровень "1"), если он был сброшен (имел состояние "0"), и сбрасывает его (задает уровень "0"), если выход был установлен (имел состояние "1").

При программировании такого решения необходимо помнить, что важно сразу после установки выхода сбросить меркер импульса (иначе выход сразу же будет сброшен вновь).

В нижеследующей программе для первого варианта приняты обозначения:
Input - вход, Output - выход,
EMembit - меркер фронта, PMembit - меркер импульса.

```
      A    Input_1;
      FP   EMembit_1;
      =    PMembit_1;
      A    PMembit_1;
      AN   Output_1;
      S    Output_1;
      R    PMembit_1;
      A    PMembit_1;
      A    Output_1;
      R    Output_1;
```

Второе решение использует условный переход JCN для проверки наличия фронта. Если CPU не обнаруживает фронта сигнала, RLO имеет значение"0" и сканирование программы продолжается с метки перехода.

В случае обнаружения положительного фронта сигнала CPU не выполняет переход к метке, а выполняет следующие два выражения. Здесь, если выход был сброшен (имел состояние "0"), то он устанавливается (получает уровень "1"), если выход был установлен (имел состояние "1"), то он сбрасывается (получает уровень "0). Хотя оператор присвоения управляет выходом, эти последние операторы как бы обладают "свойством запоминания", так как выполняются только при обнаружении положительного фронта сигнала.

```
      A      Input_2;
      FP     EMembit_2;
      JCN    M1
      AN     Output_2;
      =      Output_2;
M1:   ... ;
```

5.5 Пример системы управления ленточным конвейером

Очень простая система управления ленточным конвейером используется в этом разделе в качестве примера для демонстрации двоичных логических операций и операций с памятью для входов, выходов и меркеров.

Описание функций

Детали должны переноситься лентой конвейера; один поддон на ленте.

Особые характеристики представлены ниже:

- Если на ленте конвейера нет деталей, контроллер сообщает об этом с помощью сигнала "readyload" ("готов к загрузке").
- Сигнал "Start" ("запуск") включает движение ленты конвейера для транспортировки деталей.
- Если на оконечном участке конвейера датчик "end-of-belt" ("конец конвейера"), например, фотоэлемент, обнаруживает присутствие деталей, то контроллер сообщает об этом сигналом "ready_rem" ("готов к выгрузке") и останавливает мотор транспортера конвейера.
- По сигналу "Continue" ("продолжить") лента конвейера с деталями продолжает движение, пока датчик "end-of-belt" ("конец конвейера") не обнаружит присутствие деталей.

Функциональная блок-схема системы управления ленточным конвейером показана на рис. 5.5. В примере запрограммированы входы, выходы и меркеры. Он может быть загружен в любое место любого блока. В примере функция без функционального значения была выбрана в качестве блока.

В главе 19 "Параметры блока" такой же пример запрограммирован в функциональном блоке с параметрами; функциональный блок может вызываться многократно (в том числе для нескольких ленточных конвейеров).

Сигналы и символы

Система управления ленточным конвейером использует множество дополнительных сигналов:

- Basic_st
  устанавливает контроллер в исходное состояние.
- Man_on
  активирует движение ленты транспортера, несмотря ни на какие условия.
- /Stop
  останавливает движение ленты транспортера, как только появляется сигнал "0" (датчик в виде нормальнозамкнутого контакта, "zero active" ["активный ноль"]).
- Light_barrier1
  обеспечивает сигнал о достижении деталью конца транспортера.
- /Mfault
  аварийный сигнал от двигателя привода транспортера (например, предохранительный выключатель двигателя); конструкция датчика "zero active" ("активный ноль") обеспечивает аварийный сигнал также и при других видах сбоев, таких, например, как обрыв провода (датчика).

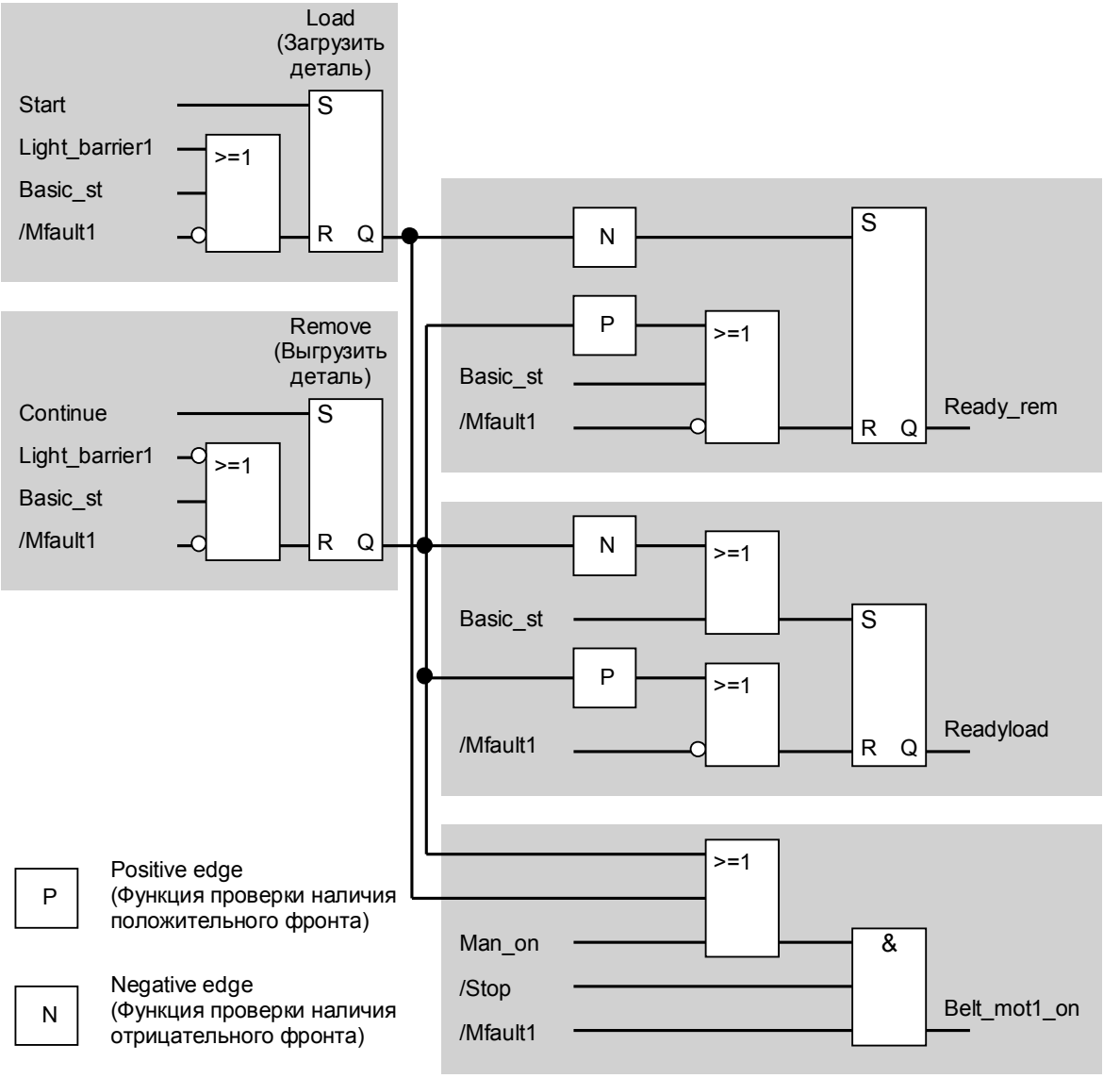


Рис. 5.5 Пример системы управления ленточным конвейером

Если необходимо использовать символьную адресацию, т.е. адресацию посредством символьных имен (символов), то необходимо сгенерировать таблицу символов Symbol Table (см. табл. 5.1) до момента компиляции или инкрементного ввода программы. Эта таблица должна содержать используемые в программе входы, выходы, меркеры и блоки.

Таблица 5.1 Таблица символов Symbol Table для примера системы управления ленточным конвейером

Symbol (Символ)	Address (Адрес)	Data Type (Тип)	Comment (Комментарий)
Belt_control	FC11	FC11	Система управления конвейером
Basic_st	I 0.0	BOOL	Установка контроллеров в исходное состояние
Man_on	I 0.1	BOOL	Включение двигателя привода конвейера
/Stop	I 0.2	BOOL	Остановка двигателя привода конвейера ("zero-active" ["активный ноль"])
Start	I 0.3	BOOL	Запуск конвейера
Continue	I 0.4	BOOL	Подтверждение того, что деталь была снята с конвейера
Light_barrier1	I 1.0	BOOL	Фотоэлемент, датчик "End of belt" ("Конец конвейера") для конвейера №1
/Mfault1	I 2.0	BOOL	Предохранительный выключатель двигателя привода ("zero-active" ["активный ноль"]) для конвейера №1
Readyload	Q 4.0	BOOL	Загрузка новых деталей на транспортер ("Готов к загрузке")
Ready_rem	Q 4.1	BOOL	Выгрузка деталей с транспортера ("Готов к выгрузке")
Belt_mot1_on	Q 5.0	BOOL	Управляющий сигнал на включение двигателя конвейера №1
Load	M 2.0	BOOL	Команда загрузки новых деталей на транспортер
Remove	M 2.1	BOOL	Команда выгрузки деталей с транспортера
EM_Rem_N	M 2.2	BOOL	Меркер фронта (отрицательного) "remove" ("выгрузка детали")
EM_Rem_P	M 2.3	BOOL	Меркер фронта (положительного) "remove" ("выгрузка детали")
EM_Loa_N	M 2.4	BOOL	Меркер фронта (отрицательного) "load" ("загрузка детали")
EM_Loa_P	M 2.5	BOOL	Меркер фронта (положительного) "load" ("загрузка детали")

Программа

Пример программы для системы управления ленточным конвейером расположен в функции без параметров. Эту функцию можно вызывать, например, в организационном блоке OB1 следующим образом:

CALL Belt_control;

Ниже на рис. 5.6 представлен исходный текст программы с символьной адресацией для нашего примера системы управления конвейером.

```
FUNCTION Belt_control :  VOID
TITLE = Control of a conveyor belt
//Пример двоичных логических операций и операций с памятью (без параметров)
NAME    :    Belt1
AUTHOR  :    Berger
FAMILY  :    STL_Book
VERSION :    01.00
BEGIN
NETWORK
TITLE = Load parts
//В этом сегменте выполняется команда "Load", которая начинает перенос
//деталей транспортером
     A    Start;             //Запуск конвейера
     S    Load;
     O    Light_barrier1;    //Детали достигают конца ленты
     O    Basic_st;
     ON   "/Mfault1";        //Предохранительный выключатель мотора
     R    Load;
NETWORK
TITLE = Parts ready for removal
//Когда детали достигли конца конвейера, они могут быть сняты
     A    Load;              //При достижении конца конвейера
     FN   EM_Loa_N;          //"Load" сбрасывается
     S    Ready_rem;         //Детали могут быть сняты
     A    Remove;
     FP   EM_Rem_P;          // Детали сняты
     O    Basic_st;
     ON   "/Mfault1";
     R    Ready_rem;
NETWORK
TITLE = Remove parts
//Команда "Remove" инициирует снятие деталей с транспортера
     A    Continue;          //Включатель реверса конвейера
     S    Remove;
     ON   Light_barrier1;    //Детали сняты с ленты
     O    Basic_st;
     ON   "/Mfault1";        //Предохранительный выключатель мотора
     R    Remove;
NETWORK
TITLE = Belt ready for loading
//Когда детали сняты с конвейера, можно поставить на ленту новые
     A    Remove;
     FN   EM_Rem_N;          //Детали сняты
     O    Basic_st;
     S    Readyload;         //Лента конвейера пуста
     A    Load;
     FP   EM_Loa_P;          //Транспортер запущен
     ON   "/Mfault1";
     R    Readyload;
NETWORK
TITLE = Control belt motor
//Двигатель привода включается и выключается в данном сегменте
     A(;
     O    Load;              //Загрузка деталей на транспортер
     O    Remove;            //Удаление деталей с транспортера
     O    Man_on;            //Запуск с помощью "Man_on" (без реманентности)
     );
```

(см. продолжение программы на следующей странице)

```
     A    "/Stop";          //Остановка двигателя транспортера
     ON   "/Mfault1";       //Предохранительный выключатель мотора
     =    Belt_motor1;
NETWORK
TITLE = Block end
     BE
END_FUNCTION
```

Рис. 5.6 Пример программы для системы управления ленточным конвейером

Глобальные символы могут также использоваться без кавычек (без апострофа), если они не содержат специальных символов. Если же символ (символьное имя) содержит специальный символ (например, пробел [space]), то такое имя должно быть заключено в кавычки. В компилированных блоках редактор STL отображает все глобальные символы в кавычках.

Представленная программа разделена на сегменты с целью большей ясности и лучшей читаемости. Последний сегмент, имеющий заголовок BLOCK END (конец блока), не является необходимым, а служит лишь для обозначения окончания блока. Такой прием бывает очень полезно использовать в случаях, когда блоки имеют чрезмерно большой размер.

6 Функции пересылки данных (move functions)

В данной главе рассматриваются функции для языка программирования STL, с помощью которых выполняются операции обмена данными с помощью аккумуляторов (регистров). К ним относятся следующие функции:

- Функции Load (функции загрузки данных в аккумулятор)
  Функции загрузки используются для загрузки данных в аккумуляторы для последующей обработки (выполнение функций обработки чисел - "digital functions" - операций сравнения, арифметических операций и т.д.)
- Функции Transfer (функции выгрузки данных из аккумулятора) Функции выгрузки используются для выгрузки численных результатов из аккумулятора accumulator1 в память CPU, например, в область меркеров.
- Accumulator functions (функции аккумуляторов) Функции аккумуляторов позволяют передавать информацию из одного аккумулятора в другой или перемещать информацию внутри аккумулятора accumulator1.

Функции Load (функции загрузки данных в аккумулятор) Вам также могут потребоваться для задания начальных значений для таймеров и счетчиков или для обработки текущих значений таймеров и счетчиков.

Системные функции SFC 20 BLKMOV, SFC 81 UBLKMOV и SFC 21 FILL используются для копирования больших объемов информации в памяти или в заданные области данных.

Функции Load (функции загрузки данных в аккумулятор) и функции Transfer (функции выгрузки данных из аккумулятора) также могут потребоваться для адресации модулей с использованием области данных пользователя; если для адресации модулей Вы используете область системных данных, Вы должны использовать системные функции для передачи записей данных.

Эти системные функции также можно использовать для параметризации модулей.

Примеры для этой главы Вы найдете на прилагаемой дискете в библиотеке STL_Book в разделе "Basic Functions" в функциональном блоке FB 106 или в исходном файле Chap_6.

6.1 Общие замечания по поводу операций загрузки и выгрузки данных

Функции Load (функции загрузки данных в аккумулятор) и функции Transfer (функции выгрузки данных из аккумулятора) позволяют производить обмен между различными областями памяти. Такой обмен данными не может производиться непосредственно, а только с использованием аккумулятора

accumulator 1. Аккумулятор - это специальный регистр в процессоре, который выполняет функции промежуточного буфера.

Во время обмена информацией направление, в котором происходит передача данных, указывается в используемой для передачи инструкции. Данные, направляемые из памяти в аккумулятор accumulator 1, называются *загружаемыми* (*loading*), тогда как данные, пересылаемые в обратном направлении, называются *выгружаемыми* (*transferring*) (содержимое аккумулятора "выгружается" ["transferred"] в область памяти).

Операции загрузки данных в аккумулятор и выгрузки данных из аккумулятора являются предопределенными операциями для выполнения *функций обработки чисел (digital functions)*, с помощью которых осуществляется управление численными значениями (digital value) (например, операция сдвига или преобразования) или комбинирование двух чисел (например, операция сложения или сравнения). Для одновременной обработки двух численных значений требуются два промежуточных буфера. В роли таких буферов выступают аккумулятор accumulator 1 и аккумулятор accumulator 2. Все CPU имеют такие специальные регистры. Кроме того, S7-400 CPU имеют два дополнительных промежуточных буфера - аккумулятор accumulator 3 и аккумулятор accumulator 4, которые используются преимущественно в арифметических операциях.

Несколько функций, называемых функциями аккумуляторов (accumulator functions), используются для копирования содержимого одного аккумулятора в другой.

На рис. 6.1 графически показаны соотношения между функциями пересылки данных и области их применения.

Функции загрузки (load) пересылают информацию из системной памяти (system memory), рабочей памяти (work memory) и периферии (I/O) в аккумулятор accumulator 1, смещая при этом "старое" (точнее сказать, "текущее") значение аккумулятора accumulator 1 в аккумулятор accumulator 2.

Функции обработки чисел (digital functions) позволяют управлять содержимым аккумулятора accumulator 1 или комбинировать численные значения, содержащиеся в аккумуляторах accumulator 1 и accumulator 2, с последующей записью результата в аккумулятор accumulator 1.

Функции аккумуляторов (accumulator functions) позволяют получить доступ к содержимому всех аккумуляторов. Источником для пересылки (transfer) информации в системную память (system memory), рабочую память (work memory) и в периферию (I/O) может служить лишь только аккумулятор accumulator 1.

Каждый аккумулятор содержит 32 разряда, тогда как все области памяти имеют байтовую структуру (byte-oriented). Обмен информацией между областями памяти и аккумулятором accumulator 1 может происходить побайтно, по 1 машинному слову и по 1 двойному машинному слову.

В данной главе функции Load (функции загрузки данных в аккумулятор) и функции Transfer (функции выгрузки данных из аккумулятора) обсуждаются в применении к адресным областям входов, выходов, меркеров, периферии (I/O) и для загрузки констант.

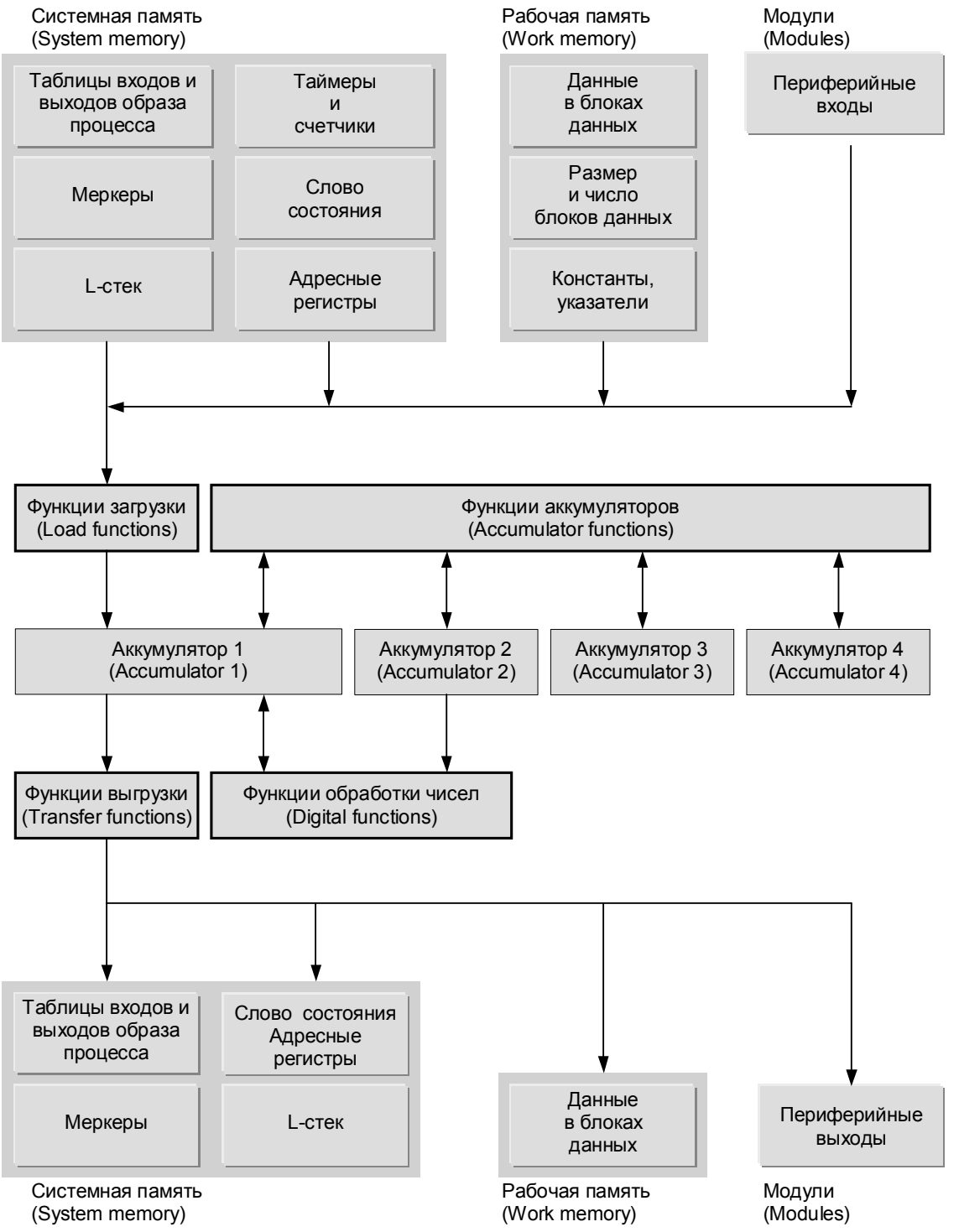


Рис. 6.1 Области памяти для функций загрузки и выгрузки данных

Функции Load (функции загрузки данных в аккумулятор) и функции Transfer (функции выгрузки данных из аккумулятора) могут быть также применены со следующими адресными областями:

- области таймеров и счетчиков
  (глава 7 "Функции таймеров", глава 8 "Функции счетчиков")
- слово состояния
  (глава 15 "Биты состояния")
- области временных локальных данных
  (L-стек, раздел 18.1.5 "Временные локальные данные")
- области адресов данных, длина и число блоков данных
  (глава 18.2 "Функции для блоков данных")
- адресные регистры и указатели
  (глава 25 "Косвенная адресация")
- адреса переменных
  (глава 26 "Прямой доступ к переменным")

6.2 Функции Load (функции загрузки данных в аккумулятор)

6.2.1 Общее представление о функциях загрузки Load

Функция загрузки состоит из оператора L (код операции загрузки) и константы, переменной или адреса с идентификатором адреса, содержимое которого функция будет загружать в аккумулятор accumulator 1.

```
L   +1200       Константа
                 (прямая адресация)
L   IW  16       Местоположение числа
                 (прямая адресация)
L   ActValue     Переменная
                 (символьная адресация)
```

CPU выполняет функцию загрузки независимо от результата логической операции RLO и битов состояния. Функция загрузки не влияет на результат логической операции и не влияет на биты состояния.

Влияние на аккумулятор accumulator 2

Функция Load (функция загрузки данных в аккумулятор) влияет на содержимое аккумулятора accumulator 2. В то время как значение адреса, константы или переменной, определенное в операторе загрузки загружается в аккумулятор accumulator 1, текущее содержимое аккумулятора accumulator 1 пересылается в аккумулятор accumulator 2. Функция загрузки Load полностью пересылает содержимое аккумулятора accumulator 1 в аккумулятор accumulator 2. Предыдущее содержимое аккумулятора accumulator 2 при этом теряется.

При использовании S7-400 CPU функция загрузки данных не влияет на содержимое аккумуляторов accumulator 3 и accumulator 4.

Общая информация об операции загрузки

Адрес числа, определенный в функции загрузки Load, может иметь размер байта (byte), слова (word) или двойного слова (duoble word) (см. рис. 6.2).

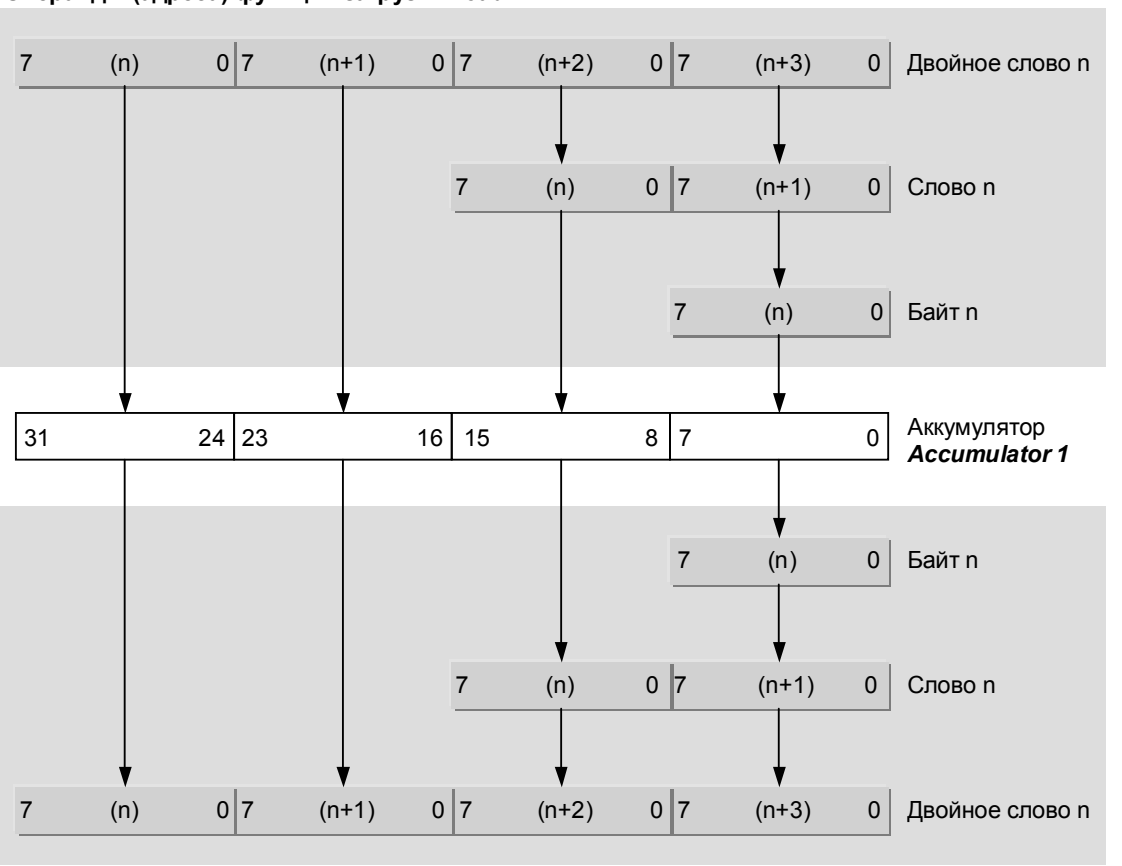


Рис. 6.2 Загрузка и выгрузка байтов, слов, двойных слов

Загрузка байта

При загрузке байта его содержимое считывается в аккумулятор accumulator 1 с выравниванием вправо. Неиспользуемые байты заполняются нулями.

Загрузка слова

При загрузке слова его содержимое считывается в аккумулятор accumulator 1 с выравниванием вправо. При этом старший байт слова (n+1) располагается в аккумуляторе с выравниванием вправо, тогда как младший байт слова (n) располагается в аккумуляторе вплотную слева от байта (n+1). Неиспользуемые байты заполняются нулями.

Загрузка двойного слова

При загрузке двойного слова его содержимое считывается в аккумулятор accumulator 1 с выравниванием вправо. При этом младший байт двойного

слова (n) занимает крайний левый байт аккумулятора, а самый старший байт двойного слова (байт n+3) занимает крайний правый байт аккумулятора.

6.2.2 Загрузка в аккумулятор из памяти

Загрузка данных из области входов

L IB n Загрузка данных из входного байта

L IW n Загрузка данных из входного слова

L ID n Загрузка данных из входного двойного слова

В некоторых типах CPU загрузка данных от входов разрешена, только если к соответствующим входным модулям имеется доступ (см. разд. 1.5.2 "Отображение процесса").

Загрузка данных из области выходов

L QB n Загрузка данных из выходного байта

L QW n Загрузка данных из выходного слова

L QD n Загрузка данных из выходного двойного слова

В некоторых типах CPU загрузка данных от выходов разрешена, только если к соответствующим выходным модулям имеется доступ (см. разд. 1.5.2 "Отображение процесса").

Загрузка данных из периферии (I/O)

L PIB n Загрузка данных из периферийного входного байта

L PIW n Загрузка данных из периферийного входного слова

L PID n Загрузка данных из периферийного входного двойного слова

При загрузке данных от области I/O входные модули адресуются как периферийные входы (PI). При этом доступны только существующие входные модули.

Надо отметить, что при непосредственной загрузке данных от I/O модуля пересылаемое значение может отличаться от значения, загружаемого из области входов образа процесса с тем же адресом, так как эти значения одинаковы лишь в начале цикла сканирования программы (когда CPU обновляет отображение процесса). Непосредственная же загрузка данных от I/O модулей позволяет записать в аккумулятор "текущие" значения сигналов входов.

Загрузка данных из меркеров

L MB n Загрузка данных из байта меркеров

L MW n Загрузка данных из слова меркеров

L MD n Загрузка данных из двойного слова меркеров

Загрузка данных из области меркеров всегда разрешена, так как такая область целиком расположена в CPU. Надо отметить, однако, что разные типы CPU могут иметь разные по размерам области меркеров.

6.2.3 Загрузка констант в аккумулятор

Загрузка констант простых типов

Вы можете загружать константу или фиксированное значение непосредственно в аккумулятор. При этом для улучшения читаемости программы Вы можете выбирать для констант подходящее представление, использовать различные форматы. В главе 3 "SIMATIC S7-программа" Вы можете найти обзор разрешенных форматов для констант. Все константы, которые могут быть загружены в аккумулятор являются константами простых типов.

L	B#16#F1	Загрузка двухразрядного шестнадцатеричного числа
L	-1000	Загрузка целого числа (INT)
L	5.0	Загрузка действительного числа (REAL)
L	S5T#2s	Загрузка данных таймера формата S5
L	C#250	Загрузка числа формата BCD (значение счетчика)
L	TOD#8:30:00	Загрузка времени суток

В главе 24 "Типы данных" описывается назначение битов констант (показана битовая структура данных).

Загрузка указателей

Указатели - это специальная форма констант, которая используется для вычисления позиции в памяти. Вы можете загружать следующие указатели в аккумулятор:

L	P#1.0	Загрузка внутризонного счетчика
L	P#M2.1	Загрузка межзонного счетчика
L	P#*name*	Загрузка адреса локальной переменной

Вы не сможете загрузить указатель DB (DB pointer) или указатель ANY (ANY pointer) в аккумулятор, так как длина этих указателей превышает 32 бита.

В главе 25 "Косвенная адресация" и в главе 26 "Прямой доступ к переменным" Вы найдете дополнительную информацию по данной теме.

6.3 Функции Transfer (функции выгрузки данных из аккумулятора)

6.3.1 Общее представление о функциях выгрузки Transfer

Функция выгрузки состоит из оператора T (код операции выгрузки) и адреса в области памяти, по которому должны быть отправлены данные из аккумулятора accumulator 1.

T	MW120	Переносит содержимое аккумулятора по определенному адресу в память (абсолютная адресация)
T	Setpoint	Переносит содержимое аккумулятора в переменную (символьная адресация)

CPU выполняет функцию выгрузки независимо от результата логической операции RLO и битов состояния. Функция выгрузки не влияет на результат логической операции и не влияет на биты состояния.

Функция выгрузки пересылает содержимое аккумулятора accumulator 1 по одному байту, или по одному слову, или по одному двойному слову в заданный адрес. При этом содержимое аккумулятора accumulator 1 остается неизменным, что делает возможным многократную пересылку данных из аккумулятора accumulator 1.

Функция выгрузки возможна только с помощью аккумулятора accumulator 1. Если есть необходимость передать данные из другого аккумулятора, то Вы должны сначала передать эти данные в аккумулятор accumulator 1, используя функции аккумуляторов, и лишь затем переслать их с помощью функции выгрузки Transfer по требуемому адресу в память.

Общая информация об операции выгрузки

Адрес числа, определенный в функции выгрузки Transfer, может иметь размер байта (byte), слова (word) или двойного слова (duoble word) (см. рис. 6.2).

Выгрузка байта

При выгрузке байта содержимое крайнего правого байта аккумулятора accumulator 1 пересылается в байт по указанному в выражении адресу.

Выгрузка слова

При выгрузке слова содержимое крайнего правого слова аккумулятора accumulator 1 пересылается в слово по указанному в выражении адресу. При этом содержимое старшего байта слова (n+1) в аккумуляторе переносится в слово назначения, где заполняет старший байт (n+1), тогда как младший байт слова (n) в аккумуляторе переносится в слово назначения, где заполняет младший байт (n).

Выгрузка двойного слова

При выгрузке двойного слова его содержимое считывается из аккумулятора accumulator 1 пересылается в двойное слово по указанному в выражении адресу. При этом содержимое младшего байта двойного слова (n) из аккумулятора занимает младший байт (n) двойного слова назначения, а самый старший байт двойного слова (n+3) из аккумулятора занимает самый старший байт двойного слова назначения (n+3).

6.3.2 Выгрузка данных из аккумулятора в различные области памяти

Выгрузка данных во входы

T	IB n	Выгрузка данных во входной байт
T	IW n	Выгрузка данных во входное слово
T	ID n	Выгрузка данных во входное двойное слово

В некоторых типах CPU выгрузка данных в входы разрешена, только если к соответствующим входным модулям имеется доступ (см. разд. 1.5.2 "Отображение процесса").

Пересылка во входы влияет только на биты в области отображения процесса, также как и операции установки и сброса этих битов (входов "inputs"). Возможное применение для таких операций заключается в инициализации значений входов в целях отладки или для запуска с определенными начальными значениями: если Вы запускаете программу на выполнение с заданными значениями входов, то программа начинает выполняться именно с заданными Вами новыми значениями, а не со значениями, полученными от входных модулей.

Выгрузка данных в выходы

T QB n Выгрузка данных в выходной байт

T QW n Выгрузка данных в выходное слово

T QD n Выгрузка данных в выходное двойное слово

В некоторых типах CPU выгрузка данных в выходы разрешена, только если к соответствующим выходным модулям имеется доступ (см. разд. 1.5.2 "Отображение процесса").

Выгрузка данных в периферийные выходы

T PQB n Выгрузка данных в периферийный выходной байт

T PQW n Выгрузка данных в периферийное выходное слово

T PQD n Выгрузка данных в периферийное выходное двойное слово

При выгрузке данных в область I/O выходные модули адресуются как периферийные выходы (PQ). При этом доступны только существующие выходные модули.

Надо отметить, что при непосредственной выгрузке данных в область I/O выходных модулей, связанных с таблицей выходов образа процесса, данные в этой таблице обновляются, ибо нет никакой разницы (в рассматриваемом контексте) между выходом (из области отображения процесса) и периферийным выходом с одинаковым адресом.

Выгрузка данных в область меркеров

T MB n Выгрузка данных в байт меркеров

T MW n Выгрузка данных в слово меркеров

T MD n Выгрузка данных в двойное слово меркеров

Выгрузка данных в область меркеров всегда разрешена, так как такая область целиком расположена в CPU. Надо отметить, однако, что разные типы CPU могут иметь разные по размерам области меркеров.

6.4 Функции аккумуляторов (Accumulator Functions)

Функции аккумуляторов (Accumulator Functions) позволяют пересылать значения из одного аккумулятора в другой или перемещать байты внутри аккумулятора accumulator 1. На выполнение функций аккумуляторов не влияют ни результат логической операции RLO ни биты состояния. Также и результат выполнения этих функций не оказывает влияния ни на RLO, ни на биты состояния.

6.4.1 Прямая пересылка данных между аккумуляторами

PUSH	Сдвиг содержимого аккумулятора "вперед"
POP	Сдвиг содержимого аккумулятора "назад"
TAK	Обмен содержимым между аккумуляторами accumulator 1 и accumulator 2
ENT	Сдвиг содержимого аккумулятора "вперед" (без аккумулятора accumulator 1)
LEAVE	Сдвиг содержимого аккумулятора "назад" (без аккумулятора accumulator 1)

Первые три функции PUSH, POP и TAK используются в CPU, имеющих только два аккумулятора (S7-300 CPU). Все пять функций используются в CPU, имеющих четыре аккумулятора (S7-400 CPU). Схематически выполнение перечисленных выше функций показано на рис. 6.3.

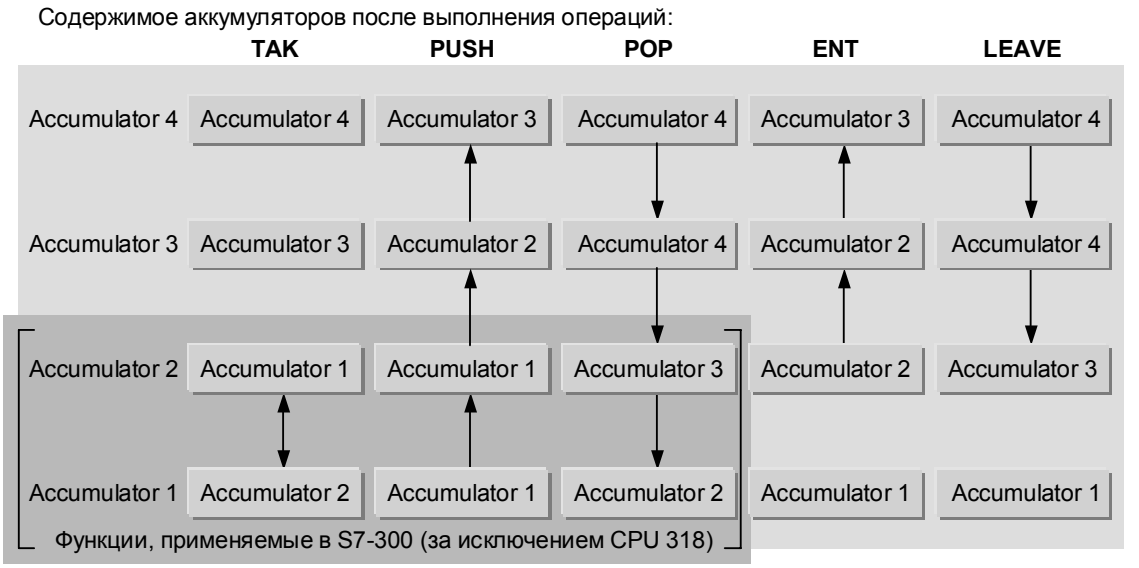


Рис. 6.3 Прямая пересылка данных между аккумуляторами в CPU S7-300 и S7-400

PUSH

Функция PUSH вызывает сдвиг содержимого аккумуляторов вдоль цепочки accumulator 1, accumulator 2, accumulator 3 и accumulator 4. При этом происходит перемещение данных из аккумулятора accumulator 3 в accumulator 4, данных из аккумулятора accumulator 2 в accumulator 3 и копирование данных из аккумулятора accumulator 1 в accumulator 2 (содержимое аккумулятора accumulator 1 остается неизменным).

Таким образом, Вы можете использовать функцию PUSH для внесения одного и того же значения в несколько аккумуляторов.

POP

Функция POP вызывает сдвиг содержимого вдоль цепочки аккумуляторов accumulator 4, accumulator 3, accumulator 2 и accumulator 1. При этом происходит перемещение данных из аккумулятора accumulator 4 в accumulator 3, данных из аккумулятора accumulator 3 в accumulator 2 и данных из аккумулятора accumulator 2 в accumulator 1 (содержимое аккумулятора accumulator 4 при этом остается неизменным).

Таким образом, Вы можете использовать функцию POP для переноса значений из аккумуляторов accumulator 4, accumulator 3 и accumulator 2 в accumulator 1, откуда эти значения могут быть пересланы в память в требуемые адреса.

TAK

Функция TAK вызывает обмен содержимым между аккумуляторами accumulator 1 и accumulator 2. При этом содержимое аккумуляторов accumulator 3 и accumulator 4 остается неизменным.

ENT

Функция ENT вызывает сдвиг содержимого аккумуляторов вдоль цепочки accumulator 2, accumulator 3 и accumulator 4. При этом происходит перемещение данных из аккумулятора accumulator 3 в accumulator 4 и копирование данных из аккумулятора accumulator 2 в accumulator 3. При этом содержимое аккумуляторов accumulator 1 и accumulator 2 остается неизменным.

Если сразу за функцией ENT следует функция загрузки Load, то последняя вызывает вызывает сдвиг содержимого аккумуляторов вдоль цепочки accumulator 1, accumulator 2, accumulator 3 и accumulator 4 (аналогично функции PUSH); при этом новое значение будет располагаться в аккумуляторе accumulator 1.

LEAVE

Функция LEAVE вызывает сдвиг содержимого аккумуляторов вдоль цепочки accumulator 4, accumulator 3 и accumulator 2. При этом происходит перемещение данных из аккумулятора accumulator 3 в accumulator 2 и копирование данных из аккумулятора accumulator 4 в accumulator 3. При этом содержимое аккумуляторов accumulator 4 и accumulator 1 остается неизменным.

Арифметические функции используют функцию LEAVE. Используя данную функцию Вы можете также смоделировать некоторые функциональные возможности других логических операций (например, в логических операциях с операндами формата слова [Word]).

Будучи записанной после логической операции с числами (digital logic operation), функция LEAVE помещает содержимое аккумуляторов accumulator 4 и accumulator 3 соответственно в аккумуляторы accumulator 3 и accumulator 2. При этом результат логической операции с числами (digital logic operation) остается неизменным в аккумуляторе accumulator 1.

6.5 Функции обмена байтами в аккумуляторе accumulator 1

CAW	Обмен местами между байтами младшего слова в аккумуляторе accumulator 1
CAD	Обмен местами между всеми байтами в аккумуляторе accumulator 1

Функция CAW меняет местами два байта в младшем слове в аккумуляторе accumulator 1. При этом байты старшего слова аккумулятора остаются неизменными.

Функция CAD меняет местами все байты в аккумуляторе accumulator 1. При этом самый старший байт становится самым младшим по номеру, а средние два байта меняются местами.

Схематически результат выполнения перечисленных выше функций показано на рис. 6.4.

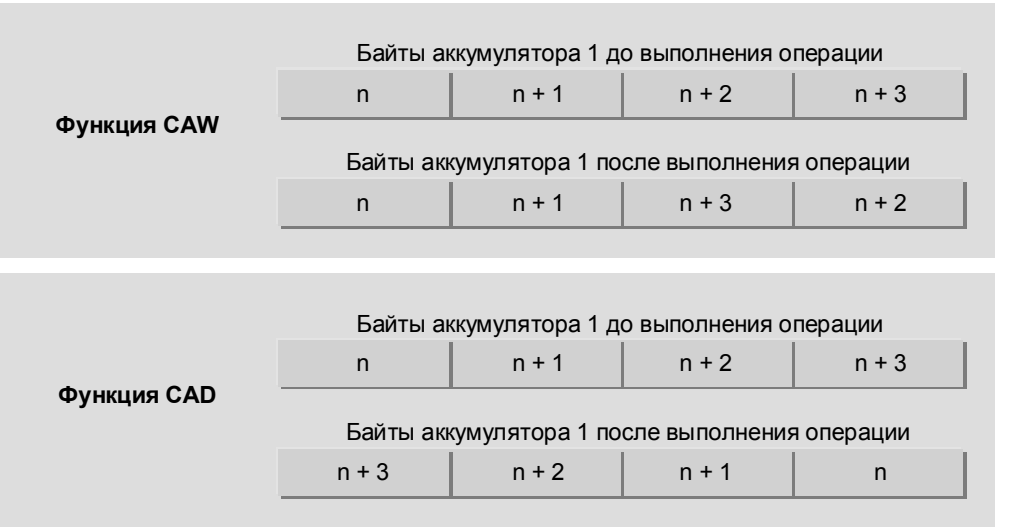


Рис. 6.4. Обмен байтов местами в аккумулятре accumulator 1

6.6 Системные функции для пересылки данных

- SFC 20 BLKMOV
  Системная функция копирования области данных
- SFC 21 FILL
  Системная функция вставки данных в область назначения
- SFC 81 UBLKMOV
  Системная функция непрерывного копирования области данных

Каждая из этих системных функций имеет два параметра типа ANY (см. табл. 6.1). Теоретически каждый из этих параметров может определять произвольный адрес, переменную или абсолютный адрес в памяти.

Таблица 6.1 Параметры для системных функций SFC 20, 21 и 81

SFC	Параметр	Назначение	Тип данных	Содержание, описание
SFC 20	SRCBLK	INPUT	ANY	Область-источник, из которой копируются данные
	RET_VAL	OUTPUT	INT	Информация об ошибках
	DSTBLK	OUTPUT	ANY	Область-приемник, в которую копируются данные
SFC 21	BVAL	INPUT	ANY	Область-источник, из которой копируются данные
	RET_VAL	OUTPUT	INT	Информация об ошибках
	BLK	OUTPUT	ANY	Область-приемник, в которую копируются данные (включая повторное копирование)
SFC 81	SRCBLK	INPUT	ANY	Область-источник, из которой копируются данные
	RET_VAL	OUTPUT	INT	Информация об ошибках
	DSTBLK	OUTPUT	ANY	Область-приемник, в которую копируются данные

Если Вы определяете переменную сложного типа, она должна быть определена целиком ("complete variable"); поэтому компоненты переменной (например, отдельные компоненты массива или структуры) не допускаются (как параметры). Для абсолютной адресации Вы можете использовать указатель ANY; такие указатели рассматриваются в разделе 25.1 "Указатели".

Вы можете также использовать функции для копирования отдельных переменных типа STRING. Тем не менее, в этом случае редактор STL-программ и редактор SCL-программ ведут себя по-разному (см. разд. 6.6.4 "Копирование переменных типа STRING").

Если Вы используете временные локальные данные в виде фактического параметра блока типа ANY, редактор принимает такой параметр, как параметр со структурой указателя ANY. В таком случае Вы можете создать указатель ANY на временные локальные данные, которые могут быть изменены во время работы программы, что означает, что Вы можете создать "область переменной" ("variable area"). В главе 26 "Прямой доступ к переменным" показано, как использовать такие "указатели ANY переменной".

6.6.1 Копирование области данных

Системная функция SFC 20 BLKMOV позволяет выполнить копирование содержания области данных в направлении возрастания (инкрементного) их адресов (параметр SRCBLK) в область назначения (параметр DSTBLK).

При этом могут быть определены следующие фактические параметры:

- любые переменные из адресных областей входов (I), выходов (Q), меркеров (M) и блоков данных (переменные блоков глобальных данных и экземплярных блоков данных),
- переменные из области временных локальных данных (особо для данных типа ANY),
- области данных с абсолютной адресацией посредством указателя ANY.

Используя функцию SFC 20 BLKMOV Вы не можете копировать значения таймеров и счетчиков или передавать информацию как из модулей, так и в модули (область I/O), а также с помощью этой функции нельзя передавть информацию из/в системные блоки данных (SDB).

Что касается входов и выходов (области образа процесса), заданные области могут копироваться вне зависимости от того, определены ли здесь адреса входных или выходных модулей. Вы можете также определить переменную или адрес области в блоке данных в загрузочной (load) памяти (блок данных с ключевым словом UNLINKED [несвязанный]).

При использовании функции SFC 20 BLKMOV область-источник и область-приемник не должны перекрываться. Если область-источник и область-приемник имеют разную длину, то наименьшая по размеру из них будет определять объем данных, переданный в область-приемник.

Пример: Переменная *Frame* (например, структурированная переменная пользовательского типа) в блоке данных "Rec_mailb" должна быть скопирована в переменную *Frame1* (такого же типа как и переменная *Frame*) в блоке данных "Buffer". Значение функции должно быть введено в переменную *Copyerror* в блоке данных "Evaluation".

```
CALL BLKMOV  (
   SRCBLK   := Rec_mailb.Frame,
   RET_VAL  := Evaluation.Copyerror,
   DSTBLK   := Buffer.Frame1);
```

6.6.2 Непрерывное копирование из области данных

Системная функция SFC 81 UBLKMOV позволяет копировать содержимое исходной области данных (параметр SRCBLK) в область назначения (параметр DSTBLK) в направлении возрастания (инкрементного) адресов данных. Операция копирования не может прерываться, поэтому в соответствующих условиях время ожидания обработки прерывания может возрастать. Объем копируемых данных может достигать 512 байт.

Для этой функции могут быть определены следующие фактические параметры:

- любые переменные из адресных областей входов (I), выходов (Q), меркеров (M) и блоков данных (переменные блоков глобальных данных и экземплярных блоков данных),

- переменные из области временных локальных данных (особо для данных типа ANY),
- области данных с абсолютной адресацией посредством указателя ANY.

Используя функцию SFC 81 BLKMOV Вы не можете копировать значения таймеров и счетчиков или передавать информацию как из модулей, так и в модули (область I/O). Также с помощью этой функции нельзя передавть информацию из/в системные блоки данных (SDB) и из/в блоки данных в загрузочной (load) памяти (блок данных с ключевым словом UNLINKED [несвязанный]).

Что касается входов и выходов (области образа процесса), заданные области могут копироваться вне зависимости от того, определены ли здесь адреса входных или выходных модулей.

При использовании функции SFC 81 BLKMOV область-источник и область-приемник не должны перекрываться. Если область-источник и область-приемник имеют разную длину, то наименьшая по размеру из них будет определять объем данных, переданный в область-приемник.

Пример: Из блока данных "Buffer" первый компонент массива *Data* должен быть скопирован в переменную *Frame* в блоке данных "Send_mailb". Значение функции должно быть сохранено в переменной *Copyerror* в блоке данных "Evaluation".

```
CALL UBLKMOV  (
   SRCBLK  := Buffer.Data[1],
   RET_VAL := Evaluation.Copyerror,
   DSTBLK  := Send_mailb.Frame1);
```

6.6.3 Вставка данных в область назначения

Системная функция SFC 21 FILL позволяет копировать содержимое исходной области данных (параметр BVAL) в область назначения (параметр BLK) в направлении возрастания (инкрементного) адресов данных. Операция копирования продолжается до полного заполнения области назначения (при условии превышения размера области назначения над областью-источником возможно многократное копирование исходных данных).

Для этой функции могут быть определены следующие фактические параметры:

- любые переменные из адресных областей входов (I), выходов (Q), меркеров (M) и блоков данных (переменные блоков глобальных данных и экземплярных блоков данных),
- переменные из области временных локальных данных (особо для данных типа ANY),
- области данных с абсолютной адресацией посредством указателя ANY.

Используя функцию SFC 21 FILL Вы не можете копировать значения таймеров и счетчиков или передавать информацию как из модулей, так и в модули (область I/O). Также с помощью этой функции нельзя передавть информацию из/в системные блоки данных (SDB).

Что касается входов и выходов (области образа процесса), заданные области могут копироваться вне зависимости от того, определены ли здесь адреса входных или выходных модулей.

При использовании функции SFC 21 FILL область-источник и область-приемник не должны перекрываться. Если область-источник больше области-приемника, то объем данных, переданный в область-приемник будет соответствовать размеру последнего; если же область-источник меньше области-приемника, то в область-приемник будут записываться (возможно многократно) копируемые данные до его полного заполнения (даже, если размеры рассматриваемых областей не кратны друг другу).

Пример: Блок данных DB 13 содержит 128 байт. Необходимо скопировать во все эти байты содержимое байта меркеров MB 80.

```
CALL SFC 21  (
  BVAL     := MB 80,
  RET_VAL  := MW 32,
  BLK      := P#DB13.DBX0.0 BYTE 128);
```

6.6.4 Копирование переменных типа STRING

Вы можете копировать отдельные переменные типа STRING с использованием системных функций SFC 20 BLKMOV и SFC 81 UBLKMOV. Редакторы STL-программ и SCL-программ ведут себя по-разному при этом.

Редактор STL-программ обрабатывает переменную типа STRING как массив байтов, так что SFC передает отдельные байты один к одному (включая два первые байта со спецификацией размера). Если, например, Вы пересылаете массив байтов в переменную типа STRING, то Вы должны задать правильную длину в первых двух байтах STRING-переменной ("length bytes") в соответствии с передаваемым массивом.

Редактор SCL-программ записывает переменную типа STRING в указатель ANY. При этом SFC передает только соответствующие позиции символов строки STRING-переменной. Если STRING-переменная является областью назначения, то при необходимости фактическая длина ее корректируется. Таким образом Вы можете, например, легко пересылать STRING-переменную в массив данных типа CHAR и наоборот.

Тем не менее, оба редактора - и для STL-программ, и для SCL-программ - копируют STRING-переменную в другую STRING-переменную вполне корректно.

7 Функции таймеров (Timer Functions)

Функции таймеров используются для управления по времени, например, для обеспечения заданного времени ожидания (waiting) или мониторинга (monitoring time), для измерения отрезков времени или для генерации импульсов

В данной главе рассматриваются выражения, содержащие функции таймеров для использования в языке программирования STL. Для языка SCL функции таймеров включены в состав стандартных функций (см. разд. 30.1 "Функции таймеров" ["Timer Functions"]).

Пользователю доступны следующие типы таймеров:

- Таймер с управляемым импульсом (Pulse timer)
- Таймер с расширенным импульсом (Extended pulse timer)
- Таймер с задержкой включения (On-delay timer)
- Таймер с задержкой включения с памятью (Retentive On-delay timer)
- Таймер с задержкой выключения (Off-delay timer)

При запуске таймера Вы можете задавать динамические характеристики (dynamic response), длительность (duration), длительность задержки запуска/выключения таймера. Вы также можете выключить или включить функцию перезапуска ("retrigger") таймеров. Для опроса таймеров используются двоичные логические операции. Функции загрузки (Load) используются для пересылки текущего значения времени в двоичном или BCD-коде в аккумулятор accumulator 1.

Примеры, рассматриваемые в данной главе, и вызовы IEC-таймеров Вы можете найти на прилагаемой дискете в библиотеке STL_Book library в разделе "Basic Functions" в функциональном блоке FB 107 или в исходном файле Chap_7.

7.1 Программирование функций таймеров

7.1.1 Запуск таймера

Таймер запускается (т.е. начинает выполняться функция таймера), если перед переходом CPU к инструкции запуска таймера произошло изменение значения результата логической операции RLO. При этом для таймера с задержкой выключения ("Off-delay timer") RLO должен изменить свое состояние со значения "1" на "0", а для всех остальных типов таймеров RLO должен изменить свое состояние со значения "0" на "1".

Вы можете запустить на выполнение любой из пяти возможных таймеров (см. рис. 7.1).

Таймер запускается как:	На STL	На SCL	
			Сигнал запуска
Таймер с управляемым импульсом (импульсный) (Pulse timer)	SP	S_PULSE	t
Таймер с расширенным импульсом (Extended pulse timer)	SE	S_PEXT	t
Таймер с задержкой включения (On-delay timer)	SD	S_ODT	t
Таймер с задержкой включения с памятью (Retentive on-delay timer)	SS	S_ODTS	t
Таймер с задержкой выключения (Off-delay timer)	SF	S_OFFDT	t

Рис. 7.1 Инструкции запуска для таймеров

7.1.2 Задание временных параметров таймера

При запуске таймера из аккумулятора accumulator 1 выбирается время запуска (running time) или длительность работы (duration). Как и когда в accumulator 1 поступают временные параметры таймера не имеет значения.

Для обеспечения лучшей читаемости программы наилучшим будет вариант, когда эти параметры загружаются в аккумулятор accumulator 1 непосредственно перед запуском функции таймера или в виде константы (непосредственно заданная величина), или в виде переменной (например, слово в памяти, содержащее значение).

Примечание:

аккумулятор accumulator 1 должен содержать корректное значение, даже если отсчет времени не начинается при выполнении инструкции запуска.

Определение длительности работы (импульса) таймера с помощью константы

```
L     S5TIME#10s;          //Длительность 10 с
L     S5T#1m10ms;          //Длительность 1 мин + 10 мс
```

В базовых языках STL, LAD и FBD параметры "время запуска" (running time) и "длительность работы (импульса)" (duration) может быть определены в часах, минутах, секундах и миллисекундах. Диапазон задания временных параметров простирается от S5TIME#10ms до S5TIME#2h46m30s (что соответствует диапазону времени, равному 9990 с). При этом Вы можете использовать как формат S5TIME#, так и формат S5T# для определения временных параметров с помощью константы.

Определение длительности работы (импульса) таймера с помощью переменной

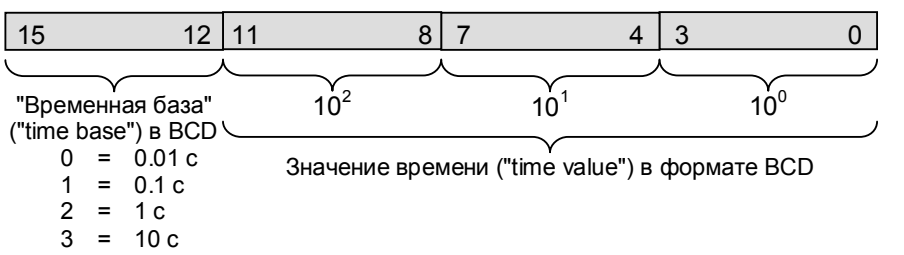
```
L     S5T#10m;             //Длительность (duration) = 10 мин
T     MW20;                //Сохранить "длительность работы"
…     ;
L     MW20;                //Загрузить "длительность работы"
```

Структура параметра "длительность работы (импульса) таймера" (duration)

Внутренняя структура временного параметра "длительность работы (импульса)" ("duration") состоит из "значения времени" ("time value") и "временной базы" ("time base"). Длительность работы (импульса) таймера равна произведению этих величин:

"длительность работы (импульса)" = "значение времени" × "временная база".

Длительность работы (импульса) таймера - это период времени, в течение которого таймер находится в активном состоянии ("timer running"). Значение времени ("time value") представляет собой число отрезков времени в сумме дающих полный период времени, в течение которого таймер должен находиться в активном состоянии ("timer running"). Величина такого отрезка времени равна значению "временной базы" ("time base") и является величиной шага по времени, которая используется операционной системой CPU для "декрементирования" таймера (рис. 7.2).



7.2 Назначение битов параметра "длительность работы" ("duration")

Вы можете непосредственно задать параметр "длительность работы" ("duration") в машинном слове. Наименьшее возможное значение для "временной базы" ("time base") обеспечивает более точное исчисление промежутков времени с помощью таймера. Например, если необходимо задать для таймера отрезок времени, равный 1 с, то можно при этом использовать одно из трех значений для "временной базы" ("time base") и, соответственно, для каждого из этих значений Вы получите свое значение длительности работы функции таймера (фактически, значение функции таймера):

"Временная база" ("time base") = 1 с; "Длительность работы" ("duration") = 2001_{hex}

"Временная база" ("time base") = 100 мс; "Длительность работы" ("duration") = 1010_{hex}

"Временная база" ("time base") = 10 мс; "Длительность работы" ("duration") = 0100_{hex}

Последний из трех вариантов является предпочтительным для данного случая.

При запуске таймера CPU использует заданное значение времени ("time value") как период времени, в течение которого таймер находится в активном состоянии ("timer running"). Операционная система обновляет таймеры через фиксированные интервалы времени независимо от процесса сканирования программы пользователя. То есть, CPU производит ступенчатое уменьшение значения функции таймера в соответствии с заданным значением для "временной базы" ("time base").

Когда значение функции таймера достигает уровня "0", это означает, что отсчитываемое таймером время истекло. При этом CPU изменяет состояние (статус) таймера (состояние сигнала в "0" или "1", в зависимости от выбранного типа таймера), при этом функция таймера перестает быть активной до следующего запуска таймера.

Если для функции таймера было задано значение времени ("time value"), равное нулю (0), то таймер остается активным, пока при обработке функции таймера CPU не обнаружит, что время заданное для таймера "истекло".

Таймеры обновляются асинхронно по отношению к процессу сканирования программы пользователя. Следовательно, возможно, что состояние таймера в начале цикла сканирования отличается от его состояния в конце цикла. Если Вы используете функцию таймера только в одном месте программы и придерживаетесь нижеследующих рекомендаций, то не возникнет ошибок из-за асинхронного обновления таймера в программе.

7.1.3 Сброс таймера (Resetting a timer)

Следующая инструкция

R T n вызывает сброс таймера.

Таймер сбрасывается, при результате логической операции RLO, равном "1", при появлении вышеуказанной инструкции. Пока RLO равен "1", проверки таймера на состояние "1" возвращают результат проверки "0"; проверки таймера на состояние "0" возвращают результат проверки "1".

Сброс таймера устанавливает для значения времени ("time value") и для "временной базы" ("time base") нулевые значения (0).

Примечание:

Сброс функции таймера не сбрасывает внутренний меркер фронта для запуска. Для повторного запуска CPU должен обработать инструкцию запуска таймера при RLO, равном "0", и только после этого при появлении фронта сигнала в соответствующем меркере фронта функция таймера сможет стартовать.

7.1.4 Разблокировка таймера (Enabling a timer)

Следующая инструкция

FR	T n	позволяет перезапуск таймера.

Инструкция FR используется для перезапуска таймера, находящегося в активном состоянии (т.е., когда функция таймера находится в активном состоянии).

Таймер перезапускается, если инструкция FR обрабатывается при положительном (возрастающем) фронте сигнала. При этом внутренний меркер фронта сбрасывается для обеспечения возможности запуска таймера. Если после этого результат логической операции RLO становится равным "1" перед появлением вышеуказанной инструкции, то таймер запускается, даже если при этом не определяется фронт сигнала.

Сброс таймера вызывает изменение величин времени ("time value") и "временной базы" ("time base") на нулевые значения (0).

Примечание:

Данная инструкция разблокировки функции таймера не требуется для запуска или сброса таймера, т.е., инструкция не является необходимой для обычных условий работы с таймером.

7.1.5 Проверка (опрос) таймера (Checking a timer)

Проверка (опрос) состояния таймера

A	T n	проверка сигнала на состояние "1" и комбинирование с RLO в соответствии с логикой AND (И).
O	T n	проверка сигнала на состояние "1" и комбинирование с RLO в соответствии с логикой OR (ИЛИ).
X	T n	проверка сигнала на состояние "1" и комбинирование с RLO в соответствии с логикой Exclusive OR (Исключающее ИЛИ).
AN	T n	проверка сигнала на состояние "0" и комбинирование с RLO в соответствии с логикой AND (И).

ON	T n	проверка сигнала на состояние "0" и комбинирование с RLO в соответствии с логикой OR (ИЛИ).
XN	T n	проверка сигнала на состояние "0" и комбинирование с RLO в соответствии с логикой Exclusive OR (Исключающее ИЛИ).

Вы можете опрашивать таймер, как если бы это был, например, вход (input), и в дальнейшем использовать результат этой проверки. В зависимости от типа таймера проверка (опрос) сигнала на состояние "1" вызывает разное поведение сигнала (импульса) таймера во времени (см. далее описание динамических характеристик).

Как и в случае с опросом входа (input) проверка (опрос) сигнала на состояние "0" возвращает результат проверки таймера с точностью до наоборот по сравнению с опросом на состояние "1" - результат проверки инвертируется.

Проверка (опрос) значения таймера (time value)

Инструкция

L	T n	загружает двоичное значение времени ("time value") таймера,
LC	T n	загружает значение времени ("time value") таймера в двоично-десятичном формате (BCD).

Функции загрузки L T и LC T считывают определенное значение времени ("time value") таймера и пересылают его в аккумулятор accumulator 1. При этом инструкция L загружает значение времени ("time value") в двоичном, а инструкция LC загружает значение времени таймера в двоично-десятичном формате. Значение времени ("time value") таймера, пересылаемое в аккумулятор accumulator 1, представляет собой текущее значение, соответствующее моменту времени, когда производится опрос таймера (если функция таймера находится в активном состоянии, то значение времени, которое загружается в аккумулятор, будет одним из значений, полученным CPU при декрементировании заданного исходного значения в сторону нулевого значения).

Непосредственная загрузка значения таймера (time value)

Значение времени ("time value"), определенное с помощью инструкции таймера, имеет двоичный формат, и может быть переслано в аккумулятор accumulator 1 в этом же (двоичном) формате. В этом случае "временная база" ("time base") будет потеряна и на ее месте в аккумуляторе accumulator 1 будут записаны нули "0".

Следовательно, в аккумуляторе будет значение, которое соответствует положительному целому (INT) числу и которое может быть обработано в дальнейшем с помощью, например, функций сравнения.

Примечание:

Это значение таймера не есть "заданная длительность работы" ("duration").

Пример:

```
L    T 15;      //загрузка текущего значения времени ("time value")
                //таймера
T    MW 34;     //сохранение текущего значения времени ("time value")
```

Загрузка значения таймера (загрузка в формате BCD ["coded load"])

Вы можете также использовать инструкцию для т.н. "кодированной" пересылки ("coded load") в аккумулятор двоичного значения времени ("time value") таймера. В этом случае и значение времени ("time value") и "временная база" ("time base") будут иметь формат BCD (двоично-десятичный). При этом также как и в предыдущем случае в содержимом аккумулятора accumulator 1 в левое машинное слово (в старшее слово) будут записаны нули "0".

Пример:

```
LC   T 16;      //загрузка текущего значения времени ("time value")
                //таймера в BCD-формате
T    MW 122;    //сохранение текущего значения времени ("time value")
```

7.1.6 Последовательность инструкций при использовании функций таймера

При программировании таймера Вам нет необходимости использовать все возможные выражения для активации функций таймеров. Вы должны использовать только те из функций, которые Вам необходимо выполнить. Обычно используется таймер с заданной длительностью импульса и двоичный опрос состояния таймера.

Чтобы выполнить функцию таймера в соответствии с описанием в предыдущих разделах необходимо соблюдать определенный порядок при программировании соответствующих операторов.

В таблице 7.1 показан оптимальный порядок для всех операторов при программировании функций таймера.

Таблица 7.1 Последовательность операторов для таймера

Функция таймера:	Примеры:
Разблокировка таймера (Enable timer)	`A  I  16.5` `FR T  5`

Запуск таймера (Start timer)	`A  I  17.5`
	`L  S5T#1s`
	`SP T  5`

Сброс таймера (Reset timer)	`A  I  18.0` `R  T  5`

Опрос численного значения таймера (Digital timer check)	`L  T  5`
	`T  MW20`
	`LC T  5`
	`T  MW22`

Двоичный опрос таймера (Binary timer check)	`A  T  5` `=  Q  2.0`

При использовании операторов выбирайте те, которые необходимы Вам, и просто пропускайте ненужные.

Если таймер запускается и сбрасывается "одновременно" (как в показанной последовательности операторов), то функция таймера сначала будет запущена на выполнение, а следующее выражение немедленно сбросит таймер. Следовательно, если после этого таймер будет проверен (опрошен), то факт запуска таймера останется незамеченным.

7.1.7 Пример часового генератора (генератора часов)

Пример показывает, как запрограммировать часовой генератор с разным отношением импульс/пауза с помощью отдельного таймера.

Со входа *Start_Input* обеспечивается запуск часового генератора. Если еще не начался отсчет времени или отсчет закончился, генератор запускается в режиме расширенного импульса (extended pulse). При каждом запуске двоичный делитель *Output* меняет состояние сигнала и при этом определяется длительность работы (duration).

```
     AN    Start_input;
     R     Timer;
     R     Output;
     JC    M1;
     A     Timer;
     JC    M1;
     AN    Output;
     =     Output;
     L     Pulse_duration;
     JC    M2;
     L     Pulse_duration;
M2:  SE    Timer;
M1:  ;     //Остальная часть программы
```

7.2 Таймер с управляемым импульсом (Pulse timer)

Ниже представлен пример законченной программы на STL для запуска таймера в режиме "управляемого импульса" (Pulse timer):

```
     A     Enable_input;
     FR    Timer;
     A     Start_input;
     L     Duration;
```

```
SP      Timer;
A       Reset_input;
R       Timer;
L       Timer;
T       Binary_time_value;
LD      Timer;
T       BCD_time_value;
A       Timer;
=       Timer_status;
```

Программа на SCL для вызова таймера в режиме "управляемого импульса" (Pulse timer):

```
BCD_time_value   := S_PULSE (
     T_NO        := Timer,
     S           := Start_input,
     TV          := Duration,
     R           := Reset_input,
     Q           := Timer_status,
     BI          := Binary_time_value);
```

Запуск таймера с управляемым импульсом (Starting a pulse timer)

На рисунке 7.3 показаны динамические характеристики таймера, запускаемого в режиме управляемого импульса (Pulse), и реакция на сброс.

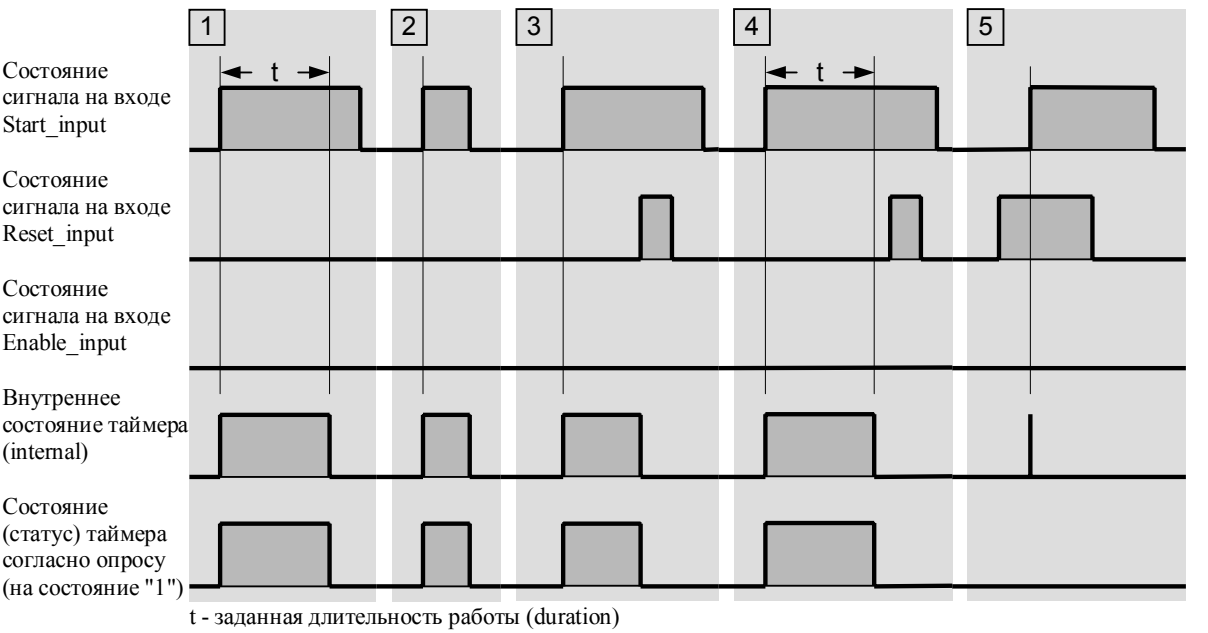


Рис. 7.3 Отклик таймера с управляемым импульсом на сигналы запуска и сброса

Показанное на рис. 7.3 поведение таймера будет соответствовать действительности, если Вы будете придерживаться рекомендованной выше последовательности операторов для STL (сначала запуск, затем сброс, затем опрос таймера). Обычно операция разблокирования (Enabling a timer) не требуется, и для программы на SCL она также не является необходимой.

1 Функция таймера запускается, когда состояние сигнала на входе Start_input меняется от состояния "0" к состоянию "1" (положительный фронт). Таймер работает, пока состояние сигнала на входе Start_input остается равным "1". Проверки (опросы) таймера на состояние "1" (timer status) возвращают результат проверки "1", пока функция таймера активна.

Убывающее значение времени (как значение таймера) отсчитывается от некоторого заданного начального (initial) значения с заданным шагом, равным заданной величине "временной базы" (time base).

2 Функция таймера перестает быть активной, когда состояние сигнала на входе Start_input меняется от состояния "1" к состоянию "0" (отрицательный фронт), если даже это происходит до момента истечения заданного времени работы (длительности - "duration"). После этого проверки (опросы) таймера на состояние "1" (timer status) возвращают результат проверки "0". Значение времени (как значение таймера) показывает время, оставшееся до окончания заданного периода работы (длительности - "duration"), обозначая точку на временной оси, в которой произошло преждевременное прерывание работы таймера.

Сброс таймера с управляемым импульсом (Resetting a pulse timer)

Операция сброса (Resetting a pulse timer) имеет статический эффект и имеет приоритет перед запуском таймера (см. рис. 7.3).

3 Состояние сигнала на входе Reset_input, равное "1", вызывает сброс таймера, если он был до этого активен. После этого проверки (опросы) таймера на состояние "1" (timer status) возвращают результат проверки "0". Значение времени (time value) и значение "временной базы" (time base) также сбрасываются и становятся равными "0". Отрицательный фронт сигнала (т.е. переход его от состояния "1" к "0") на входе Reset_input в то время, пока на входе Start_input присутствует состояние "1", никак не сказывается на режиме таймера.

4 Если функция таймера не активна, то присутствие состояния "1" на входе Reset_input также никак не сказывается на режиме таймера.

5 Если сигнал на входе Start_input изменяет свое состояние с "0" на "1" (присутствует положительный фронт сигнала), в то время, как на входе Reset_input присутствует состояние "1", то таймер запускается, но последующая инструкция сброса немедленно его сбрасывает (на рис. 7.3 это показано жирной вертикальной чертой). Если проверки (опросы) таймера на состояние "1" (timer status) следуют по времени после сброса таймера, то короткое время работы таймера после его запуска не скажется на результатах проверки - результат проверки будет равен "0".

Разблокировка таймера с управляемым импульсом (Enabling a pulse timer)

На рисунке 7.4 показана функция разблокировки таймера, запускаемого в режиме управляемого импульса (Pulse).

Функция разблокировки таймера (Enabling a pulse timer) позволяет вновь запускать (в том числе и уже активный таймер) посредством приложения ко входу Enabling_input положительного фронта сигнала (переход сигнала от состояния "0" к состоянию "1").

Функция разблокировки таймера (Enabling a pulse timer) доступна для использования только в языке программирования STL.

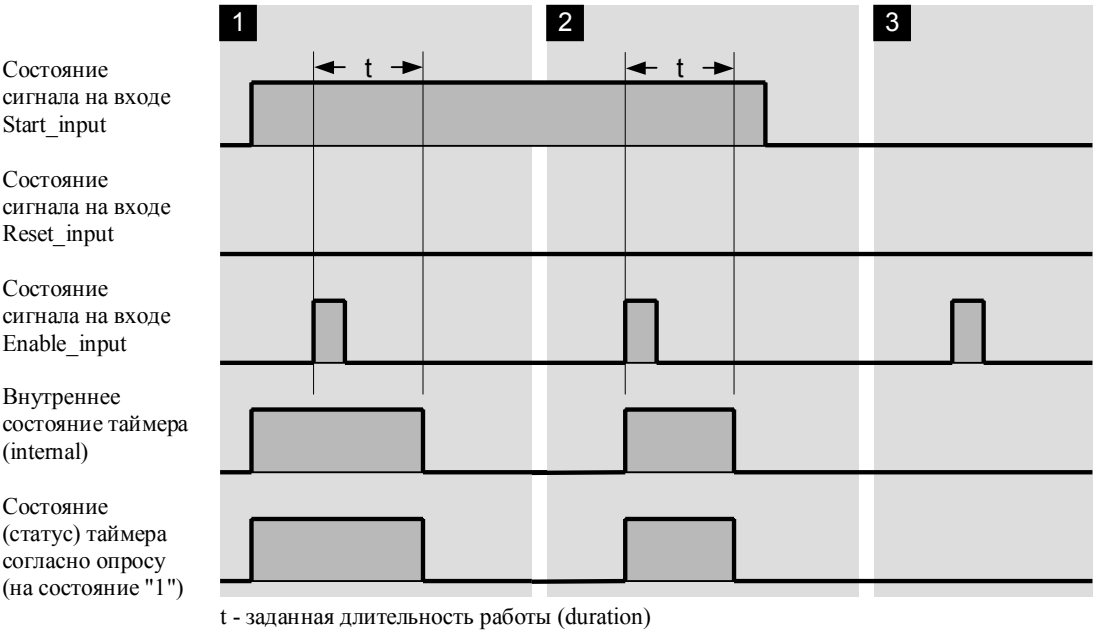


Рис. 7.4 Функция разблокировки (Enabling) таймера с управляемым импульсом

1 Если функция таймера активна и состояние сигнала на входе Enable_input меняется от состояния "0" к состоянию "1" (положительный фронт), то таймер будет перезапущен, если состояние сигнала на входе Start_input остается равным "1". При этом заданное значение длительности работы ("duration") будет вновь взято как исходное значение в момент перезапуска таймера. Последующее изменение сигнала на входе Enable_input от состояния "1" к состоянию "0" (отрицательный фронт) не изменяет режима таймера.

2 Если состояние сигнала на входе Start_input все еще остается равным "1", а функция таймера пассивна, то в момент изменения сигнала на входе Enable_input от состояния "0" к состоянию "1" (положительный фронт) таймер будет вновь запущен в режиме управляемого импульса (pulse timer). При этом заданное значение длительности работы ("duration") будет взято как исходное значение.

3 Если состояние сигнала на входе Start_input равно "0", то изменение сигнала на входе Enable_input от состояния "0" к состоянию "1" (положительный фронт) не будет иметь никакого эффекта.

7.3 Таймер с расширенным импульсом (Extended pulse timer)

Ниже представлен пример законченной программы на STL для запуска таймера в режиме "расширенного импульса" (Extended pulse timer):

```
A       Enable_input;
FR      Timer;
A       Start_input;
L       Duration;
SE      Timer;
A       Reset_input;
R       Timer;
L       Timer;
T       Binary_time_value;
LD      Timer;
T       BCD_time_value;
A       Timer;
=       Timer_status;
```

Программа на SCL для вызова таймера в режиме "расширенного импульса" (Extended pulse timer):

```
BCD_time_value   := S_PEXT (
     T_NO        := Timer,
     S           := Start_input,
     TV          := Duration,
     R           := Reset_input,
     Q           := Timer_status,
     BI          := Binary_time_value);
```

Запуск таймера с расширенным импульсом (Starting an extended pulse timer)

На рисунке 7.5 показаны динамические характеристики таймера, запускаемого в режиме "расширенного импульса" (Extended pulse timer), и его реакция на сигнал сброса.

Показанное на рис. 7.5 поведение таймера будет соответствовать действительности, если Вы будете придерживаться рекомендованной выше последовательности операторов для STL (сначала запуск, затем сброс, затем опрос таймера). Обычно операция разблокирования (Enabling a timer) не требуется для работы таймера в программе STL, и для программы на SCL она также не является необходимой.

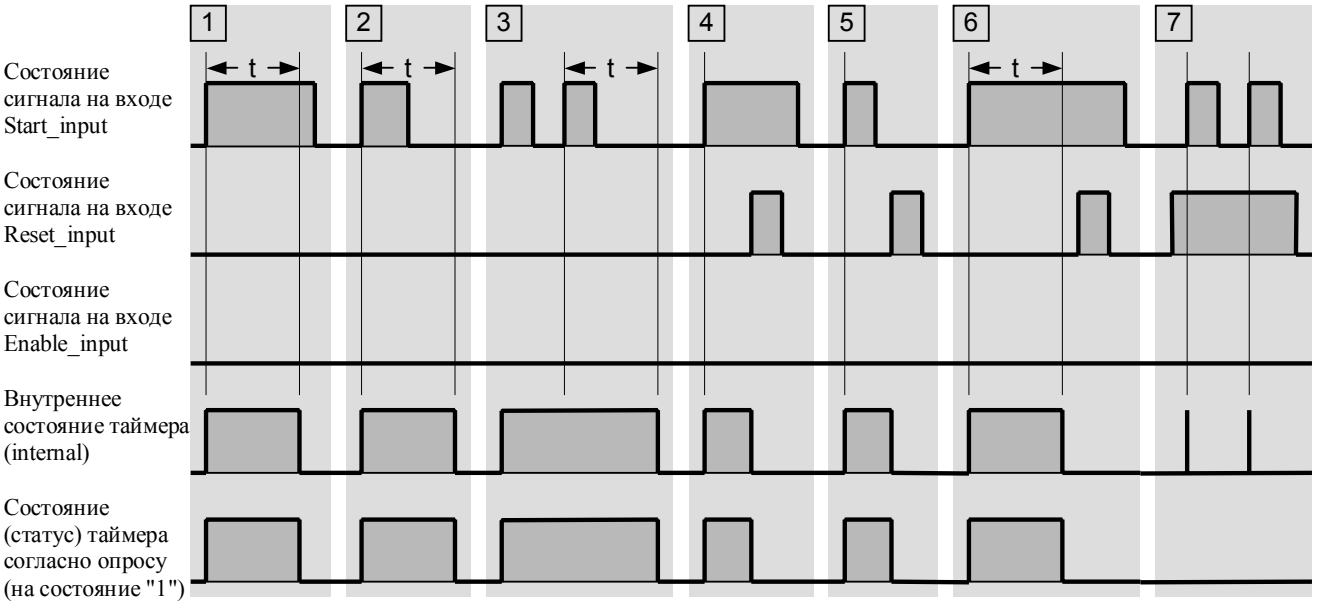


Рис. 7.5 Отклик таймера в режиме "расширенного импульса" (Extended pulse timer) на сигналы запуска и сброса

1 2 Функция таймера запускается, когда состояние сигнала на входе Start_input меняется от состояния "0" к состоянию "1" (положительный фронт). Таймер работает до момента истечения заданного времени работы (длительности - "duration"), даже если до этого момента состояние сигнала на входе Start_input изменится снова от состояния "1" к состоянию "0". Проверки (опросы) таймера на состояние "1" (timer status) возвращают результат проверки "1", пока функция таймера активна.

Убывающее значение времени (как значение таймера) отсчитывается от некоторого заданного начального (initial) значения с заданным шагом, равным заданной величине "временной базы" (time base).

3 Если состояние сигнала на входе Start_input меняется от состояния "0" к состоянию "1" (положительный фронт), в то время, пока функция таймера активна, то таймер будет перезапущен. При этом заданное значение длительности работы ("duration") будет вновь взято как исходное значение в момент перезапуска таймера.

Таким образом таймер может быть перезапущен столько раз, сколько требуется для нормальной работы программы, невзирая на время, оставшееся в предыдущем рабочем периоде таймера.

Сброс таймера с режимом "расширенного импульса" (Extended pulse timer)

Операция сброса таймера с режимом "расширенного импульса" (Extended pulse timer) имеет статический эффект и имеет приоритет перед запуском таймера (см. рис. 7.5).

4 5 Состояние сигнала на входе Reset_input, равное "1", вызывает сброс таймера, если он был до этого активен. После этого проверки (опросы) таймера на состояние "1" (timer status) возвращают результат проверки "0". Значение времени (time value) и значение "временной базы" (time base) также сбрасываются и становятся равными "0". Отрицательный фронт сигнала (т.е. переход его от состояния "1" к "0") на входе Reset_input в то время, пока на входе Start_input присутствует состояние "1", никак не сказывается на режиме таймера.

6 Если функция таймера не активна, то присутствие состояния "1" на входе Reset_input также никак не сказывается на режиме таймера.

7 Если сигнал на входе Start_input изменяет свое состояние с "0" на "1" (присутствует положительный фронт сигнала), в то время, как на входе Reset_input присутствует состояние "1", то таймер запускается, но последующая инструкция сброса немедленно его сбрасывает (на рис. 7.5 это показано жирной вертикальной чертой). Если проверки (опросы) таймера на состояние "1" (timer status) следуют по времени после сброса таймера, то короткое время работы таймера после его запуска не скажется на результатах проверки - результат проверки будет равен "0".

Разблокировка таймера с режимом "расширенного импульса" (Extended pulse timer)

Функция разблокировки таймера (Anabling extended pulse timer) позволяет вновь запускать (в том числе и уже активный таймер) посредством приложения ко входу Enabling_input положительного фронта сигнала (переход сигнала от состояния "0" к состоянию "1").

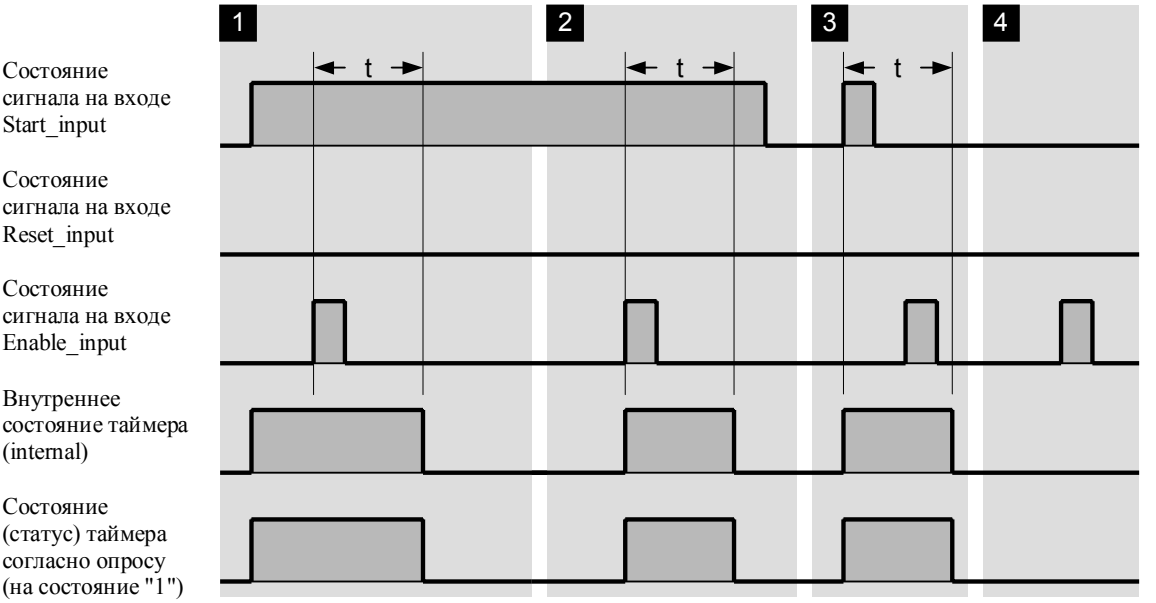
Функция разблокировки таймера (Anabling a pulse timer) доступна для использования только в языке программирования STL.

1 Если функция таймера активна и состояние сигнала на входе Enable_input меняется от состояния "0" к состоянию "1" (положительный фронт), то таймер будет перезапущен, если состояние сигнала на входе Start_input остается равным "1". При этом заданное значение длительности работы ("duration") будет вновь взято как исходное значение в момент перезапуска таймера. Последующее изменение сигнала на входе Enable_input от состояния "1" к состоянию "0" (отрицательный фронт) не изменяет режима таймера.

2 Если состояние сигнала на входе Start_input все еще остается равным "1", а функция таймера пассивна, то в момент изменения сигнала на входе Enable_input от состояния "0" к состоянию "1" (положительный фронт) таймер будет вновь запущен в режиме "расширенного импульса" (extended pulse timer). При этом заданное значение длительности работы ("duration") будет взято как исходное значение.

3 4 Если состояние сигнала на входе Start_input равно "0", то изменение сигнала на входе Enable_input от состояния "0" к состоянию "1" (положительный фронт) не будет иметь никакого эффекта.

На рисунке 7.6 показана функция разблокировки таймера, запускаемого в режиме "расширенного импульса" (Extended pulse timer).



t - заданная длительность работы (duration)

Рис. 7.6 Функция разблокировки (Enabling) таймера с режимом "расширенного импульса" (Extended pulse timer).

7.4 Таймер с задержкой включения (On-delay timer)

Ниже представлен пример законченной программы на STL для запуска таймера с задержкой включения (On-delay timer):

```
A       Enable_input;
FR      Timer;
A       Start_input;
L       Duration;
SD      Timer;
A       Reset_input;
R       Timer;
L       Timer;
T       Binary_time_value;
LD      Timer;
```

```
T       BCD_time_value;
A       Timer;
=       Timer_status;
```

Программа на SCL для вызова таймера с задержкой включения (On-delay timer):

```
BCD_time_value   := S_ODT (
     T_NO        := Timer,
     S           := Start_input,
     TV          := Duration,
     R           := Reset_input,
     Q           := Timer_status,
     BI          := Binary_time_value);
```

Запуск таймера с задержкой включения (On-delay timer)

На рисунке 7.7 показаны динамические характеристики таймера, запускаемого в режиме с задержкой включения (On-delay timer), и его реакция на сброс.

Показанное на рис. 7.7 поведение таймера будет соответствовать действительности, если Вы будете придерживаться рекомендованной выше последовательности операторов для STL (сначала запуск, затем сброс, затем опрос таймера). Обычно операция разблокирования (Enabling a timer) не требуется для работы таймера в программе STL, она также не является необходимой и для программы на SCL.

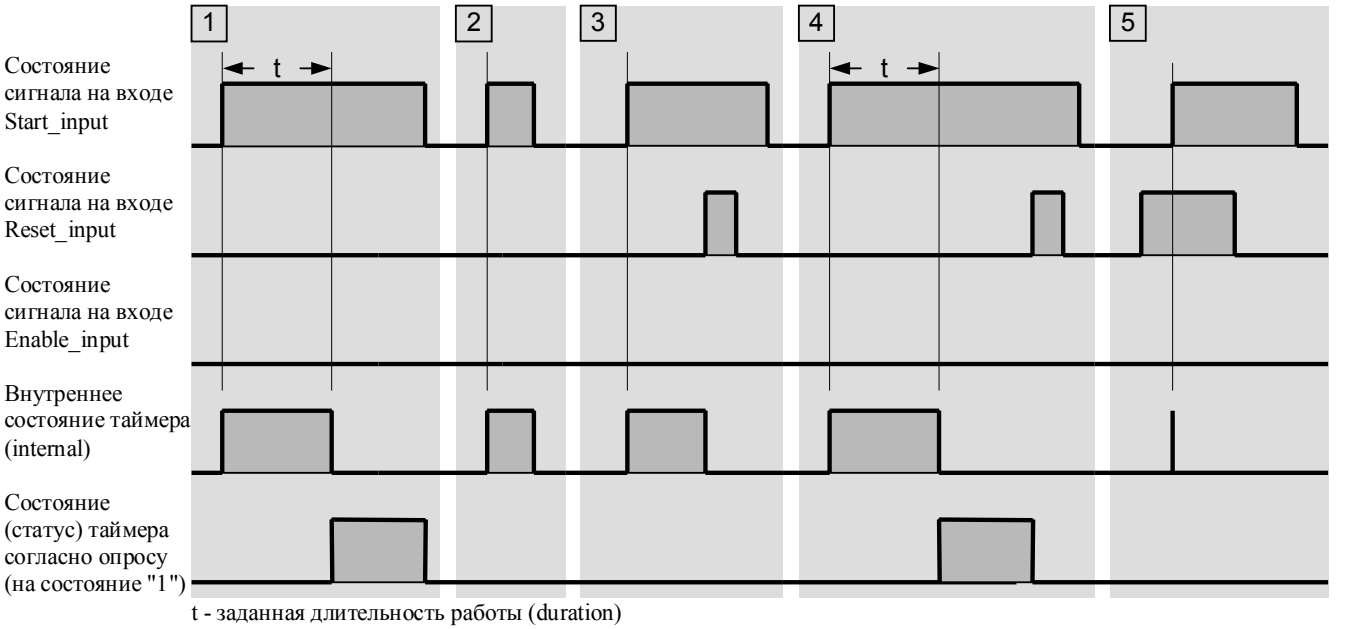


Рис. 7.7 Отклик таймера с задержкой включения (On-delay timer) на сигналы запуска и сброса

1 Функция таймера запускается, когда состояние сигнала на входе Start_input меняется от состояния "0" к состоянию "1" (положительный фронт). Таймер работает, до момента истечения заданного времени работы (длительности - "duration"). Проверки (опросы) таймера на состояние "1" (timer status) возвращают результат проверки "1", если до момента истечения заданного времени работы таймера (длительности - "duration") не сбрасывался в состояние "0" сигнал на входе Start_input, и пока состояние сигнала на входе Start_input остается равным "1", не появится сигнал, равный "1", на входе Reset_input, вызывающий сброс таймера.

Убывающее значение времени (как значение таймера) отсчитывается от некоторого заданного начального (initial) значения с заданным шагом, равным заданной величине "временной базы" (time base).

2 Функция таймера перестает быть активной, когда состояние сигнала на входе Start_input меняется от состояния "1" к состоянию "0" (отрицательный фронт), если даже это происходит до момента истечения заданного времени работы (длительности - "duration"). После этого проверки (опросы) таймера на состояние "1" (timer status) возвращают результат проверки "0". Значение времени (как значение таймера) показывает время, оставшееся до окончания заданного периода работы (длительности - "duration"), обозначая точку на временной оси, в которой произошло преждевременное прерывание работы таймера.

Сброс таймера с задержкой включения (Resetting an on-delay timer)

Операция сброса таймера с задержкой включения (Resetting an on-delay timer) имеет статический эффект и имеет приоритет перед запуском таймера (см. рис. 7.7).

3 4 Состояние сигнала на входе Reset_input, равное "1", вызывает сброс функции таймера, кончился ли отсчет заданного времени таймера или нет. После этого проверки (опросы) таймера на состояние "1" (timer status) возвращают результат проверки "0", даже если отсчет заданного времени таймера закончился и на входе Start_input присутствует состояние "1". Значение времени (time value) и значение "временной базы" (time base) также сбрасываются и становятся равными "0".
Отрицательный фронт сигнала (т.е. переход его от состояния "1" к "0") на входе Reset_input в то время, пока на входе Start_input присутствует состояние "1", никак не сказывается на режиме таймера.
Результат логической операции RLO, равный "1", на входе Reset_input также сбрасывает таймер, даже если отсчет заданного времени таймера закончился. После этого проверки (опросы) таймера на состояние "1" (timer status) возвращают результат проверки "0"

5 Если сигнал на входе Start_input изменяет свое состояние с "0" на "1" (положительный фронт сигнала), в то время, как на входе Reset_input присутствует состояние "1", то таймер запускается, но последующая инструкция сброса немедленно его сбрасывает (на рис. 7.7 это показано жирной вертикальной чертой). Если проверки (опросы) таймера на состояние "1" (timer status) следуют по времени после сброса таймера, то короткое время работы таймера после его запуска не скажется на результатах проверки - результат проверки будет равен "0".

Разблокировка таймера с задержкой включения (Anabling an on-delay timer)

Функция разблокировки таймера (Anabling an on-delay timer) позволяет вновь запускать отсчет времени (или перезапускать сначала отсчет времени уже активного таймера) посредством приложения ко входу Enabling_input положительного фронта сигнала (переход сигнала от состояния "0" к состоянию "1").

Функция разблокировки таймера (Anabling an on-delay timer) доступна для использования только в языке программирования STL.

На рисунке 7.8 показана функция разблокировки таймера, запускаемого в режиме с задержкой включения (On-delay timer).

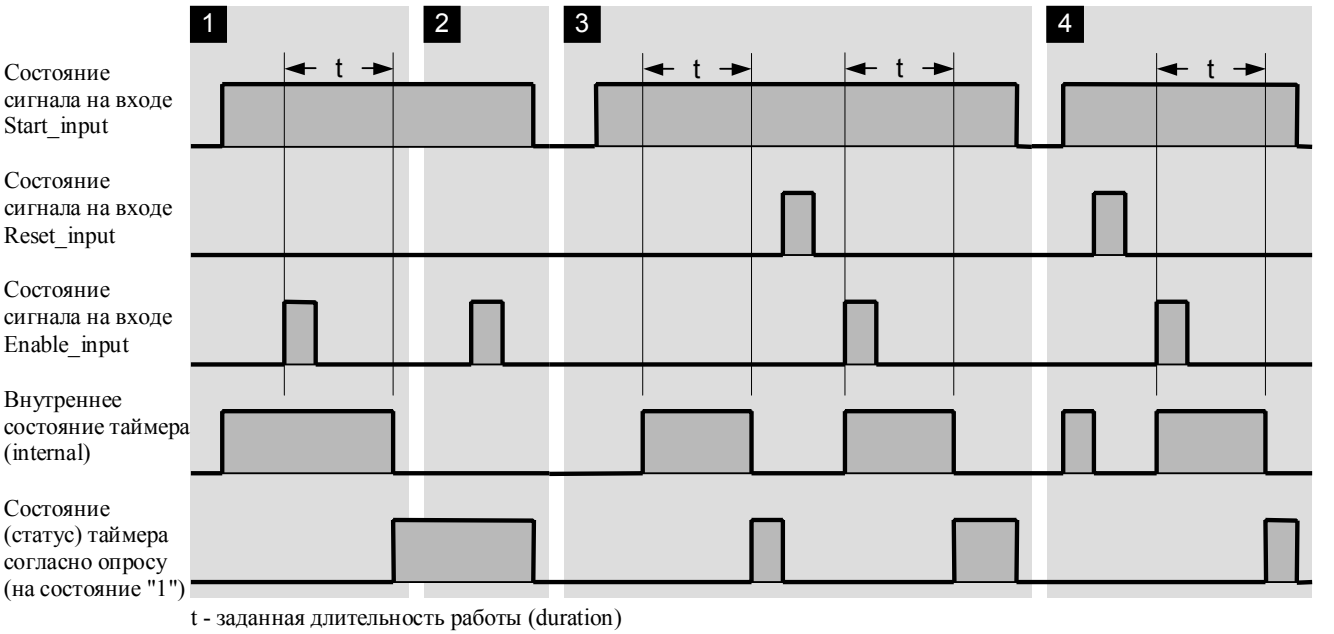


Рис. 7.8 Функция разблокировки (Enabling) таймера с задержкой включения (On-delay timer)

1 Если функция таймера активна (идет отсчет времени таймера) и состояние сигнала на входе Enable_input меняется от состояния "0" к состоянию "1" (положительный фронт), то отсчет времени таймера будет перезапущен сначала, если состояние сигнала на входе Start_input остается равным "1". При этом заданное значение длительности работы ("duration") будет вновь взято как исходное значение в момент перезапуска таймера. Последующее изменение сигнала на входе Enable_input от состояния "1" к состоянию "0" (отрицательный фронт) не изменяет режима таймера.

2 Если состояние сигнала на входе Enable_input меняется от состояния "0" к состоянию "1" (положительный фронт) после того, как отсчет заданного времени таймера закончился без прерывания, на режим работы таймера этот положительный фронт не окажет никакого влияния.

3 4 Если состояние сигнала на входе Start_input все еще остается равным "1", а функция таймера была деактивирована сигналом на входе сброса Reset_input, то в момент изменения сигнала на входе Enable_input от состояния "0" к состоянию "1" (положительный фронт) таймер будет вновь запущен в режиме с задержкой включения (On-delay timer). При этом заданное значение длительности работы ("duration") будет взято как исходное значение.

Если состояние сигнала на входе Start_input равно "0", то изменение сигнала на входе Enable_input от состояния "0" к состоянию "1" (положительный фронт) не будет иметь никакого эффекта.

7.5 Таймер с задержкой включения с памятью (Retentive On-delay timer)

Ниже представлен пример законченной программы на STL для запуска таймера с задержкой включения с памятью (Retentive On-delay timer):

```
A       Enable_input;
FR      Timer;
A       Start_input;
L       Duration;
SS      Timer;
A       Reset_input;
R       Timer;
L       Timer;
T       Binary_time_value;
LD      Timer;
T       BCD_time_value;
A       Timer;
=       Timer_status;
```

Программа на SCL для вызова таймера с задержкой включения с памятью (Retentive On-delay timer):

```
BCD_time_value   := S_ODTS (
     T_NO        := Timer,
     S           := Start_input,
     TV          := Duration,
     R           := Reset_input,
     Q           := Timer_status,
     BI          := Binary_time_value);
```

Запуск таймера с задержкой включения с памятью (Retentive On-delay timer)

На рисунке 7.9 показаны динамические характеристики таймера, запускаемого в режиме с задержкой включения с памятью (Retentive On-delay timer), и его реакция на сброс.

Показанное на рис. 7.9 поведение таймера будет соответствовать действительности, если Вы будете придерживаться рекомендованной выше последовательности операторов для STL (сначала запуск, затем сброс, затем опрос таймера). Обычно операция разблокирования (Enabling a timer) не требуется для работы таймера в программе STL, и для программы на SCL она также не является необходимой.

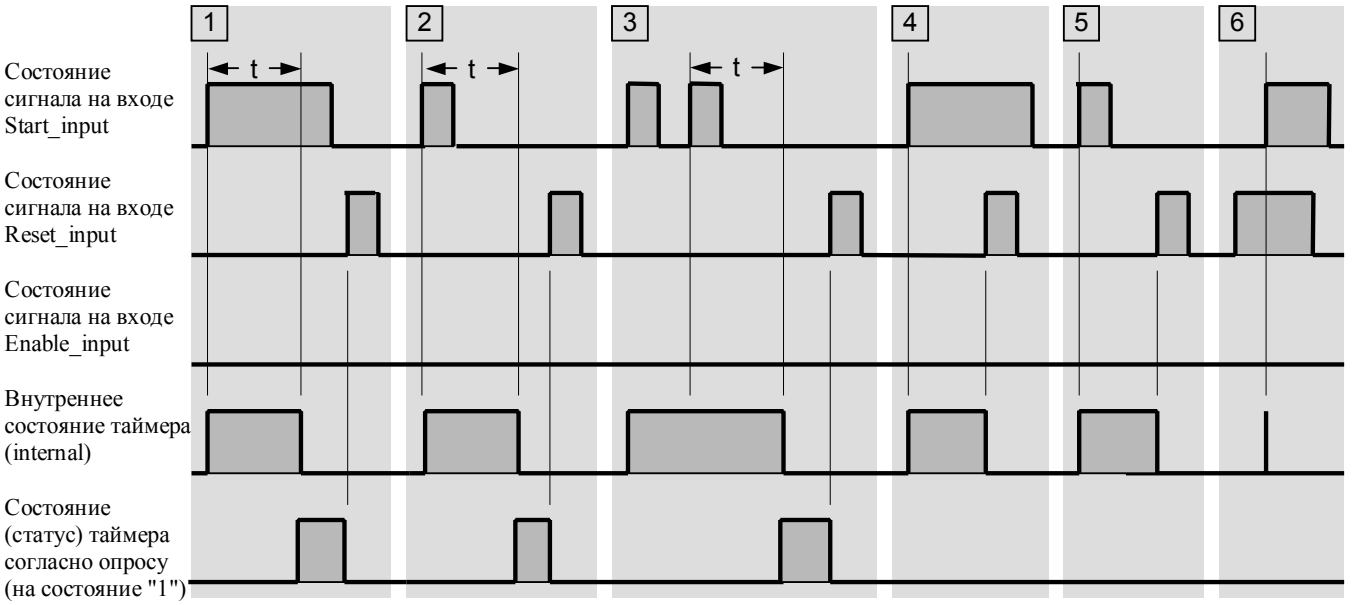


Рис. 7.9 Отклик таймера с задержкой включения с памятью (Retentive On-delay timer) на сигналы запуска и сброса

1 2 Функция таймера запускается, когда состояние сигнала на входе Start_input меняется от состояния "0" к состоянию "1" (положительный фронт). Отсчет времени таймера происходит до момента истечения заданного времени работы (длительности - "duration"), даже если до этого момента состояние сигнала на входе Start_input изменится снова от состояния "1" к состоянию "0". Если отсчет времени завершился без прерывания, то в дальнейшем проверки (опросы) таймера на состояние "1" (timer status) возвращают результат проверки "1" вне зависимости от состояния сигнала на входе Start_input. При этом проверки (опросы) таймера на состояние "1" (timer status) будут возвращать результат проверки "1" до момента сброса таймера вне зависимости от состояния сигнала на входе Start_input.

Убывающее значение времени (как значение таймера) отсчитывается от некоторого заданного начального (initial) значения с заданным шагом, равным заданной величине "временной базы" (time base).

3 Если состояние сигнала на входе Start_input меняется от состояния "0" к состоянию "1" (положительный фронт), в то время, пока функция таймера активна, то таймер будет перезапущен. При этом заданное значение длительности работы ("duration") будет вновь взято как исходное значение в момент перезапуска таймера.

Таким образом таймер может быть перезапущен столько раз, сколько требуется для нормальной работы программы, невзирая на время отсчета, оставшееся в предыдущем рабочем периоде таймера.

Сброс таймера с задержкой включения с памятью (Retentive On-delay timer)

Операция сброса таймера с задержкой включения с памятью (Retentive On-delay timer) имеет статический эффект и имеет приоритет перед запуском таймера (см. рис. 7.9).

4 5 Состояние сигнала на входе Reset_input, равное "1", вызывает сброс таймера, если он был до этого активен и вне зависимости от состояния сигнала на входе Start_input. После этого проверки (опросы) таймера на состояние "1" (timer status) возвращают результат проверки "0". Значение времени (time value) и значение "временной базы" (time base) также сбрасываются и становятся равными "0".

6 Если сигнал на входе Start_input изменяет свое состояние с "0" на "1" (положительный фронт сигнала), в то время, как на входе Reset_input присутствует состояние "1", то таймер запускается, но последующая инструкция сброса немедленно его сбрасывает (на рис. 7.9 это показано жирной вертикальной чертой). Если проверки (опросы) таймера на состояние "1" (timer status) следуют по времени после сброса таймера, то короткое время работы таймера после его запуска не скажется на результатах проверки - результат проверки будет равен "0".

Разблокировка таймера с задержкой включения с памятью (Anabling a retentive on-delay timer)

Функция разблокировки таймера (Anabling a retentive on-delay timer) позволяет вновь запускать отсчет времени (или перезапускать сначала отсчет времени уже активного таймера) посредством приложения ко входу Enabling_input положительного фронта сигнала (переход сигнала от состояния "0" к состоянию "1").

Функция разблокировки таймера (Anabling a retentive on-delay timer) доступна для использования только в языке программирования STL.

На рисунке 7.10 показана функция разблокировки таймера, запускаемого в режиме с задержкой включения с памятью (Retentive On-delay timer).

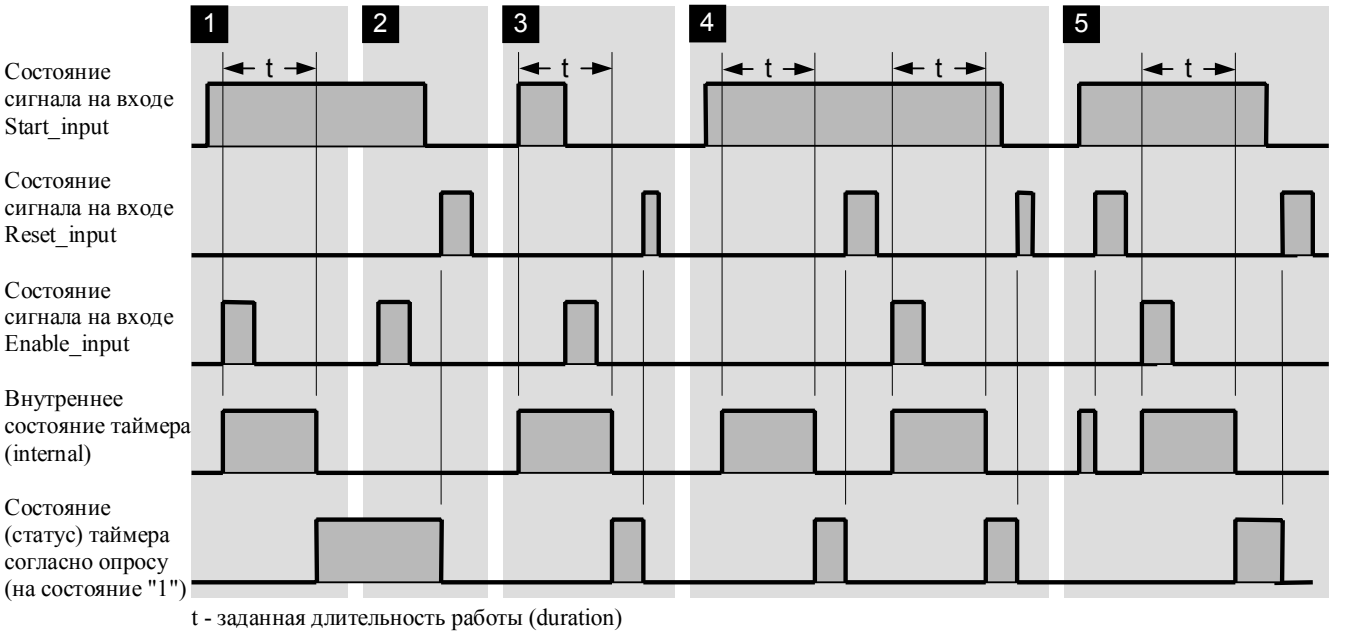


Рис. 7.10 Функция разблокировки (Enabling) таймера с задержкой включения с памятью (Retentive On-delay timer)

1 Если функция таймера активна (идет отсчет времени таймера) и состояние сигнала на входе Enable_input меняется от состояния "0" к состоянию "1" (положительный фронт), то отсчет времени таймера будет перезапущен сначала, если состояние сигнала на входе Start_input остается равным "1". При этом заданное значение длительности работы ("duration") будет вновь взято как исходное значение в момент перезапуска таймера. Последующее изменение сигнала на входе Enable_input от состояния "1" к состоянию "0" (отрицательный фронт) не изменяет режима таймера.

2 Если состояние сигнала на входе Enable_input меняется от состояния "0" к состоянию "1" (положительный фронт) после того, как отсчет заданного времени таймера закончился без прерывания, на режим работы таймера этот положительный фронт не окажет никакого влияния.

3 Если состояние сигнала на входе Start_input равно "0", то изменение сигнала на входе Enable_input от состояния "0" к состоянию "1" (положительный фронт) не будет иметь никакого эффекта.

4 **5** Если состояние сигнала на входе Start_input все еще остается равным "1", а функция таймера была деактивирована сигналом на входе сброса Reset_input, то в момент изменения сигнала на входе Enable_input от состояния "0" к состоянию "1" (положительный фронт) таймер будет вновь запущен в режиме с задержкой включения с памятью (Retentive on-delay timer). При этом заданное значение длительности работы ("duration") будет взято как исходное значение.

7.6 Таймер с задержкой выключения (Off-delay timer)

Ниже представлен пример законченной программы на STL для запуска таймера с задержкой выключения (Off-delay timer):

```
A       Enable_input;
FR      Timer;
A       Start_input;
L       Duration;
SF      Timer;
A       Reset_input;
R       Timer;
L       Timer;
T       Binary_time_value;
LD      Timer;
T       BCD_time_value;
A       Timer;
=       Timer_status;
```

Программа на SCL для вызова таймера с задержкой выключения (Off-delay timer):

```
BCD_time_value   := S_OFFDT (
     T_NO        := Timer,
     S           := Start_input,
     TV          := Duration,
     R           := Reset_input,
     Q           := Timer_status,
     BI          := Binary_time_value);
```

Запуск таймера с задержкой выключения (Off-delay timer)

На рисунке 7.11 показаны динамические характеристики таймера, запускаемого в режиме с задержкой выключения (Off-delay timer), и его реакция на сброс.

Показанное на рис. 7.11 поведение таймера будет соответствовать действительности, если Вы будете придерживаться рекомендованной выше последовательности операторов для STL (сначала запуск, затем сброс, затем опрос таймера). Обычно операция разблокирования (Enabling a timer) не требуется для работы таймера в программе STL, и для программы на SCL она также не является необходимой.

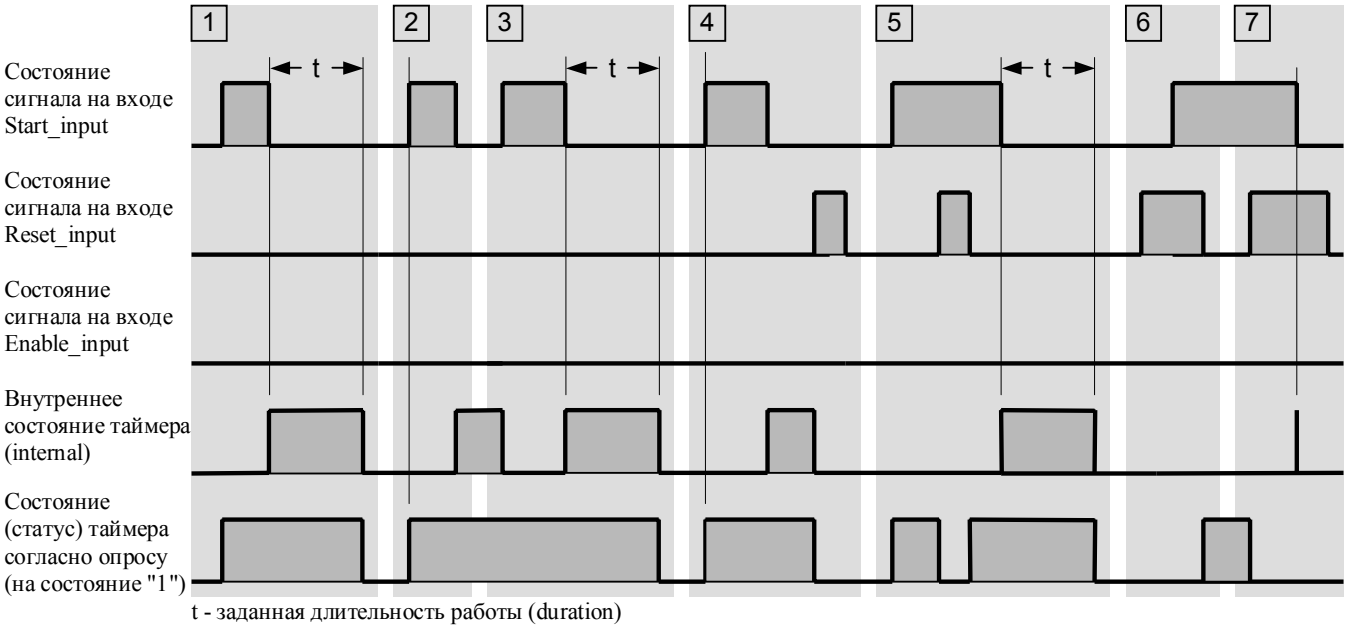


Рис. 7.11 Отклик таймера с задержкой выключения (Off-delay timer) на сигналы запуска и сброса

1 2 Функция таймера запускается, когда состояние сигнала на входе Start_input меняется от состояния "1" к состоянию "0" (отрицательный фронт). Таймер работает, до момента истечения заданного времени работы (длительности - "duration"). Проверки (опросы) таймера на состояние "1" (timer status) возвращают результат проверки "1", если до момента истечения заданного времени работы таймера (длительности - "duration") на входе Reset_input не появлялся сигнал "1", вызывающий сброс таймера, и если состояние сигнала на входе Start_input остается равным "1".

Убывающее значение времени (как значение таймера) отсчитывается от некоторого заданного начального (initial) значения с заданным шагом, равным заданной величине "временной базы" (time base).

3 Функция таймера перестает быть активной, когда состояние сигнала на входе Start_input меняется от состояния "0" к состоянию "1" (положительный фронт), если это происходит до момента истечения заданного времени работы (длительности - "duration"). При этом таймер сбрасывается, и после этого проверки (опросы) таймера на состояние "1" (timer status) возвращают результат проверки "0". Таймер не запускается до появления нового отрицательного фронта сигнала на входе Start_input.

Сброс таймера с задержкой выключения (Resetting an off-delay timer)

Операция сброса таймера с задержкой выключения (Resetting an off-delay timer) имеет статический эффект и имеет приоритет перед запуском таймера (см. рис. 7.11).

4 Состояние сигнала на входе Reset_input, равное "1", вызывает сброс функции таймера. После этого проверки (опросы) таймера на состояние "1" (timer status) возвращают результат проверки "0". Значение времени (time value) и значение "временной базы" (time base) также сбрасываются и становятся равными "0".

5 6 Если сигнал на входе Reset_input изменяет свое состояние с "0" на "1" (положительный фронт сигнала), в то время, как на входе Start_input присутствует состояние "1", то выход таймера сбрасывается (проверки [опросы] таймера на состояние "1" [timer status] после сброса выхода таймера, дают результат проверки, равный "0").

Отрицательный фронт сигнала (т.е. переход его от состояния "1" к "0") на входе Reset_input в то время, пока на входе Start_input присутствует состояние "1", возвращает выход таймера в состояние "1".

Если сигнал на входе Start_input изменяет свое состояние с "1" на "0" (отрицательный фронт сигнала), в то время, как на входе Reset_input присутствует состояние "1", то таймер запускается, но последующая инструкция сброса немедленно его сбрасывает (на рис. 7.11 это показано жирной вертикальной чертой). Если проверки (опросы) таймера на состояние "1" (timer status) следуют по времени после сброса таймера, то короткое время работы таймера после его запуска не скажется на результатах проверки - результат проверки будет равен "0".

Разблокировка таймера с задержкой выключения (Anabling an off-delay timer)

Функция разблокировки таймера (Anabling an off-delay timer) позволяет вновь запускать отсчет времени (или перезапускать сначала отсчет времени уже активного таймера) посредством приложения ко входу Enabling_input положительного фронта сигнала (переход сигнала от состояния "0" к состоянию "1").

Функция разблокировки таймера (Anabling an off-delay timer) доступна для использования только в языке программирования STL.

На рисунке 7.12 показана функция разблокировки таймера, запускаемого в режиме с задержкой выключения (Off-delay timer).

1 Если функция таймера неактивна (нет отсчета времени таймера), состояние сигнала на входе Start_input остается равным "1", а состояние сигнала на входе Enable_input меняется от состояния "0" к состоянию "1" (положительный фронт), то на режим работы таймера ни этот положительный фронт, ни последующий отрицательный фронт сигнала на входе Enable_input не окажут никакого влияния.

2 Если состояние сигнала на входе Enable_input меняется от состояния "0" к состоянию "1" (положительный фронт) в то время, когда функция таймера активна (запущен отсчет заданного времени таймера), отсчет времени таймера будет перезапущен сначала. При этом заданное значение длительности работы ("duration") будет вновь взято как исходное значение в момент перезапуска таймера.

3 Если состояние сигнала на входе Enable_input меняется от состояния "0" к состоянию "1" (положительный фронт) или от состояния "1" к состоянию "0" (отрицательный фронт) в то время, когда функция таймера неактивна (нет отсчета времени таймера), то на режим работы таймера этот положительный или отрицательный фронт сигнала не окажут никакого влияния.

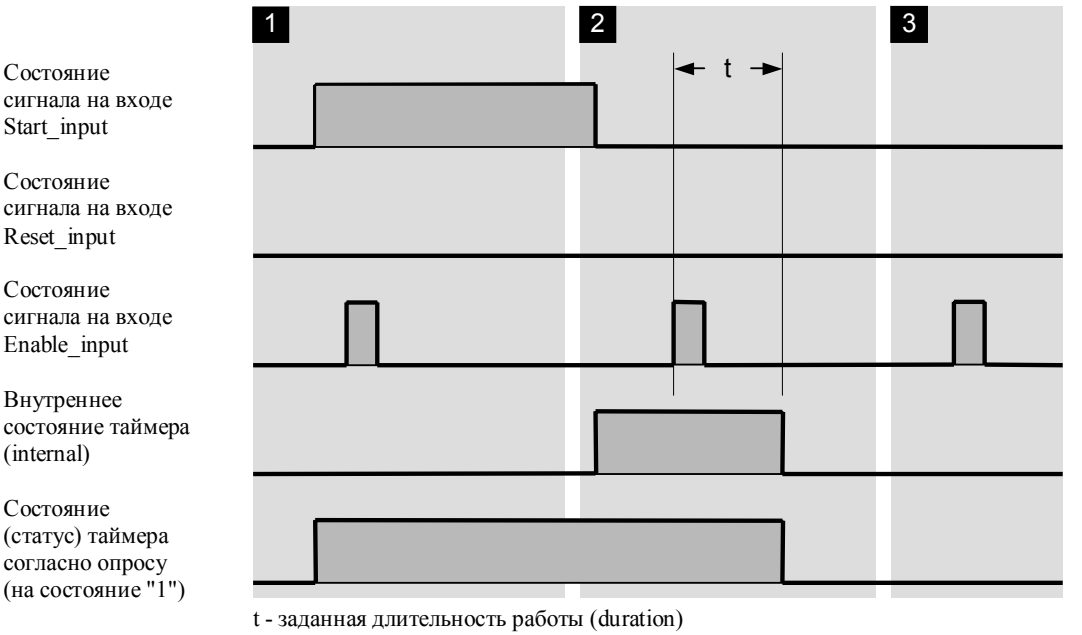


Рис. 7.12 Функция разблокировки (Enabling) таймера с задержкой выключения (Off-delay timer)

7.7 IEC-функции таймеров (IEC Timer Functions)

IEC-функции таймеров (IEC Timer Functions) встроены в операционную систему CPU как системные функциональные блоки SFB.

В соответствующим образом оснащенных CPU могут быть доступны следующие функции таймеров:

- SFB 3 TP

  Генератор импульсов
- SFB 4 TON

  Генерация импульса с задержкой включения
- SFB 5 TOF

  Генерация импульса с задержкой выключения

На рис. 7.13 представлены динамические характеристики этих таймеров.

Вы можете вызывать эти SFB с экземплярными блоками данных или использовать эти SFB как локльные экземпляры в функциональном блоке.

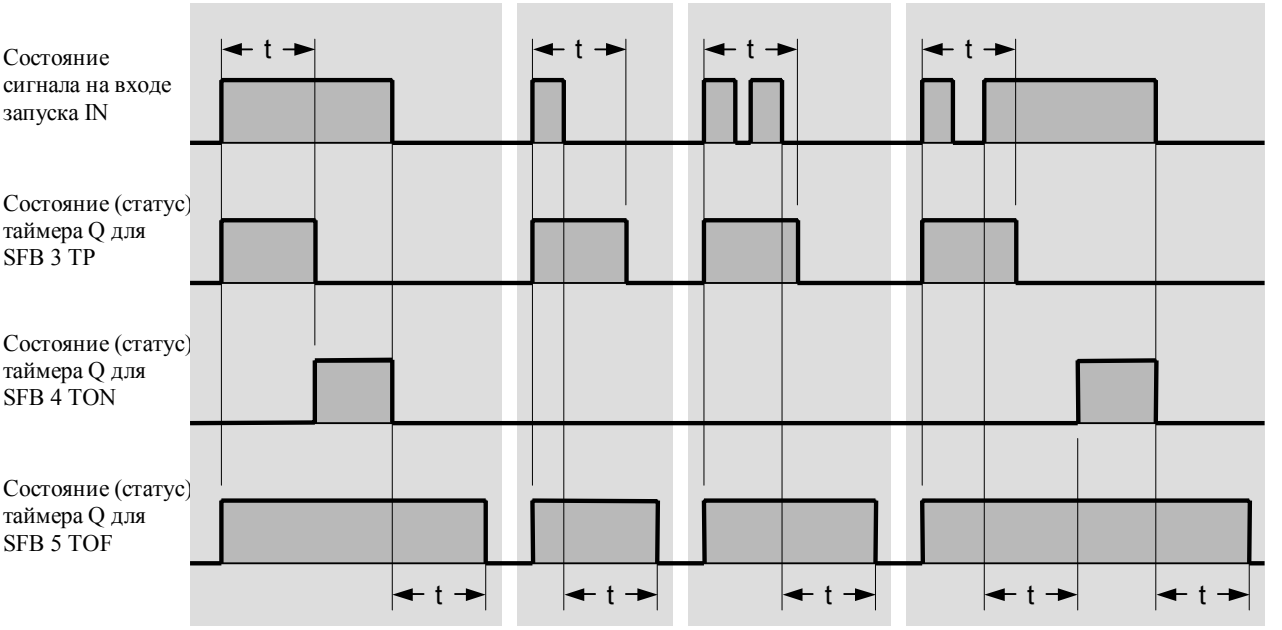


Рис. 7.13 Динамические характеристики IEC-функций таймеров

Вы можете найти описание интерфейса функций для программирования в автономном режиме (offline) в "стандартной библиотеке" *Standard Library* в *System Function Blocks*.

Примеры вызовов этих функций находятся на прилагаемой дискете в библиотеке STL_Book в программе "Basic Functions" в функциональном блоке FB 107 или в исходном файле Chap_7 или в библиотеке SCL_Book в программе "30 SCL Functions".

Таблица 7.2 Параметры IEC-функций таймеров

Название (Name)	Объявление (Declaration)	Тип (Data type)	Описание (Description)
IN	INPUT	BOOL	Вход запуска (Start input)
PT	INPUT	TIME	Длина импульса (Pulse length) или продолжительность задержки включения (Delay duration)
Q	INPUT	BOOL	Состояние таймера (Timer status)
ET	INPUT	TIME	Истекшее время (Elapsed time)

7.7.1 Генератор импульсов SFB 3 TP

Параметры IEC-таймера SFB 3 TP показаны в таблице 7.2.

Если RLO на входе запуска IN таймера изменяет свое состояние с "0" на "1", то функция таймера активизируется, т.е. запускается отсчет времени в течение заданного периода времени (duration), который не зависит от изменений состояния RLO на входе запуска. При этом выход Q возвращает сигнал "1" все то время, пока идет отсчет времени.

Выход ET возвращает длительность времени (duration), в течение которого выход Q находится в установленном состоянии. Этот период времени (duration) начинается в момент T#0s и заканчивается в заданное пользователем время PT. Если время PT истекло, то ET сохраняет это значение, пока состояние сигнала на входе IN не станет равным "0". Если состояние сигнала на входе IN станет равным "0" до того, как истекло заданное значение времени PT (duration), то на выходе ET значение изменится на T#0s сразу же после того, как закончится отсчет времени PT.

Если необходимо снова инициализировать функцию таймера, запустите ее с заданной величиной PT = T#0s.

IEC-таймер SFB 3 TP работает в рабочих режимах RESTART и RUN. IEC-таймер SFB 3 TP сбрасывается (инициализируется) при холодном перезапуске.

7.7.2 Генератор импульсов с задержкой включения SFB 4 TON

Параметры IEC-таймера с задержкой включения SFB 4 TON показаны в таблице 7.2

Если RLO на входе запуска IN таймера изменяет свое состояние с "0" на "1", то функция таймера активизируется, т.е. запускается отсчет времени в течение заданного периода времени (duration). При этом выход Q возвращает сигнал "1" после того, как отсчет времени завершился без досрочного прерывания. Если до истечения заданного периода времени (duration) RLO на входе запуска изменяет свое состояние с "1" на "0", то функция таймера сбрасывается. Отсчет времени запускается снова, если на входе запуска вновь появляется положительный фронт сигнала.

Выход ET возвращает длительность времени (duration), в течение которого функция таймера активна. Этот период времени (duration) начинается в момент T#0s и заканчивается в заданное пользователем время PT. Если время PT истекло, то ET сохраняет это значение, пока состояние сигнала на входе IN вновь не станет равным "0". Если состояние сигнала на входе IN станет равным "0" до того, как истекло заданное значение времени PT (duration), то на выходе ET значение изменится на T#0s немедленно.

Если необходимо снова инициализировать функцию таймера, запустите ее с заданной величиной PT = T#0s.

IEC-таймер SFB 4 TON работает в рабочих режимах RESTART и RUN. IEC-таймер SFB 4 TON сбрасывается (инициализируется) при холодном перезапуске.

7.7.3 Генератор импульсов с задержкой выключения SFB 5 TOF

Параметры IEC-таймера с задержкой выключения SFB 5 TOF показаны в таблице 7.2

Как только RLO на входе запуска IN таймера изменяет свое состояние с "0" на "1", при проверке выход таймера Q возвращает сигнал "1". После того как RLO на входе запуска изменяет свое состояние с "1" на "0", функция таймера активизируется, т.е. запускается отсчет времени в течение заданного периода времени (duration). При этом выход Q возвращает сигнал, равный "1", пока отсчет времени не завершится без досрочного прерывания. Если до истечения заданного периода времени (duration) RLO на входе запуска опять изменяет свое состояние с "0" на "1", то функция таймера сбрасывается, при этом выход Q сохраняет значение сигнала, равное "1". Отсчет времени запускается снова, если на входе запуска вновь появляется отрицательный фронт сигнала.

Выход ET возвращает длительность времени (duration), в течение которого функция таймера активна. Этот период времени (duration) начинается в момент T#0s и заканчивается в заданное пользователем время PT. Если время PT истекло, то ET сохраняет это значение, пока состояние сигнала на входе IN вновь не станет равным "1". Если состояние сигнала на входе IN станет равным "1" до того, как истекло заданное значение времени PT (duration), то на выходе ET значение изменится на T#0s немедленно.

Если необходимо снова инициализировать функцию таймера, запустите ее с заданной величиной PT = T#0s.

IEC-таймер SFB 5 TOF работает в рабочих режимах RESTART и RUN. IEC-таймер SFB 5 TOF сбрасывается (инициализируется) при холодном перезапуске.

8 Функции счетчиков (Counter Functions)

Функции счетчиков позволяют решать задачи счета непосредственно в CPU. Счетчики позволяют выполнять прямой и обратный счет и используют при этом "трехдекадный" формат значения счетчика и диапазон значений от 000 до 999.

В данной главе рассматриваются выражения, содержащие функции счетчиков для использования в языке программирования STL. Для языка SCL функции счетчиков включены в состав стандартных функций (см. разд. 30.2 "Функции счетчиков" ["Counter Functions"]).

Скорость счета счетчиков зависит от времени сканирования Вашей программы! Для обеспечения процесса счета CPU должен обнаруживать на входе счетчика изменения входного сигнала, иначе говоря, входной импульс (или междуимпульсная пауза) на входе должна присутствовать по крайней мере один цикл сканирования программы. Чем продолжительнее цикл сканирования программы, тем медленнее скорость счета счетчика.

Примечание:

В S7-300 CPU со встроенными функциями (CPU 3xxIFM) имеются встроенные функции счетчика, которые обеспечивают счет с использованием специального входа счетчика с частотой следования импульсов до 10 кГц.

Значения счетчиков, описанных в данной главе, сохраняются в системной памяти CPU. Вы можете задавать для счетчика начальное значение (initial value). Вы также можете сбрасывать счетчик, включать режим прямого или обратного счета счетчика. Существует возможность определения состояния счетчика (содержит ли счетчик нулевое или ненулевое значение). Функции загрузки (load) используются для пересылки текущего значения счетчика в двоичном или BCD-коде в аккумулятор accumulator 1.

Примеры, рассматриваемые в данной главе, и вызовы IEC-счетчиков Вы можете найти на прилагаемой дискете в библиотеке STL_Book library в разделе "Basic Functions" в функциональном блоке FB 108 или в исходном файле Chap_8.

8.1 Установка и сброс счетчиков

Установка счетчика (Setting a counter)

S C n Установка счетчика

Счетчик устанавливается, если RLO переходит от "0" к "1" перед операцией установки S счетчика (Set). Т.е. для установки счетчика всегда требуется положительный фронт.

Установить счетчик - это значит загрузить в счетчик начальное значение.

Начальное значение, которое загружается в счетчик, должно находиться в аккумуляторе accumulator 1 (см. ниже). Оно должно быть в диапазоне от 0 до 999.

Рис. 8.1 Назначение битов для значения счетчика ("counter value")

Спецификация счетчика (Specifying a counter)

Функция установки счетчика ("set counter") использует значение в аккумуляторе accumulator 1 как "значение счетчика" ("count value"). Как и когда это значение появляется в аккумуляторе accumulator 1 не имеет значения.

Для обеспечения лучшей читаемости программы наилучшим будет вариант, когда эти параметры загружаются в аккумулятор accumulator 1 непосредственно перед запуском функции счетчика или в виде константы (т.е., в виде непосредственно заданной величины), или в виде переменной (например, через слово в памяти, содержащее значение счетчика).

Примечание:

аккумулятор accumulator 1 должен содержать корректное значение, даже если счетчик не установлен во время выполнения инструкции.

Определение (спецификация) счетчика с помощью константы

```
L     C#100;              //Значение счетчика 100
L     W#16#0100;          //Значение счетчика 100
```

Значения счетчика лежат в диапазоне 000 … 999. При этом используются три декады. Значение счетчика может быть только в формате BCD. При этом счетчики не могут работать с отрицательными числами.

Для задания константы Вы можете использовать C# или W#16# (и только с десятичными числами).

Определение (спецификация) счетчика с помощью переменной

```
L     C#200;          //Значение счетчика 200
T     MW 56;          //Сохранить значение счетчика
…     ;
L     MW 56;          //Загрузить значение счетчика
```

Для выполнения операции Set предполагается наличие значения счетчика в аккумуляторе accumulator 1, состоящего из трех декад и расположенного в нем с выравниванием вправо. Назначение битов в счетчике (тип C#) подробно описано в главе 24 "Типы данных".

Сброс счетчика (Resetting a counter)

R C n Сброс счетчика

Счетчик сбрасывается, если RLO имеет значение "1" перед тем, как в программе встретится операция сброса R счетчика (Reset). Пока RLO равен "1", проверки счетчика на состояние "1" возвращают результат проверки "0"; проверки счетчика на состояние "0" возвращают результат проверки "1". Сброс устанавливает для значения счетчика ("count value") нулевое значение (0).

Примечание:

Сброс функции счетчика не сбрасывает внутренний меркер фронта для установки счетчика, для включения режима прямого счета и для включения режима обратного счета. Для повторного запуска CPU должен обработать инструкцию установки счетчика или повторного запуска на счет при RLO, равном "0", и только после этого при появлении фронта сигнала в соответствующем меркере фронта счетчик может быть установлен или запущен. Также для повторной активации функций счетчика Вы можете использовать операцию разблокировки счетчика.

8.2 Счет (Counting)

Прямой счет (Counting up)

Инструкция

CU C n вызывает процесс прямого счета.

Счетчик выполняет прямой счет ("инкрементируется"), если инструкция CU обрабатывается при положительном (возрастающем) фронте сигнала RLO (RLO меняет свое состояние с "0" на "1").

Каждый положительный фронт сигнала, предшествующий операции CU, увеличивает значение счетчика ("count value") на единицу, пока не будет достигнут верхний предел, равный 999. После этого положительный фронт сигнала RLO перед вводом CU никак не будет влиять на состояние счетчика.

Обратный счет (Counting Down)

Инструкция

CD	C n	вызывает процесс обратного счета.

Счетчик выполняет обратный счет ("декрементируется"), если инструкция CD обрабатывается при положительном (возрастающем) фронте сигнала RLO (RLO меняет свое состояние с "0" на "1").

Каждый положительный фронт сигнала, предшествующий операции CD, уменьшает значение счетчика ("count value") на единицу, пока не будет достигнут нижний предел, равный 0. После этого положительный фронт сигнала RLO перед вводом CD никак не будет влиять на состояние счетчика.

Значения счетчика отрицательными быть не могут.

8.3 Проверка (опрос) счетчика (Checking a Counter)

Проверка (опрос) состояния счетчика (binary counter check)

A	C n	проверка сигнала на состояние "1" и комбинирование с RLO в соответствии с логикой AND (И).
O	C n	проверка сигнала на состояние "1" и комбинирование с RLO в соответствии с логикой OR (ИЛИ).
X	C n	проверка сигнала на состояние "1" и комбинирование с RLO в соответствии с логикой Exclusive OR (Исключающее ИЛИ).
AN	C n	проверка сигнала на состояние "0" и комбинирование с RLO в соответствии с логикой AND (И).
ON	C n	проверка сигнала на состояние "0" и комбинирование с RLO в соответствии с логикой OR (ИЛИ).
XN	C n	проверка сигнала на состояние "0" и комбинирование с RLO в соответствии с логикой Exclusive OR (Исключающее ИЛИ).

Вы можете опрашивать счетчик, как если бы это был, например, вход (input), и в дальнейшем использовать результат этой проверки. Проверка (опрос) сигнала на состояние "1" вызывает результат "1", если значение счетчика больше 0 и результат, равный "0", если значение счетчика равно 0.

Непосредственная загрузка значения счетчика (direct loading of a count value)

Инструкция

L C n непосредственно загружает двоичное значение счетчика ("count value").

Функция загрузки L C пересылает определенное значение ("count value") счетчика, определенного в инструкции, в аккумулятор accumulator 1 в форме двоичного числа. Значение счетчика, пересылаемое в аккумулятор accumulator 1, представляет собой текущее значение, соответствующее моменту времени, когда производится опрос. Теперь значение в аккумуляторе accumulator 1 соответствует положительному числу целого типа (INT) и может использоваться для дальнейшей обработки, например, с помощью арифметических функций.

Пример:

```
L   C 99;     //загрузка текущего значения счетчика
T   MW 76;    //сохранение текущего значения счетчика
```

Загрузка значения счетчика в BCD-формате ("coded loading")

Инструкция

LD C n загружает двоично-десятичное ("кодированное") значение счетчика ("count value").

Вы можете также использовать инструкцию для т.н. "кодированной" пересылки ("coded load") в аккумулятор accumulator 1 в формате двоично-десятичного числа значения счетчика, определенного в инструкции. Значение счетчика, пересылаемое в аккумулятор accumulator 1, представляет собой текущее значение, соответствующее моменту времени, когда производится опрос. Содержимое аккумулятора accumulator 1 доступно для дальнейшего использования в виде числа в формате BCD-числа, выровненного вправо. Оно имеет такую же структуру как и заданное значение счетчика.

Пример:

```
LD  C 99;     //загрузка текущего значения счетчика в BCD-формате
T   MW 50;    //сохранение текущего значения счетчика
```

8.4 Разблокировка счетчика (Enabling a counter)

Инструкция

FR C n позволяет выполнить переустановку (перезапуск) счетчика.

При использовании инструкции FR Вы можете установить (Set) счетчик или запустить на счет без предшествующего операции положительного фронта сигнала RLO. Тем не менее, выполнение этих операций со счетчиком будет возможно только, пока RLO имеет значение "1".

Функция разблокировки активна, если перед тем, как она встретится, RLO переходит от состояния "0" к состоянию "1". Положительный фронт сигнала RLO - это всегда необходимое условие для выполнения разблокировки счетчика.

Данная инструкция разблокировки функции счетчика не требуется для установки, запуска на счет или сброса счетчика (т.е., инструкция не является необходимой для обычных условий работы со счетчиком).

Примечание:

Инструкция разблокировки функций счетчика воздействует на функции установки, запуска на счет и сброса счетчика одновременно! Положительный фронт сигнала RLO перед выполнением разблокировки счетчика вызывает все последующие инструкции (S, CU и CD), которые имеют сигнал запуска "1".

Ниже представлен пример, показывающий принципы работы инструкции разблокировки (соответствующие диаграммы показаны на рис. 8.2):

```
A       "Enable";
FR      "Counter";
A       "Count up";
CU      "Counter";
A       "Count down";
CD      "Counter";
A       "Set";
L       C#020;
S       "Counter";
A       "Reset";
R       "Counter";
A       "Counter";
=       "Counter status";
```

Ниже рассмотрены фазы соответствующих диаграмм, показанных на рис. 8.2:

1 Положительный фронт сигнала на входе установки счетчика (Set) устанавливает счетчик на начальное значение 20.

2 Положительный фронт сигнала на входе CU инкрементирует счетчик (его значение увеличивается на 1).

3 Так как сигнал на входе Set имеет значение "1", инструкция разблокирования счетчика инкрементирует счетчик - его значение увеличивается на 1.

4 Положительный фронт сигнала на входе сброса (Reset) декрементирует счетчик (его значение уменьшается на 1).

5 Инструкция разблокирования счетчика создает условий для выполнения прямого и обратного счета: сигнал "1" присутствует на обоих входах.

6 Положительный фронт сигнала на входе установки счетчика (Set) устанавливает счетчик на начальное значение 20.

7 Сигнал на входе Reset, имеющий значение "1", сбрасывает счетчик. Проверка его на состояние "1" возвращает результат "0".

8 Так как состояние сигнала на входе Set все еще равно "1", инструкция разблокирования устанавливает счетчик с начальным значением 20. Проверка счетчика на состояние "1" возвращает результат "1".

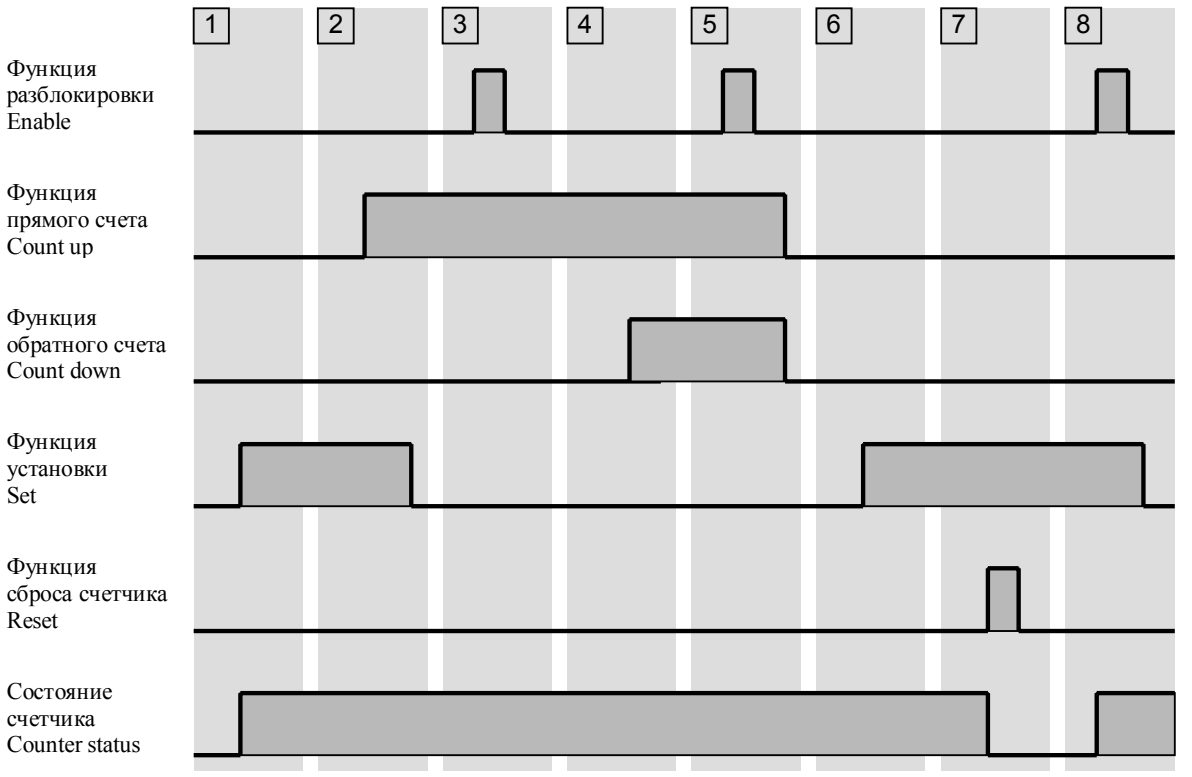


Рис. 8.2 Принцип работы инструкции разблокировки Enable

8.5 Последовательность инструкций при использовании функций счетчика

При программировании счетчика Вам нет необходимости использовать все возможные выражения для активации функций счетчиков. Вы должны использовать только те из функций, которые Вам необходимо выполнить. Например, для выполнения функции обратного счета обычно используются операция установки для счетчика заданного значения (initial value), операция обратного счета и двоичный опрос счетчика на состояние "0".

Чтобы выполнить функцию счетчика в соответствии с описанием в предыдущих разделах необходимо соблюдать определенный порядок при программировании соответствующих операторов.

В таблице 8.1 показан оптимальный порядок для всех операторов при программировании функций счетчика. Вы можете просто пропустить ненужные операторы, когда Вы будете записывать программу, ориентируясь на рекомендуемую последовательность операторов, например, пропустите функцию разблокировки Enable.

Таблица 8.1 Последовательность операторов для счетчика

Функция счетчика:	Примеры:
Разблокировка счетчика (Enable counter)	`A  I  22.0` `FR C  17`
Функция прямого счета (Count up)	`A  I  22.1` `CU C  17`
Функция обратного счета (Count down)	`A  I  22.2` `CD C  17`

Установка счетчика (Set counter)	`A  I  22.3`
	`L  C#500`
	`S  C  17`

Сброс счетчика (Reset counter)	`A  I  22.4` `R  C  17`

Проверка численного значения счетчика (Digital counter check)	`L  C  17`
	`T  MW 30`
	`LC C  17`
	`T  MW 32`

Двоичный опрос счетчика (Binary counter check)	`A  C  17` `=  Q  13.0`

Если функция сброса счетчика Reset должна иметь статическое ("static") влияние на операции CU, CD и S и не должна зависеть от результата логической операции (RLO), то Вы должны записать выражение с операцией Reset для счетчика после выражений с вышеуказанными операциями, но перед операцией проверки счетчика.

Если при этом счетчик устанавливается (set) и сбрасывается (reset) "одновременно", то счетчик сначала получит заданное значение, а затем немедленно будет сброшен операцией Reset. Таким образом, последующая проверка счетчика не позволит установить тот факт, что счетчик кратковременно находился в установленном состоянии.

Если функция установки счетчика Set должна иметь статическое ("static") влияние на операции счета и не должна зависеть от результата логической операции (RLO), то Вы должны записать выражение с операцией Set после выражений с операциями счета.

Если при этом счетчик устанавливается (set) и сбрасывается (reset) "одновременно", то операции счета вначале будут изменять состояние счетчика, а затем счетчик немедленно будет установлен операцией Set и получит заданное значение, которое и сохранится в нем до окончания сканирования программы.

Таким образом, последовательность выражений с операциями прямого и обратного счета не будет иметь влияние на счетчик.

8.6 IEC-функции счетчиков (IEC Counter Functions)

IEC-функции счетчиков (IEC Counter Functions) встроены в операционную систему CPU как системные функциональные блоки SFB.

В соответствующим образом оснащенных CPU могут быть доступны следующие функции счетчиков:

- SFB 0 CTU

  Функция прямого счета
- SFB 1 CTD

  Функция обратного счета
- SFB 2 CTUD

  Функция прямого и обратного счета

Вы можете вызывать эти SFB с экземплярными блоками данных или использовать эти SFB как локальные экземпляры в функциональном блоке.

Вы можете найти описание интерфейса функций для программирования в автономном режиме (offline) в "стандартной библиотеке" *Standard Library* в разделе *System Function Blocks.*

Примеры вызовов этих функций находятся на прилагаемой дискете в библиотеке STL_Book в разделе "Basic Functions" в функциональном блоке FB 108 или в исходном файле Chap_8 или в библиотеке SCL_Book в разделе "30 SCL Functions".

Параметры вышеуказанных IEC-функций счетчиков предлагаются Вашему вниманию в таблице 8.2.

Таблица 8.2 Параметры IEC-функций счетчиков

Название (Name)	Представление в SFB			Объявление (Declaration)	Тип (Data type)	Описание (Description)
CU	0	-	2	INPUT	BOOL	Вход прямого счета (Up count input)
CD	-	1	2	INPUT	BOOL	Вход обратного счета (Down count input)
R	0	-	2	INPUT	BOOL	Вход сброса счетчика (Reset input)
LOAD	-	1	2	INPUT	BOOL	Вход загрузки (Load input)
PV	0	1	2	INPUT	INT	Заданное значение счетчика (Preset value)
Q	0	1	-	OUTPUT	BOOL	Состояние счетчика (Counter status)
QU	-	-	2	OUTPUT	BOOL	Состояние счетчика в режиме прямого счета (Count counter status up)
QD	-	-	2	OUTPUT	BOOL	Состояние счетчика в режиме обратного счета (Count counter status down)
CV	0		2	OUTPUT	INT	Текущее значение счетчика (Current counter value)

8.6.1 Функция прямого счета SFB 0 CTU

Параметры IEC-счетчика SFB 0 CTU показаны в таблице 8.2.

Если сигнал на входе прямого счета счетчика (up counter input) CU изменяет свое состояние с "0" на "1" ("прямой фронт"), тогда текущее значение счетчика инкрементируется (увеличивается на единицу) и отображается на выходе CV. При первом вызове (при состоянии сигнала "0" на входе сброса R) значение счетчика соответствует заранее заданному значению на входе PV (Preset value).

Если текущее значение достигает верхнего предела, равного 32767, значение счетчика больше не увеличивается. При этом сигнал на входе CU игнорируется.

Если сигнал на входе сброса R принимает значение "1", то счетчик сбрасывается в 0. При этом положительный фронт на входе счетчика CU игнорируется, пока состояние сигнала на входе сброса R равно "1". На выход Q счетчика будет выводиться значение "1", если значение на выходе CV будет больше или равно заранее заданного значения счетчика на входе PV.

IEC-счетчик SFB 0 CTU работает в рабочих режимах RESTART и RUN. IEC-счетчик SFB 0 CTU сбрасывается при холодном перезапуске.

8.6.2 Функция обратного счета SFB 1 CTD

Параметры IEC-счетчика SFB 1 CTD показаны в таблице 8.2.

Если сигнал на входе обратного счета счетчика (down counter input) CD изменяет свое состояние с "0" на "1" ("прямой фронт"), тогда текущее значение счетчика декрементируется (уменьшается на единицу) и отображается на выходе CV. При первом вызове (при состоянии сигнала "0" на входе LOAD) значение счетчика соответствует заранее заданному значению на входе PV (Preset value).

Если текущее значение достигает нижнего предела, равного -32768, значение счетчика далее не уменьшается. При этом сигнал на входе CD игнорируется.

Если сигнал на входе LOAD имеет значение "1", то счетчик сбрасывается и принимает заранее заданное значение (на входе PV). При этом положительный фронт на входе счетчика CD игнорируется, пока состояние сигнала на входе LOAD равно "1". На выход Q счетчика будет выводиться значение "1", если значение на выходе CV будет меньше или равно нулю.

IEC-счетчик SFB 1 CTD работает в рабочих режимах RESTART и RUN. IEC-счетчик SFB 1 CTD сбрасывается при холодном перезапуске.

8.6.3 Функция прямого и обратного счета SFB 2 CTUD

Параметры IEC-счетчика SFB 2 CTUD показаны в таблице 8.2.

Если сигнал на входе прямого счета счетчика (up counter input) CU изменяет свое состояние с "0" на "1" ("прямой фронт"), тогда текущее значение счетчика инкрементируется (увеличивается на единицу) и отображается на выходе CV. Если сигнал на входе обратного счета счетчика (down counter input) CD изменяет свое состояние с "0" на "1" ("прямой фронт"), тогда текущее значение счетчика декрементируется (уменьшается на единицу) и отображается на выходе CV. Если сигнал на обоих входах счетчика CU и CD изменяет свое состояние с "0" на "1" ("прямой фронт"), тогда текущее значение счетчика не изменяется.

Если текущее значение достигает верхнего предела, равного 32767, значение счетчика больше не увеличивается. При этом в дальнейшем сигнал на входе CU игнорируется.

Если текущее значение достигает нижнего предела, равного -32768, значение счетчика далее не уменьшается. При этом в дальнейшем сигнал на входе CD игнорируется.

Если сигнал на входе LOAD имеет значение "1", то счетчик сбрасывается и принимает заранее заданное значение (на входе PV). При этом положительные фронты сигналов на входах счетчика игнорируются, пока

состояние сигнала на входе LOAD равно "1".

Если состояние сигнала на входе сброса R принимает значение "1", то счетчик сбрасывается в 0. При этом положительные фронты сигналов на входах счетчика и состояние сигнала "1" на входе LOAD игнорируются, пока состояние сигнала на входе сброса R равно "1". На выход QU счетчика будет выводиться значение "1", если значение на выходе CV будет больше или равно значению на входе PV. На выход QD счетчика будет выводиться значение "1", если значение на выходе CV будет меньше или равно нулю.

IEC-счетчик SFB 2 CTUD работает в рабочих режимах RESTART и RUN. IEC-счетчик SFB 2 CTUD сбрасывается при холодном перезапуске.

8.7 Пример счетчика деталей

В данном разделе представлен пример, с помощью которого иллюстрируется работа с таймерами и счетчиками. В этом примере запрограммированы входы, выходы и меркеры, так что данная программа может быть включена в любое место любого блока. Пример выполнен как функция без параметров.

Описание функций

Детали должны переноситься лентой конвейера. Для обнаружения и подсчета деталей используется фотодатчик. После того, как подсчитанное число деталей становится равным заданному максимальному количеству, счетчик посылает сигнал окончания работы "Finished". Счетчик снабжен цепью слежения. Если состояние сигнала от фотодатчика не меняется в течение заданного времени, эта цепь слежения генерирует соответствующий сигнал.

Вход "Set" обеспечивает передачу счетчику начального значения (число деталей, которое должно быть сосчитано). Положительный фронт сигнала от фотодатчика вызывает уменьшение на единицу значения счетчика. Когда значение счетчика достигнет значения 0, счетчикает посылает сигнал окончания работы "Finished". Должно выполняться условие, согласно которому детали на ленте конвейера лежат отдельно (с интервалом между отдельными деталями).

Вход "Set" обеспечивает также установку сигнала "Active". Контроллер отслеживает изменение состояния сигнала, поступающего от фотодатчика, только во время установления сигнала "Active". Сигнал "Active" сбрасывается при завершении счета, когда последняя деталь минует фотодатчик.

В активном состоянии положительный фронт сигнала от фотодатчика запускает таймер со значением времени "Dura1" в режиме таймера с памятью. Если на входе Start таймера "0" в следующем цикле сканирования таймер в дальнейшем не продолжает отсчет времени. Новый положительный фронт сигнала от фотодатчика перезапускает таймер. Следующий положительный фронт сигнала от фотодатчика, перезапускающий таймер, генерируется, после того как фотодатчик выдаст отрицательный фронт сигнала. Тогда таймер запустится со значением времени "Dura2".

Если фотодатчик обнаружит деталь через промежуток времени, больший чем значение "Dura1" или будет свободен ("free") в течение промежутка времени, большего, чем значение "Dura2", то время заканчивается и таймер сигнализирует о сбое - выдает сигнал "Fault".

Первый сигнал "Active" запускает таймер со значением времени "Dura2". Сигнал "Set" запускает счетчик со схемой слежения. Положительные и отрицательные фронты сигнала, поступающие от фодатчика, используются для управления счетчиком, для выбора промежутка времени и для запуска (перезапуска) таймера (watchdog timer).

Проверка наличия положительного и отрицательного фронта сигнала требуется часто, и, поэтому, в качестве "сверхоперативной памяти" можно использовать область временных локальных данных для запоминания результата проверки. Временные локальные данные - это "внутриблочные" переменные (переменные, объявленные в блоке, а не в таблице символов). В данном примере результаты проверки хранятся в меркерах импульса ("pulse memory bits") в области временных локальных даных. Сигналы от меркеров фронта ("edge memory bits") требуются также в последующих циклах сканирования программы, поэтому эти меркеры не должны располагаться в области временных локальных даных.

Данная программа-пример выполнен как функция без параметров. Вы можете вызывать эту функцию (например, из OB 1) следующим образом:

```
Call "Counter_control";
```

В конце данной главы представлен исходный текст программы-примера с символьной адресацией таймеров, счетчиков и меркеров.

Глобальные символы могут также использоваться без кавычек (без апострофа), если они не содержат специальных символов. Если же символ (символьное имя) содержит специальный символ (например, умляут или пробел [space]), то такое имя должно быть заключено в кавычки. В компилированных блоках редактор STL отображает все глобальные символы в кавычках.

Для большей ясности и лучшей читаемости представленная программа разделена на сегменты. Последний сегмент, имеющий заголовок BLOCK END (конец блока), не является необходимым, а служит лишь для обозначения окончания блока. Такой прием бывает очень полезно использовать в случаях, когда блоки имеют чрезмерно большой размер.

Вы можете найти таблицу символов (symbol table) на прилагаемой к данной книге дискете в библиотеке STL_Book в разделе "Conveyor Example" в объекте *Symbol*, в исходной программе "Conveyor" в разделе исходных файлов *Source Files* и в скомпилированной программе в разделе *Blocks,* в функции FC 12.

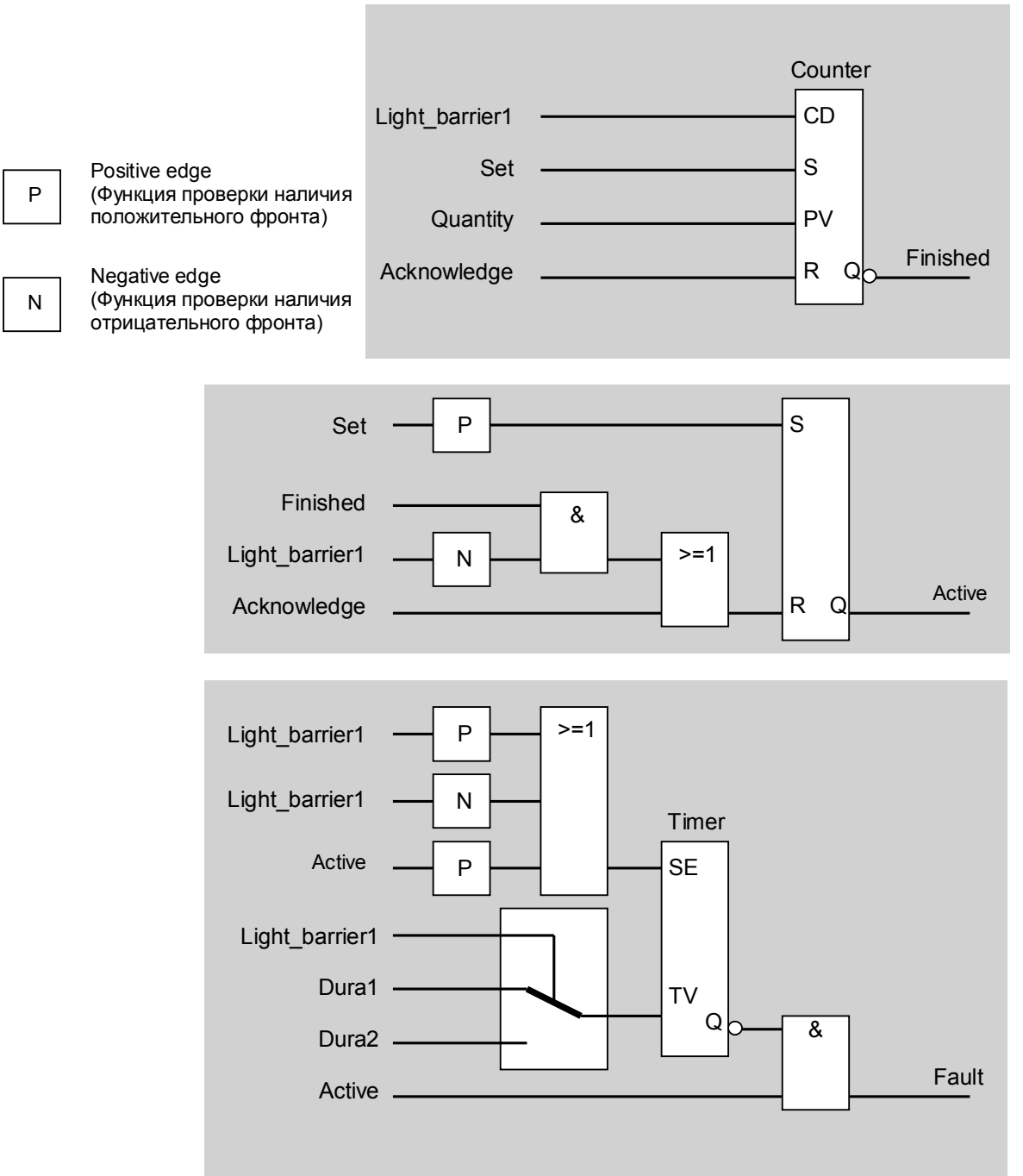


Рис. 8.3 Пример счетчика деталей

```
FUNCTION  "Counter_control" : VOID
//TITLE = Счетчик деталей с системой слежения
//Данный пример иллюстрирует работу таймеров и счетчиков
NAME    : Count
AUTHOR  : Berger
FAMILY  : STL_Book
Version : 01.00
VAR_TEMP
    PM_LB_P : BOOL;        //Положительный фронт импульса от фотодатчика
    PM_LB_N : BOOL;        //Отрицательный фронт импульса от фотодатчика
END_VAR
BEGIN
NETWORK
TITLE = Counter_Control
    A   Light_barnerl;      //При срабатывании фотодатчика
    CD  Count;              //декрементировать счетчик на 1
    A   Set;
    L   Quantity;           //Загрузить в счетчик заданное количество
//                            "Quantity"
    S   Count;
    A   Acknowledge;
    R   Count;
    AN  Count;              //Когда значение счетчика достигнет 0,
    =   Finished;           //генерируется сигнал окончания работы
NETWORK
TITLE = Activate monitor
    A   Light_barrierl;
    FP  EM_LB_P;            //Генерировать меркер импульса
    =   PM_LB_P;            //при положительном фронте сигнала от датчика
    A   Light_barrierl;
    FN  EM_LB_N;            //Генерировать меркер импульса
    =   PM_LB_N;            //при отрицательном фронте сигнала от датчика
    A   Set;
    FP  EM_ST_P;
    S   Active;             //Активировать систему слежения
    A   Finished;
    A   PM_LB_N;
    0   Acknowledge;
    R   Active;             //Деактивировать систему слежения
NETWORK
TITLE = Monitoring circuit
    L   Dura1;              //Если фотодатчик в состоянии "1",
    A   Light_barrierl;     //то выполняется переход JC к Dl и
    JC  D1;                 //аккумулятор получает значение "Dura1",
    L   Dura2;              //иначе - значение "Dura2"
Dl: A   Active;
    FP  EM_AC_P;            //Если положительный фронт в "Active",
    0   PM_LB_P;            //или положительный фронт от датчика,
    0   PM_LB_N;            //или отрицательный фронт от датчика, то
    SE  Monitor;            //таймер запускается или перезапускается
    AN  Monitor;
    A   Active;             //Если время заканчивается во время
//                            действия сигнала "Active",
    =   Fault;              //то генерируется сигнал сбоя "Fault"
NETWORK
TITLE = Block End
    BE;
END_FUNCTION
```

Функции для обработки чисел

Функции для обработки чисел используются для обработки численных значений преимущественно типов INT, DINT и REAL, что значительно расширяет функциональные возможности PLC. Здесь будут рассмотрены функции для обработки чисел, используемые в STL-программах. В языке программирования SCL функции сравнения, логические функции с данными формата Word и арифметические функции выполняются с помощью операторов (см. раздел 27.4 "Выражения"); остальные функции для обработки чисел включены в язык SCL в виде стандартных функций (см. разделы 30.3 "Математические функции", 30.4 "Функции сдвига и ротации", 30.5 "Функции преобразования").

Функции сравнения формируют двоичный результат сравнения двух величин. Функции сравнения могут обрабатывать данные типов INT, DINT и REAL.

Арифметические функции позволяют выполнять в программе вычисления. Все базовые арифметические операции могут быть выполнены с данными типов INT, DINT и REAL.

Математические функции расширяют собой вычислительные возможности в программе. Математические функции добавляют к базовым арифметическим функциям такое расширение как тригонометрические функции.

Функции преобразования позволяют до и после выполнения вычислений приводить численные значения к требуемому типу данных.

Функции сдвига позволяют смещать содержимое аккумулятора влево или вправо. Функции сдвига позволяют также проверить бит, смещенный последним.

Логические функции с данными формата Word используются для маскирования численных значений для проверки отдельных битов и задания для них значений "0" или "1".

Функции для обработки чисел работают, главным образом, со значениями в блоках данных. Это могут быть блоки глобальных данных или экземплярные блоки данных, если используются статические локальные данные. В разделе 18.2 "Функции для блоков данных" показано использование блоков данных и варианты адресации данных.

За исключением аккумуляторов для хранения временных результатов очень удобны временные локальные данные.

9 Функции сравнения

Операции сравнения величин на предмет их равенства, неравенства; операции: больше чем, больше чем или равно, меньше чем, меньше чем или равно; функции сравнения в двоичных логических операциях.

10 Арифметические функции

Базовые арифметические операции; цепочки вычислений; сложение констант; декрементирование и инкрементирование.

11 Математические функции

Тригонометрические функции; обратные тригонометрические функции; возведение в квадрат; извлечение квадратного корня; экспоненты и логарифмы.

12 Функции преобразования

Преобразование данных типов INT/DINT в данные BCD типа и наоборот; преобразование данных типа DINT в данные типа REAL и наоборот с различными способами округления; нахождение обратного кода двоичного числа, инвертирование и нахождение абсолютной величины.

13 Функции сдвига

Сдвиг влево, вправо, на слово и двойное слово, сдвиг в соответствии с правилами для знаков; циклический сдвиг содержимого аккумулятора 1 влево или вправо; сдвиг и циклический сдвиг с параметрами сдвига, заданными константой или в аккумуляторе 2.

14 Логические функции с данными формата Word

Операции AND (И), OR (ИЛИ), Exclusive OR (Исключающее ИЛИ); логические операции со словом, с двойным словом, с константой или с содержимым аккумулятора 2.

9 Функции сравнения

Функции сравнения обеспечивают выполнение операции сравнения двух численных значений, одно из которых находится в аккумуляторе accumulator 1, а второе находится в аккумуляторе accumulator 2. После выполнения операции сравнения функции сравнения устанавливают RLO (результат логической операции) и биты состояния CC0 и CC1. Этот результат может в дальнейшем быть использован в двоичных логических операциях, в операциях с памятью или в операторах перехода. В таблице 9.1 Вы найдете обзор доступных пользователю функций сравнения.

Таблица 9.1
Общее представление функций сравнения

Операция сравнения	Типы данных		
	INT	DINT	REAL
Равно	==I	==D	==R
Не равно	<>I	<>D	<>R
Больше чем	>I	>D	>R
Больше чем или равно	>=I	>=D	>=R
Меньше чем	<I	<D	<R
Меньше чем или равно	<=I	<=D	<=R

В главе 15 "Биты состояния" будет показано, как функции сравнения устанавливают биты состояния CC0 и CC1.

В этом разделе будут рассмотрены функции для обработки чисел, используемые в языке программирования STL. Тогда как в языке программирования SCL выполнение функций сравнения обеспечивается с помощью соответствующих выражений (см. раздел 27.4.2 "Операторы сравнения").

Примеры, рассматриваемые в данной главе, Вы можете найти на прилагаемой дискете в библиотеке STL_Book library в разделе "Digital Functions" ("Функции для обработки чисел") в функциональном блоке FB 109 или в исходном файле Chap_9.

9.1 Общее представление функций сравнения

Вы можете программировать функции сравнения в соответствии со следующей общей схемой:

```
Загрузка адреса Address1;
Загрузка адреса Address2;
Функция сравнения;
Присвоение результата Result;
```

При выполнении первой операции загрузки Load из первого адреса число помещается в аккумулятор accumulator 1. При загрузке из второго адреса содержимое аккумулятора accumulator 1 перемещается в аккумулятор accumulator 2 (см. раздел 6.2 "Операции загрузки"). Теперь с содержимым двух аккумуляторов можно выполнить операции с помощью функций сравнения.

Функции сравнения возвращают двоичный результат (тип BOOL), который может быть назначен двоичному адресу или может быть использован в какой-либо другой двоичной проверке (опросе).

Функции сравнения не изменяют содержимого аккумуляторов. Они всегда выполняются вне всякой связи с какими-либо условиями.

В таблице 9.2 показаны примеры для различных типов данных. Инструкции сравнения выполняются в соответствии с определенными параметрами независимо от содержимого аккумуляторов.

Таблица 9.2
Примеры функций сравнения

Применение функции сравнения для данных типа INT	Меркер M99.0 сбрасывается, если значение в слове MW 92 равно 120, иначе меркер не сбрасывается	`L  MW 92;`
		`L  120;`
		`==I;`
		`R  M 99.0;`

Применение функции сравнения для данных типа DINT	Переменная "CompResult" в блоке данных "Global_DB" устанавливается, если переменная "CompVal1" меньше чем "CompVal2", иначе она сбрасывается	`L "Global_DB"."CompVal1";`
		`L "Global_DB"."CompVal2";`
		`<D;`
		`= "Global_DB"."CompResult";`

Применение функции сравнения для данных типа REAL	Если переменная #Actval больше чем или равна переменной #Calibra, то переменная #Recali устанавливается, иначе - не устанавливается.	`L  #Actval;`
		`L  #Calibra;`
		`>=R;`
		`S  #Recali;`

В случае, когда данные имеют тип INT, CPU сравнивает только расположенные справа слова (младшие слова) в аккумуляторах; содержимое слов, расположенных слева (старшие слова), в расчет не берется.

В случае, когда данные имеют тип REAL, выполняется проверка для определения корректности (в соответствии с типом данных) содержимого аккумуляторов. Если результат этой проверки отрицателен, то CPU сбрасывает RLO в состояние "0", а биты состояния CC0, CC1, OV и OS устанавливает в состояние "1".

9.2 Описание функций сравнения

Проверка на равенство чисел

Инструкция для сравнения чисел с целью определения их равенства интерпретирует содержимое аккумуляторов в соответствии с типом данных, указанным в инструкции. Инструкция призвана выявить равенство двух числовых величин. После выполнения инструкции для RLO устанавливается значение "1", в следующих случаях:

- для данных типа INT
  если содержимое младшего слова аккумулятора accumulator 2 равно содержимому младшего слова аккумулятора accumulator 1,

- для данных типа DINT
  если содержимое аккумулятора accumulator 2 равно содержимому аккумулятора accumulator 1,

- для данных типа REAL
  если содержимое аккумулятора accumulator 2 равно содержимому аккумулятора accumulator 1 при условии, что оба аккумулятора содержат корректные действительные (REAL) числа.

Если два числа формата REAL равны, но некорректны, условие равенства ("equal to") не выполняется (RLO = "0").

Проверка на неравенство чисел

Инструкция для сравнения чисел с целью определения их неравенства интерпретирует содержимое аккумуляторов в соответствии с типом данных, указанным в инструкции. Инструкция призвана выявить неравенство двух числовых величин. После выполнения инструкции для RLO устанавливается значение "1", в следующих случаях:

- для данных типа INT
  если содержимое младшего слова аккумулятора accumulator 2 не равно содержимому младшего слова аккумулятора accumulator 1,

- для данных типа DINT
  если содержимое аккумулятора accumulator 2 не равно содержимому аккумулятора accumulator 1,

- для данных типа REAL
  если содержимое аккумулятора accumulator 2 не равно содержимому аккумулятора accumulator 1 при условии, что оба аккумулятора содержат корректные действительные (REAL) числа.

Если два числа формата REAL не равны, но одно из них или оба некорректны, то условие неравенства ("not equal to") не выполняется (RLO = "0").

Проверка чисел на отношение по формуле "больше"

Инструкция для сравнения с целью определения выполнения условия "больше" интерпретирует содержимое аккумуляторов в соответствии с типом данных, указанным в инструкции. Инструкция призвана выявить, не является ли число, находящееся в аккумуляторе accumulator 2, больше числа, находящегося в аккумуляторе accumulator 1. После выполнения инструкции для RLO устанавливается значение "1", в следующих случаях:

- для данных типа INT
  если содержимое младшего слова аккумулятора accumulator 2 больше чем содержимое младшего слова аккумулятора accumulator 1,

- для данных типа DINT
  если содержимое аккумулятора accumulator 2 больше чем содержимое аккумулятора accumulator 1,

- для данных типа REAL
  если содержимое аккумулятора accumulator 2 больше чем содержимое аккумулятора accumulator 1 при условии, что оба аккумулятора содержат корректные действительные (REAL) числа.

Проверка чисел на отношение по формуле "больше или равно"

Инструкция для сравнения с целью определения выполнения условия "больше или равно" интерпретирует содержимое аккумуляторов в соответствии с типом данных, указанным в инструкции. Инструкция призвана выявить, не является ли число, находящееся в аккумуляторе accumulator 2, большим или равным числу, находящемуся в аккумуляторе accumulator 1. После выполнения инструкции для RLO устанавливается значение "1", в следующих случаях:

- для данных типа INT
  если содержимое младшего слова аккумулятора accumulator 2 больше или равно содержимому младшего слова аккумулятора accumulator 1,

- для данных типа DINT

если содержимое аккумулятора accumulator 2 больше или равно содержимому аккумулятора accumulator 1,

- для данных типа REAL
  если содержимое аккумулятора accumulator 2 больше или равно содержимому аккумулятора accumulator 1 при условии, что оба аккумулятора содержат корректные действительные (REAL) числа.

Проверка чисел на отношение по формуле "меньше"

Инструкция для сравнения с целью определения выполнения условия "меньше" интерпретирует содержимое аккумуляторов в соответствии с типом данных, указанным в инструкции. Инструкция призвана выявить, не является ли число, находящееся в аккумуляторе accumulator 2, меньше числа, находящегося в аккумуляторе accumulator 1. После выполнения инструкции для RLO устанавливается значение "1", в следующих случаях:

- для данных типа INT
  если содержимое младшего слова аккумулятора accumulator 2 меньше чем содержимое младшего слова аккумулятора accumulator 1,

- для данных типа DINT
  если содержимое аккумулятора accumulator 2 меньше чем содержимое аккумулятора accumulator 1,

- для данных типа REAL
  если содержимое аккумулятора accumulator 2 меньше чем содержимое аккумулятора accumulator 1 при условии, что оба аккумулятора содержат корректные действительные (REAL) числа.

Проверка чисел на отношение по формуле "меньше или равно"

Инструкция для сравнения с целью определения выполнения условия "меньше или равно" интерпретирует содержимое аккумуляторов в соответствии с типом данных, указанным в инструкции. Инструкция призвана выявить, не является ли число, находящееся в аккумуляторе accumulator 2, меньшим или равным числу, находящемуся в аккумуляторе accumulator 1. После выполнения инструкции для RLO устанавливается значение "1", в следующих случаях:

- для данных типа INT
  если содержимое младшего слова аккумулятора accumulator 2 меньше или равно содержимому младшего слова аккумулятора accumulator 1,

- для данных типа DINT
  если содержимое аккумулятора accumulator 2 меньше или равно содержимому аккумулятора accumulator 1,

- для данных типа REAL
  если содержимое аккумулятора accumulator 2 меньше или равно содержимому аккумулятора accumulator 1 при условии, что оба аккумулятора содержат корректные действительные (REAL) числа.

9.3 Функции сравнения в логических операциях

Функции сравнения возвращают двоичный результат логической операции RLO и, следовательно, могут быть использованы в сочетании с другими двоичными функциями. Функции сравнения устанавливают бит состояния FC, например, в логических операциях функция сравнения всегда является первичным опросом ("first check").

Функция сравнения в начале логической операции

Функция сравнения в начале логической операции всегда является первичным опросом ("first check"). RLO, возвращаемый функцией сравнения может быть непосредственно использован в логических операциях (в функциях проверки).

```
L  MW 120;
L  512;
>I;
A  Input1;
=  Output1;
```

В примере выход *Output1* устанавливается, если выполняется условие сравнения функции сравнения, а вход *Input1* имеет состояние "1".

Функция сравнения внутри логической операции

Функция сравнения внутри логической операции должна быть заключена в скобки, так как функция сравнения начинает новый логический этап (первичный опрос ["first check"]).

```
O  Input2;
O( ;
L  MW 122;
L  200;
<=I;
);
O  Input3;
=  Output2;
```

В примере выход *Output2* устанавливается, если вход *Input2* или вход *Input3* имеет состояние "1" или если выполняется условие сравнения функции сравнения.

Многократное использование функции сравнения

Так как функции сравнения не изменяют содержимого аккумуляторов, возможно многократное использование этих функций при программировании на STL.

```
L  MW 124;
L  1200;
>I;
JC GREA;
==I;
JC EQUA;
```

В примере использованы две функции сравнения для одного и того же содержимого аккумуляторов. Первая операция сравнения генерирует RLO = "1", если MW 124 больше чем 1200, поэтому выполняется переход к метке GREA. Без перезагрузки аккумуляторов вторая функция сравнения выполняет проверку условия равенства и генерирует новое значение RLO.

Функция сравнения устанавливает биты состояния, исходя из соотношения сравниваемых величин, т.е. независимо от типа проверяемого условия в операции сравнения. Вы можете использовать это, делая проверку битов состояния в соответствующих функциях перехода.

Вышеприведенный пример может быть также запрограммирован следующим образом:

```
L  MW 124;
L  1200;
>I;
JP GREA;
JZ EQUA;
```

В этом примере результат выполнения функции сравнения проверяется с помощью битов состояния CC0 и CC1. Собственно условие сравнения ("больше чем") в данном случае не влияет на установку битов состояния; могут быть использованы и другие условия сравнения, например, "меньше чем". Оператор JP проверяет не является ли первая из сравниваемых величин больше, чем вторая. Оператор JZ проверяет, не являются ли обе сравниваемые величины равными.

10 Арифметические функции

Арифметические функции обеспечивают выполнение базовых арифметических операций с двумя числовыми значениями, одно из которых находится в аккумуляторе accumulator 1, а второе находится в аккумуляторе accumulator 2. Результат арифметической операции помещается в аккумулятор accumulator 1. Биты состояния CC0, CC1, OV и OS обеспечивают информацию, касающуюся выполнения вычислений и результата (см. главу 15 "Биты состояния"). В таблице 10.1 Вы найдете обзор доступных пользователю арифметических функций.

Таблица 10.1
Обзор арифметических функций

Арифметическая функция	Типы данных		
	INT	DINT	REAL
Сложение (Addition)	+I	+D	+R
Вычитание (Subtraction)	-I	-D	-R
Умножение (Multiplication)	*I	*D	*R
Деление (Division with quotient as result)	/I	/D	/R
Остаток от деления (Division with remainder as result)	-	MOD	-

В добавление к базовым арифметическим операциям с участием числа, находящегося в аккумуляторе accumulator 2, Вы также можете прибавлять константы непосредственно к содержимому аккумулятора accumulator 1 или изменять его содержимое на некоторую фиксированную величину.

В этом разделе будут рассмотрены арифметические функции, используемые в языке программирования STL. В языке программирования SCL выполнение арифметических функций обеспечивается с помощью соответствующих выражений (см. раздел 27.4.1 "Арифметические выражения").

Примеры, рассматриваемые в данной главе, Вы можете найти на прилагаемой дискете в библиотеке STL_Book library в разделе "Digital Functions" ("Функции для обработки чисел") в функциональном блоке FB 110 или в исходном файле Chap_10.

10.1 Общее представление арифметических функций

Вы можете программировать арифметические функции в соответствии со следующей общей схемой:

```
Загрузка (load) адреса Address1;
Загрузка (load) адреса Address2;
Арифметическая функция;
Передача (transfer) результата Result;
```

При выполнении первой операции загрузки Load из первого адреса число помещается в аккумулятор accumulator 1. При загрузке из второго адреса содержимое аккумулятора accumulator 1 перемещается в аккумулятор accumulator 2 (см. раздел 6.2 "Операции загрузки"). Теперь с содержимым двух аккумуляторов можно выполнить операции с помощью арифметических функций.

Результат операции сохраняется в аккумуляторе accumulator 1.

Арифметические функции выполняются в соответствии с заданными параметрами. Они всегда выполняются вне всякой связи с какими-либо условиями.

В таблице 10.2 показаны примеры для различных типов данных.

Таблица 10.2
Примеры арифметических функций

Применение арифметических функций для данных типа INT	Значение в слове MW 100 делится на 250; целый результат хранится в слове MW 102.	`L  MW 100;`
		`L  250;`
		`/I;`
		`T  MW 102;`

Применение арифметических функций для данных типа DINT	Значения переменных "ArithVal1" и "ArithVal2" складываются, и результат заносится в переменную "ArithResult". Все переменные в блоке данных "Global_DB".	`L "Global_DB"."ArithVal1";`
		`L "Global_DB"."ArithVal2";`
		`+D;`
		`T "Global_DB"."ArithResult";`

Применение арифметических функций для данных типа REAL	Переменные #Actval и #Factor перемножаются; результат умножения передается в переменную #Display.	`L  #Actval;`
		`L  #Factor;`
		`*R;`
		`T  #Display;`

В случае, когда данные имеют тип INT, арифметические операции выполняются только для расположенных справа слов (младшие слова) в аккумуляторах; слова, расположенные слева (старшие), игнорируются.

В случае, когда данные имеют тип REAL, выполняется проверка для определения корректности (в соответствии с типом данных) содержимого аккумуляторов.

В S7-300 CPU (за исключением CPU 318) выполнение арифметических функций не изменяет содержимого аккумулятора accumulator 2; в S7-400 CPU и CPU 318 содержимое аккумулятора accumulator 2 после выполнения операции переписывается содержимым аккумулятора accumulator 3. При этом содержимое аккумулятора accumulator 4 смещается (заменяет собой прежнее значение) в аккумулятор accumulator 3 (см. рис. 10.1).

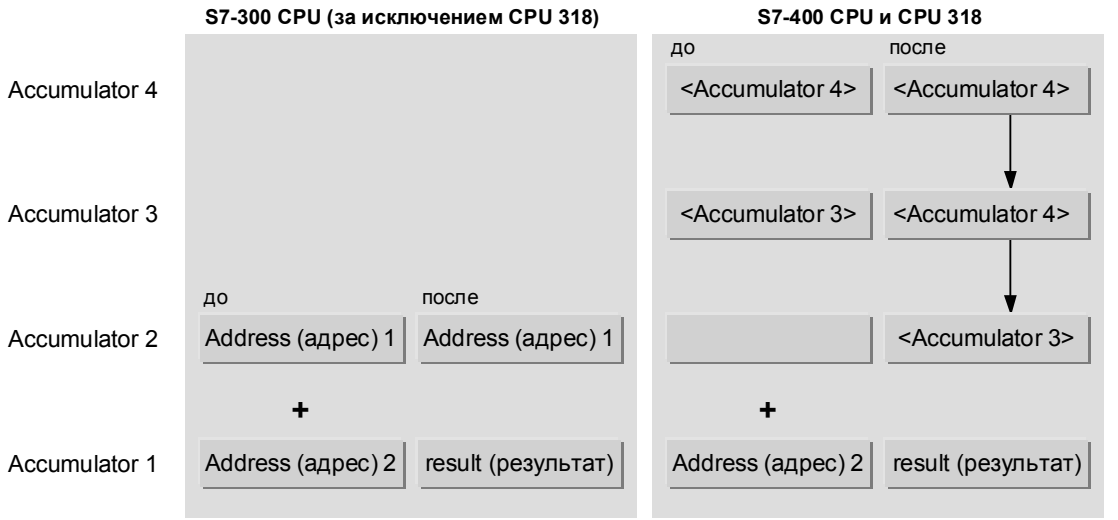


Рис. 10.1 Содержимое аккумуляторов при выполнении арифметических функций

10.2 Вычисления с данными типа INT

Сложение данных типа INT

Инструкция +I для сложения интерпретирует содержимое младших слов аккумуляторов 1 и 2 как числа целого типа (INT). Инструкция призвана выполнить сложение этих двух числовых величин и сохранить результат сложения в аккумуляторе accumulator 1.

После выполнения инструкции биты состояния CC0 и CC1 показывают, какова результирующая сумма: отрицательна, равна нулю или положительна. Биты состояния OV и OS указывают на нарушение границ разрешенного диапазона.

Вычитание данных типа INT

Инструкция - I для вычитания интерпретирует содержимое младших слов аккумуляторов 1 и 2 как числа целого типа (INT). Инструкция призвана выполнить вычитание числа, находящегося в аккумуляторе accumulator 1, из числа, находящегося в аккумуляторе accumulator 2 и сохранить результат

вычитания в аккумуляторе accumulator 1.

После выполнения инструкции биты состояния CC0 и CC1 показывают, какова разность: отрицательна, равна нулю или положительна. Биты состояния OV и OS указывают на нарушение границ разрешенного диапазона.

Умножение данных типа INT

Инструкция *I для перемножения интерпретирует содержимое младших слов аккумуляторов 1 и 2 как числа целого типа (INT). Инструкция призвана выполнить перемножение этих двух числовых величин и сохранить результат перемножения в формате числа DINT в аккумуляторе accumulator 1.

После выполнения инструкции биты состояния CC0 и CC1 показывают, каков результат перемножения чисел: отрицателен, равен нулю или положителен. Биты состояния OV и OS указывают на нарушение границ разрешенного диапазона.

Для последующей операции перемножения результат, находящийся в аккумуляторе accumulator 1, имеет формат числа DINT.

Деление данных типа INT

Инструкция /I для деления интерпретирует содержимое младших слов аккумуляторов 1 и 2 как числа целого типа (INT). Инструкция призвана выполнить деление числа, находящегося в аккумуляторе accumulator 2 (делимое), на число, находящееся в аккумуляторе accumulator 1 (делитель), и сохранить два результата деления: частное и остаток, оба имеющие тип INT (см. рис. 10.2).

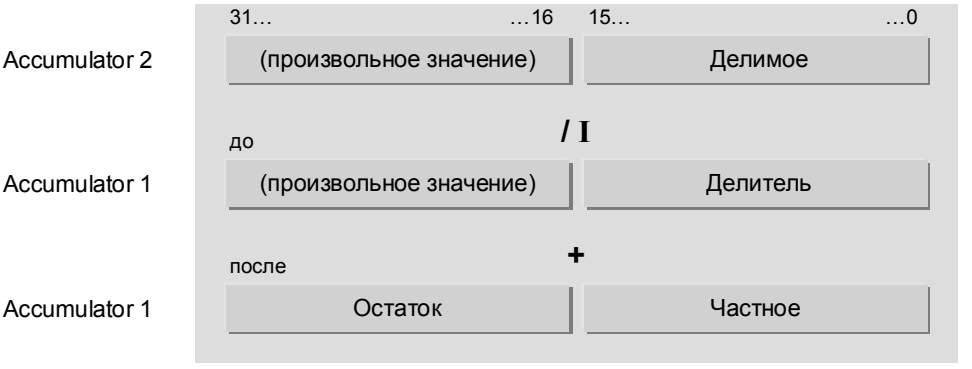


Рис. 10.2 Результат, возвращаемый арифметической функцией деления / I.

После выполнения инструкции младшее слово аккумулятора accumulator 1 содержит частное от деления. Частное является целым результатом операции деления. Частное от деления равно нулю в двух случаях: если делимое равно нулю, в то время как делитель не равен нулю, или если делимое меньше чем делитель. Частное от деления отрицательно, если делитель меньше нуля.

После выполнения инструкции старшее слово аккумулятора accumulator 1 содержит остаток от деления (не путать остаток от деления с дробной частью результата операции деления!). Остаток от деления отрицателен, если делимое меньше нуля.

После выполнения инструкции биты состояния CC0 и CC1 показывают, каково частное от деления: отрицательно, равно нулю или положительно. Биты состояния OV и OS указывают на нарушение границ разрешенного диапазона.

В случае деления на ноль частное от деления и остаток возвращаются с нулевыми значениями, а биты состояния CC0, CC1, OV и OS устанавливаются в состояние "1".

10.3 Вычисления с данными типа DINT

Сложение данных типа DINT

Инструкция +D для сложения интерпретирует содержимое аккумуляторов 1 и 2 как числа целого типа (DINT). Инструкция призвана выполнить сложение этих двух числовых величин и сохранить результат сложения в аккумуляторе accumulator 1.

После выполнения инструкции биты состояния CC0 и CC1 показывают, какова результирующая сумма: отрицательна, равна нулю или положительна. Биты состояния OV и OS указывают на нарушение границ разрешенного диапазона.

Вычитание данных типа DINT

Инструкция -D для вычитания интерпретирует содержимое аккумуляторов 1 и 2 как числа целого типа (DINT). Инструкция призвана выполнить вычитание числа, находящегося в аккумуляторе accumulator 1, из числа, находящегося в аккумуляторе accumulator 2 и сохранить результат вычитания (разность) в аккумуляторе accumulator 1.

После выполнения инструкции биты состояния CC0 и CC1 показывают, какова разность: отрицательна, равна нулю или положительна. Биты состояния OV и OS указывают на нарушение границ разрешенного диапазона.

Умножение данных типа DINT

Инструкция *D для перемножения интерпретирует содержимое аккумуляторов 1 и 2 как числа целого типа (DINT). Инструкция призвана выполнить перемножение этих двух числовых величин и сохранить результат перемножения в формате числа в аккумуляторе accumulator 1.

После выполнения инструкции биты состояния CC0 и CC1 показывают, каков результат перемножения чисел: отрицателен, равен нулю или положителен. Биты состояния OV и OS указывают на нарушение границ разрешенного диапазона.

Деление данных типа DINT

Инструкция /D для деления интерпретирует содержимое аккумуляторов 1 и 2 как числа целого типа (DINT). Инструкция призвана выполнить деление числа, находящегося в аккумуляторе accumulator 2 (делимое), на число, находящееся в аккумуляторе accumulator 1 (делитель), и сохранить частное от деления в аккумуляторе accumulator 1.

Частное является целым результатом операции деления. Частное от деления равно нулю в двух случаях: если делимое равно нулю, в то время как делитель не равен нулю, или если делимое меньше чем делитель. Частное от деления отрицательно, если делитель меньше нуля.

После выполнения инструкции биты состояния CC0 и CC1 показывают, каково частное от деления: отрицательно, равно нулю или положительно. Биты состояния OV и OS указывают на нарушение границ разрешенного диапазона.

В случае деления на ноль частное от деления возвращается с нулевым значением, а биты состояния CC0, CC1, OV и OS устанавливаются в состояние "1".

Остаток от деления данных типа DINT

Инструкция MOD для нахождения остатка от деления интерпретирует содержимое аккумуляторов 1 и 2 как числа целого типа (DINT). Инструкция призвана выполнить деление числа, находящегося в аккумуляторе accumulator 2 (делимое), на число, находящееся в аккумуляторе accumulator 1 (делитель), и сохранить остаток от деления в аккумуляторе accumulator 1.

Остаток - это то, что остается в результате операции деления нацело. Нельзя путать остаток от деления с дробной частью результата операции деления. Остаток от деления отрицателен, если делимое меньше нуля.

После выполнения инструкции биты состояния CC0 и CC1 показывают, каков остаток от деления: отрицателен, равен нулю или положителен. Биты состояния OV и OS указывают на нарушение границ разрешенного диапазона.

В случае деления на ноль остаток от деления возвращается с нулевым значением, а биты состояния CC0, CC1, OV и OS устанавливаются в состояние "1".

10.4 Вычисления с данными типа REAL

Сложение данных типа REAL

Инструкция +R для сложения интерпретирует содержимое аккумуляторов 1 и 2 как числа типа (REAL). Инструкция призвана выполнить сложение этих двух числовых величин и сохранить результат сложения в аккумуляторе accumulator 1.

После выполнения инструкции биты состояния CC0 и CC1 показывают, какова результирующая сумма: отрицательна, равна нулю или положительна. Биты состояния OV и OS указывают на нарушение границ разрешенного диапазона.

В случае, когда не соблюдаются корректные условия для выполнения операции (одно из входных значений является некорректным действительным числом (invalid REAL number) или делается попытка сложить [-бесконечное число] и [+бесконечное число]), тогда операция +R возвращает некорректное результирующее значение в аккумулятор accumulator 1 и устанавливает биты состояния CC0, CC1, OV и OS в состояние "1".

Вычитание данных типа REAL

Инструкция -R для вычитания интерпретирует содержимое аккумуляторов 1 и 2 как числа типа (REAL). Инструкция призвана выполнить вычитание числа, находящегося в аккумуляторе accumulator 1, из числа, находящегося в аккумуляторе accumulator 2 и сохранить результат вычитания (разность) в аккумуляторе accumulator 1.

После выполнения инструкции биты состояния CC0 и CC1 показывают, какова разность: отрицательна, равна нулю или положительна. Биты состояния OV и OS указывают на нарушение границ разрешенного диапазона.

В случае, когда не соблюдаются корректные условия для выполнения операции (одно из входных значений является некорректным действительным числом (invalid REAL number) или делается попытка вычесть [+бесконечное число] из [+бесконечного числа]), тогда операция -R возвращает некорректное результирующее значение в аккумулятор accumulator 1 и устанавливает биты состояния CC0, CC1, OV и OS в состояние "1".

Умножение данных типа REAL

Инструкция *R для перемножения интерпретирует содержимое аккумуляторов 1 и 2 как числа типа (REAL). Инструкция призвана выполнить перемножение этих двух числовых величин и сохранить результат перемножения в формате числа в аккумуляторе accumulator 1.

После выполнения инструкции биты состояния CC0 и CC1 показывают, каков результат перемножения чисел: отрицателен, равен нулю или положителен. Биты состояния OV и OS указывают на нарушение границ разрешенного диапазона.

В случае, когда не соблюдаются корректные условия для выполнения операции (одно из входных значений является некорректным действительным числом (invalid REAL number) или делается попытка перемножить бесконечное число и 0), тогда операция *R возвращает некорректное результирующее значение в аккумулятор accumulator 1 и устанавливает биты состояния CC0, CC1, OV и OS в состояние "1".

Деление данных типа REAL

Инструкция /R для деления интерпретирует содержимое аккумуляторов 1 и 2 как числа типа (REAL). Инструкция призвана выполнить деление числа, находящегося в аккумуляторе accumulator 2 (делимое), на число, находящееся в аккумуляторе accumulator 1 (делитель), и сохранить частное от деления в аккумуляторе accumulator 1.

После выполнения инструкции биты состояния CC0 и CC1 показывают, каково частное от деления: отрицательно, равно нулю или положительно. Биты состояния OV и OS указывают на нарушение границ разрешенного диапазона.

В случае деления на ноль частное от деления возвращается с нулевым значением, а биты состояния CC0, CC1, OV и OS устанавливаются в состояние "1".

В случае, когда не соблюдаются корректные условия для выполнения операции (одно из входных значений является некорректным действительным числом (invalid REAL number) или делается попытка разделить бесконечное число на бесконечное число или 0 разделить на 0), тогда операция /R возвращает некорректное результирующее значение в аккумулятор accumulator 1 и устанавливает биты состояния CC0, CC1, OV и OS в состояние "1".

10.5 Последовательное выполнение арифметических функций

Вы можете запрограммировать выполнение арифметических операций одну за другой; в этом случае результат выполнения первой операции будет использоваться для обработки в последующей операции. При этом аккумуляторы используются для временного хранения данных.

Примечание:

Необходимо помнить, что последовательное выполнение операций в процессорах S7-300 CPU и S7-400 CPU происходит по-разному (S7-300 CPU имеют только 2 аккумулятора, тогда как S7-400 CPU имеют 4).

Цепочки вычислений в S7-300 CPU (кроме CPU 318)

Цепочки вычислений выполняются следующим образом: после выполнения арифметической операции следует операция загрузки, после чего следует комбинирование вновь введенного значения согласно текущей инструкции с результатом предыдущей операции.

Пример:

Result1 := Value1 + Value2 - Value3

```
L  Value1;
L  Value2;
+I;                //Value1 + Value2
```

```
L  Value3;
-I;                //Сумма - Value3
T  Result1;
```

В CPU с двумя аккумуляторами первое загруженное значение остается без изменений в аккумуляторе accumulator 2 во время выполнения арифметической функции и может быть вновь использовано без необходимости повторной его загрузки.

Пример:

Result2 := Value5 + 2 · Value6

```
L  Value6;
L  Value5;
+R;                //Value5 + Value6
+R;                //Сумма + Value6
T  Result2;
```

Пример:

Result3 := Value7 · $(Value8)^2$

```
L  Value8;
L  Value7;
*D;                //Value7 · Value8
*D;                //Произведение · Value8
T  Result3;
```

Цепочки вычислений в S7-400 CPU и CPU 318

Цепочки вычислений выполняются следующим образом: после выполнения арифметической операции следует операция загрузки, после чего следует комбинирование вновь введенного значения согласно текущей инструкции с результатом предыдущей операции. В CPU, имеющих 4 аккумулятора, значение из аккумулятора accumulator 3 может быть смещено в аккумулятор accumulator 2 сразу после первой операции вычисления. Заблаговременно с помощью инструкции ENT Вы можете сохранить промежуточный результат в аккумуляторе accumulator 3 (например, перед строкой с оператором вычисления) (см. раздел 6.4 "Функции аккумуляторов").

Пример:

Result4 := (Value1 + Value2) · (Value3 - Value4)

```
L  Value1;
L  Value2;
+I ;
L  Value3;
ENT;
```

```
L  Value4;
-I ;
*I ;
T  Result4;
```

В данном примере сначала вычисляется сумма *Value1* и *Value2.* Затем во время загрузки в аккумулятор accumulator 1 *Value3* сумма сдвигается в аккумулятор accumulator 2. Оттуда инструкция ENT копирует ее в аккумулятор accumulator 3. После того как будет загружено значение *Value4, Value3* окажется в аккумуляторе accumulator 2. После выполнения операции вычитания сумма "выбирается" из аккумулятора accumulator 3 в accumulator 2. После этого сумма и разность могут быть перемножены.

10.6 Добавление констант к содержимому аккумулятора Accumulator 1

+	B#16#bb	добавление байтовой константы (byte)
+	±w	добавление константы формата слово (word)
+	L#±d	добавление константы формата двойное слово (duobleword)

Вы можете запрограммировать добавление константы в соответствии со следующей общей схемой:

```
Загрузка адреса Address;
Сложение с константой Const;
Передача результата  Result;
```

Операция добавления константы более предпочтительна для вычислений с адресуемыми данными, потому что по сравнению с арифметическими функциями она не влияет ни на содержимое остальных аккумуляторов, ни на биты состояния.

Операция добавления константы предназначена выполнять сложение указанной в инструкции константы с содержимым аккумулятора accumulator 1. Вы можете определить константу как шестнадцатеричную константу формата byte или как десятичную константу формата word или как десятичную константу формата duobleword. Если необходимо прибавить константу формата word, используя тип DINT, добавьте к описанию константы префикс L#. Если десятичная константа по величине превышает разрешенный диапазон для типа INT, вычисление (добавление константы) автоматически выполняется как для типа данных DINT.

Вы можете записать десятичное число со знаком минус - таким образом можно выполнить операцию вычитания константы из значения в аккумуляторе. Перед добавлением байтовой константы она преобразуется в

число формата INT со знаком.

Как и для арифметических функций, с данными типа INT операция добавления байтовой константы или константы формата word влияет только на младшее слово аккумулятора accumulator 1 и не влияет на старшее слово аккумулятора.

Если разрешенный диапазон для типа INT нарушается, то бит 15 (бит знака) переписывается. Операция добавления константы формата word обрабатывает все 32 бита аккумулятора, что соответствует операции вычисления для данных типа DINT.

Указанные операции добавления константы выполняются вне всякой связи с какими-либо условиями.

Пример операции добавления констант:

```
L   AddValue1;
+   B#16#21;
T   AddResult1;
```

В примере значение переменной AddValue1 увеличивается на 33 и передается в переменную AddResult1.

```
L   AddValue2;
+   -33;
T   AddResult2;
```

В примере значение переменной AddValue2 уменьшается на 33 и сохраняется в переменной AddResult2.

```
L   AddValue3;
+   L#-1;
T   AddResult3;
```

В примере значение переменной AddValue3 уменьшается на 1 и сохраняется в переменной AddResult3. Операция вычитания как при вычислениях с данными типа DINT.

10.7 Операции декрементирования и инкрементирования

DEC	n	Декрементирование
INC	n	Инкрементирование

Вы можете запрограммировать операции декрементирования и инкрементирования в соответствии со следующей общей схемой:

```
Загрузка адреса Address;
Декрементирование;
Передача результата  Result;
```

```
Загрузка адреса Address;
Инкрементирование;
Передача результата  Result;
```

Операции декрементирования и инкрементирования меняют значение в аккумуляторе accumulator 1. Содержимое аккумулятора уменьшается (декрементируется) или увеличивается (инкрементируется) на число, определенное в параметре инструкции. Этот параметр может принимать значение в диапазоне от 0 до 255.

При этом изменяется только значение в младшем байте аккумулятора. Старший байт аккумулятора остается без изменения. Операция инкрементирования выполняется таким образом, что если значение в результате инкрементирования превышает величину 255, то расчет значения начинается с начала (с 0), а операция декрементирования выполняется так, что если значение в результате декрементирования становится меньше 0, то расчет значения начинается с максимально возможного значения (с 255).

Операции декрементирования и инкрементирования выполняются независимо от значения RLO. Эти операции выполняются при появлении соответствующих инструкций и не влияют ни на RLO, ни на биты состояния.

Примеры операций декрементирования и инкрементирования:

```
L   IncValue;
INC 5;
T   IncResult;
```

В примере значение переменной IncValue увеличивается на 5 и передается в переменную IncResult.

```
L   DecValue;
DEC 7;
T   DecResult;
```

В примере значение переменной DecValue уменьшается на 7 и сохраняется в переменной DecResult.

11 Математические функции

К математическим функциям относятся следующие функции:

- синус (SIN), косинус (COS), тангенс (TAN)
- арксинус (ASIN), арккосинус (ACOS), арктангенс (ATAN)
- возведение в квадрат (SQR), извлечение квадратного корня (SQRT)
- экспонента (EXP), логарифм (LN)

Все математические функции обрабатывают числа в формате REAL. В зависимости от результата математические функции устанавливают биты состояния CC0, CC1, OV и OS (см. главу 15 "Биты состояния").

В этой главе будут рассмотрены математические функции, используемые в языке программирования STL. В языке программирования SCL выполнение математических функций обеспечивается с помощью соответствующих стандартных функций SCL (см. раздел 30.3 "Математические выражения").

Примеры, рассматриваемые в данной главе, Вы можете найти на прилагаемой дискете в библиотеке STL_Book library в разделе "Digital Functions" ("Функции для обработки чисел") в функциональном блоке FB 111 или в исходном файле Chap_11.

11.1 Общее представление математических функций

Математические функции в качестве входного значения используют число, находящееся в аккумуляторе accumulator 1, данное число обрабатывается в соответствии с инструкцией функции и вновь сохраняется в аккумуляторе accumulator 1.

Вы можете программировать математические функции в соответствии со следующей общей схемой:

```
Загрузка (load) адреса Address;
Математическая функция;
Передача (transfer) результата Result;
```

Математические функции изменяют только содержимое аккумулятора accumulator 1; содержимое всех остальных аккумуляторов остается неизменным. Математические функции всегда выполняются вне всякой связи с какими-либо условиями.

В таблице 11.1 показаны три примера применения математических функций.

Таблица 11.1 Примеры математических функций

Применение функции SIN	Значение в двойном слове MD 110 содержит значение угла в радианах. Функция генерирует синус угла и сохраняет результат в двойном слове MD 104.	`L MD 110;`
		`SIN;`
		`T MD 104;`

Применение функции SQRT	Функция генерирует квадратный корень из значения в переменной "MathValue1" и результат сохраняется в переменной "MathRoot". Все переменные в блоке данных "Global_DB".	`L "Global_DB".MathValue1;`
		`SQRT;`
		`T "Global_DB".MathRoot;`

Применение функции EXP	Переменные #Result содержит показатель степени e и #Exponent.	`L #Exponent;`
		`EXP;`
		`T #Result;`

Математические функции выполняются в соответствии с правилами обработки чисел типа REAL, даже когда применяется абсолютная адресация и тип входного числа не описан.

В случае, когда аккумулятор accumulator 1 содержит некорректное действительное число (invalid REAL number) при выполнении математической функции, тогда функция возвращает некорректное результирующее значение (REAL) в аккумулятор accumulator 1 и устанавливает биты состояния CC0, CC1, OV и OS в состояние "1".

11.2 Тригонометрические функции

Тригонометрические функции

- SIN синус
- COS косинус
- TAN тангенс

воспринимают действительное число (REAL) в аккумуляторе accumulator 1 как значение угла, выраженное в радианах.

Обычно для измерения углов применяются две единицы измерения: градусы с диапазоном измерения от 0° до 360° и радианы с диапазоном измерения от 0 до 2π (где π = +3.141593e+00). Обе эти единицы пропорциональны друг другу и могут быть преобразованы одна в другую. Например, угол 90° в радианах имеет значение π / 2 или +1.570796e+00.

В случаях, когда угол, выраженный в радианах, имеет величину, большую, чем 2π или +6.283185e+00, то из этой величины вычитается 2π или кратное 2π, до тех пор, пока угол не станет меньше 2π.

Пример:

Расчет реактивной мощности: $P_s = U \cdot I \cdot Sin\ (\varphi)$

```
L    PHI;          //Загрузка угла фи
SIN  ;
L    Current;      //Загрузка значения тока
*R   ;
L    Voltage;      //Загрузка значения напряжения
*R   ;
T    I_Power;      //Сохранение значения реактивной
                   //мощности
```

Необходимо отметить, что угол должен быть выражен в радианах. Если угол выражается в градусах, его значение должно быть умножено на коэффициент

π / 180 = +1.745329e-02

до того, как Вы используете значение угла в тригонометрических функциях.

11.3 Обратные тригонометрические функции (Arc-функции)

Arc-функции

- ASIN арксинус
- ACOS арккосинус
- ATAN арктангенс

являются обратными по отношению к соответствующим тригонометрическим функциям, рассмотренным выше. Эти функции используют действительное число (REAL), находящееся в аккумуляторе accumulator 1, и возвращают значение угла, выраженное в радианах (см. табл. 11.2).

Если превышается разрешенный диапазон значений, то обратные тригонометрические функции возвращают некорректное действительное число и устанавливают биты состояния CC0, CC1, OV и OS в состояние "1".

Таблица 11.2 Диапазоны значений для Arc-функций

Функция	Разрешенный диапазон значений	Возвращаемое значение
ASIN	-1 … +1	$-\pi/2 \ldots +\pi/2$
ACOS	-1 … +1	$0 \ldots +\pi$
ATAN	Не ограничен	$-\pi/2 \ldots +\pi/2$

Обычно

Пример:

В прямоугольном треугольнике величина одного из катетов относится к величине гипотенузы как 0,343. Сколько градусов составляет угол между ними?

Операция Arcsin(0.343) возвращает угол, выраженный в радианах; умножение на коэффициент 360/2π (= 57.2958) позволяет определить угол в градусах (приблизительно 20°).

Программа для выполнения указанных операций:

```
L    0.343;
ASIN  ;
L    57.2958;
*R   ;
T    Angle_Degree;
```

11.4 Другие математические функции

Пользователю доступны также еще несколько математических функций:

- SQR возведение в квадрат
- SQRT извлечение квадратного корня
- EXP экспоненциальная функция по основанию e
- LN логарифмическая функция по основанию e (натуральный логрифм)

Возведение в квадрат

Функция SQR выполняет операцию возведения в квадрат содержимого аккумулятора accumulator 1.

Пример:

Расчет объема цилиндра: $V = r^2 \cdot \pi \cdot H$

```
L    Radius;          //Загрузка значения радиуса
                      //основания
SQR  ;
L    Height;          //Загрузка значения высоты
*R   ;
L    3.141592;        //Загрузка значения числа π
*R   ;
T    Volume;          //Сохранение значения объема
```

Извлечение квадратного корня

Функция SQRT выполняет операцию извлечения квадратного корня из содержимого аккумулятора accumulator 1. Если содержимое аккумулятора меньше нуля, то функция возвращают некорректное действительное число и устанавливают биты состояния CC0, CC1, OV и OS в состояние "1". Если аккумулятор содержит "-0" (минус 0), функция возвращает "-0".

Пример:

$c = SQRT(a^2 + b^2)$

```
L    #a;
SQR  ;
L    #b;
SQR  ;
+R   ;
SQRT  ;
T    #c;
```

(Если b или c объявлены локальными переменными, перед их символами должен стоять префикс #, чтобы компилятор распознал их как локальные переменные; если b или c являются глобальными переменными, они должны быть заключены в кавычки).

Экспоненциальная функция (степенная функция с основанием e)

Функция EXP выполняет операцию нахождения степени с основанием e (содержимое аккумулятора accumulator 1 берется как показатель степени для этой функции (e^{Accu1}); e = 2.718282e+00).

Пример:
любая степень может быть определена по формуле

$a^b = e^{b \cdot \ln a}$

```
L    Value_a;
LN   ;
L    Value_b;
*R   ;
EXP  ;
T    Power;        //Значение степени
```

Логарифмическая функция (натуральный логарифм)

Функция LN выполняет операцию нахождения натурального логарифма (содержимое аккумулятора accumulator 1 берется как входное значение для этой функции; при этом e = 2.718282e+00). Если содержимое аккумулятора меньше нуля или равно нулю, то функция возвращают некорректное действительное число и устанавливают биты состояния CC0, CC1, OV и OS в состояние "1".

Функция натурального логарифма является обратной по отношению к экспоненциальной функции:

если $y = e^x$, то $x = \ln(y)$.

Пример:
с помощью переходных формул можно вычислять логарифмы по любому основанию:

Базовая формула:

$$\log_b a = \frac{\log_n a}{\log_n b}$$

В базовой формуле *b* и *n* являются произвольными основаниями. Если Вы назначите n = e, то можно будет вычислить логарифм по любому основанию, используя в вычислениях натуральные логарифмы:

$$\log_b a = \frac{\ln a}{\ln b}$$

В частном случае, при расчетах логарифмов по основанию 10 (десятичных логарифмов) формула принимает вид:

$$\lg a = \frac{\ln a}{\ln 10} = 0.4342945 \cdot \ln a$$

12 Функции преобразования

Функции преобразования преобразуют (конвертируют) тип данных, находящихся в аккумуляторе accumulator 1. На рис.12.1 представлен обзор функций преобразования, рассматриваемых в данной главе.

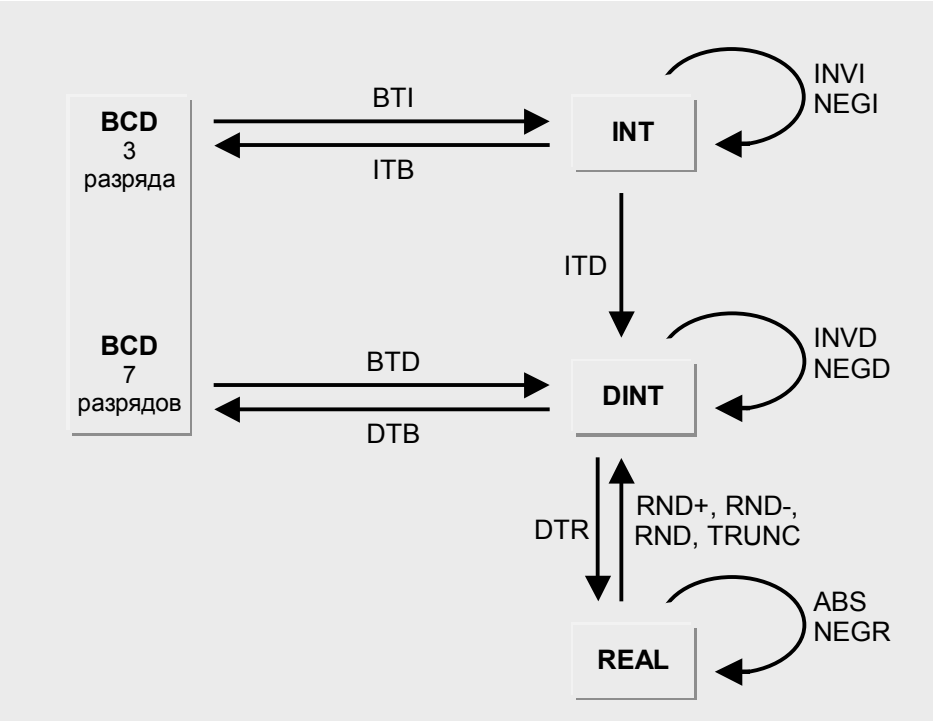


Рис.12.1 Обзор функций преобразования

В этой главе будут рассмотрены функции преобразования, используемые в языке программирования STL. В языке программирования SCL выполнение функций преобразования обеспечивается с помощью соответствующих стандартных функций SCL (см. раздел 30.5 "Функции преобразования").

Вы можете найти также подробную информацию о битовой структуре различных форматов данных в главе 24 "Типы данных", а также информацию о том, как функции преобразования устанавливают биты состояния в главе 15 "Биты состояния".

Примеры, рассматриваемые в данной главе, Вы можете найти на прилагаемой дискете в библиотеке STL_Book library в разделе "Digital Functions" ("Функции для обработки чисел") в функциональном блоке FB 112 или в исходном файле Chap_12.

12.1 Выполнение функций преобразования

Функции преобразования воздействуют только на данные, находящиеся в аккумуляторе accumulator 1. При этом отдельные функции преобразования воздействуют только на содержимое младшего слова в аккумуляторе accumulator 1 (биты с 0 по 15), другие функции воздействуют на содержимое аккумулятора в целом.

Вы можете программировать функции преобразования в соответствии со следующей общей схемой:

Загрузка (load) адреса Address;

Функция преобразования;

Передача (transfer) результата Result;

В таблице 12.1 показаны примеры применения функций преобразования для данных каждого типа. Функции преобразования конвертируют формат данных в соответствии с инструкцией функции, даже если используется абсолютная адресация и тип данных в аккумуляторе не был объявлен. Функции преобразования всегда выполняются вне всякой связи с какими-либо условиями.

Таблица 12.1 Примеры функций преобразования

Преобразование данных типа INT	Значение в двойном слове MW 120 интерпретируется как число INT. Функция сохраняет результат в двойном слове MW 122 как BCD-число.	`L MW 120;`
		`ITB;`
		`T MW 122;`

Преобразование данных типа DINT	Значение в переменной "ConvertDINT" интерпретируется как число DINT. Функция сохраняет результат в переменной "ConvertREAL" как число REAL. Все переменные в блоке данных "Global_DB".	`L "Global_DB".ConvertDINT;`
		`DTR;`
		`T "Global_DB".ConvertREAL;`

Преобразование данных типа REAL	Генерируется абсолютное значение для переменной #Display.	`L #Display;`
		`ABS;`
		`T #Display;`

Последовательное выполнение функций преобразования

Вы можете использовать содержимое аккумулятора accumulator 1 в нескольких последовательно выполняемых функциях преобразования, и в этом поэтапном процессе преобразования нет необходимости использовать дополнительные ресурсы для временного хранения конвертированных значений.

Пример:

```
L    BCD_Number;
BTI  ;               //BCD  ->  INT
ITD  ;               //INT  ->  DINT
DTR  ;               //DINT ->  REAL
T    REAL_Number;
```

В данном примере BCD-число в три шага преобразуется в число формата REAL.

12.2 Преобразование чисел форматов INT и DINT

Следующие функции обеспечивают выполнение преобразований чисел форматов INT и DINT:

- ITD преобразует INT в DINT
- ITB преобразует INT в BCD
- DTB преобразует DINT в BCD
- DTR преобразует DINT в REAL

Преобразование данных формата INT в формат DINT

Функция ITD интерпретирует содержимое младшего слова в аккумуляторе accumulator 1 (биты с 0 по 15) как число типа INT и пересылает значение бита 15 (бит знака) в старшее слово, т.е. в биты с 16 по 31.

Функция преобразования INT в DINT не устанавливает битов состояния.

Преобразование данных формата INT в формат BCD

Функция ITB интерпретирует содержимое младшего слова в аккумуляторе accumulator 1 (биты с 0 по 15) как число типа INT и преобразует его в трехразрядное BCD-число. Трехразрядное BCD-число в аккумуляторе accumulator 1 располагаются с выравниванием вправо и представляет собой значение десятичного числа. Знак располагается в битах с 12 по 15. Если все эти биты имеют значение "0", то знак положительный, если все биты имеют значение "1", то знак отрицательный. Содержимое старшего слова (биты с 16 по 31) остается без изменений.

Если исходное INT-число слишком велико, для того, чтобы выполнить преобразование в BCD-число (>999), тогда функция ITB устанавливает биты состояния OV и OS. Преобразование в таком случае не будет выполнено.

Преобразование данных формата DINT в формат BCD

Функция DTB интерпретирует содержимое аккумулятора accumulator 1 как число типа DINT и преобразует его в семиразрядное BCD-число. Семиразрядное BCD-число в аккумуляторе accumulator 1 располагаются с выравниванием вправо и представляет собой значение десятичного

числа. Знак располагается в битах с 28 по 31. Если все эти биты имеют значение "0", то знак положительный, если все биты имеют значение "1", то знак отрицательный.

Если исходное DINT-число слишком велико, для того, чтобы выполнить преобразование в BCD-число (> 9 999 999), в таком случае функция DTB устанавливает биты состояния OV и OS и преобразование в таком случае не может быть выполнено.

Преобразование данных формата DINT в формат REAL

Функция DTR интерпретирует содержимое аккумулятора accumulator 1 как число формата DINT и преобразует его в число формата REAL. Так как число формата DINT имеет большую точность чем число формата REAL, то в процессе преобразования формата числа может произойти округление, но только до следующего целого числа (как при выполнении операции RND). Если все эти биты имеют значение "0", то знак положительный, если все биты имеют значение "1", то знак отрицательный.

Функция DTR не устанавливает битов состояния.

12.3 Преобразование чисел формата BCD

Следующие функции обеспечивают выполнение преобразований чисел формата BCD:

- BTI преобразует BCD в INT
- BTD преобразует BCD в DINT

Преобразование данных формата BCD в формат INT

Функция BTI интерпретирует содержимое младшего слова в аккумуляторе accumulator 1 (биты с 0 по 15) как трехразрядное BCD-число и преобразует его в число типа INT. Три разряда числа в аккумуляторе accumulator 1 располагаются с выравниванием вправо и представляют собой значение десятичного числа. Знак располагается в битах с 12 по 15. Если все эти биты имеют значение "0", то знак положительный, если все биты имеют значение "1", то знак отрицательный. В процессе преобразования берется в расчет только состояние бита 15.

Содержимое старшего слова (биты с 16 по 31) остается без изменений.

При обнаружении "псевдотетрады" в исходном BCD-числе (численные значения с 10 по 15 или с A по F в шестнадцатеричном числе), CPU сообщает об ошибке назначения параметра ("a parameter assignment error") и вызывает организационный блок OB 121 (блок обработки синхронных ошибок). Если блок OB 121 не запрограммирован, то CPU переходит в состояние STOP.

Функция преобразования BTI не устанавливает битов состояния.

Преобразование данных формата BCD в формат DINT

Функция BTD интерпретирует содержимое аккумулятора accumulator 1 как семиразрядное BCD-число и преобразует его в число типа DINT. Семь разрядов числа в аккумуляторе accumulator 1 располагаются с выравниванием вправо и представляют собой значение десятичного числа. Биты с 28 по 31 содержат знак. Если все эти биты имеют значение "0", то знак положительный, если все биты имеют значение "1", то знак отрицательный. В процессе преобразования берется в расчет только состояние бита 31.

При обнаружении "псевдотетрады" в исходном BCD-числе (численные значения с 10 по 15 или с A по F в шестнадцатеричном числе), CPU сообщает об ошибке назначения параметра ("a parameter assignment error") и вызывает организационный блок OB 121 (блок обработки синхронных ошибок). Если блок OB 121 недоступен, то CPU переходит в состояние STOP.

Функция преобразования BTD не устанавливает битов состояния.

12.4 Функции преобразования чисел формата REAL

Существует несколько функций, обеспечивающих выполнение операций преобразования дробного числа формата REAL в число формата DINT (преобразование дробного числа в целое число). Эти функции отличаются друг от друга способом выполнения операции округления.

- RND+ преобразование с округлением "вверх" до следующего целого
- RND- преобразование с округлением "вниз" до следующего целого
- RND преобразование с округлением до следующего целого числа
- TRUNC преобразование без округления (усечение)

В таблице 12.2 показано различие действий указанных функций преобразования чисел формата REAL в числа формата DINT. Диапазон от -1 до +1 был выбран выбран для примера.

Преобразование с округлением "вверх" до следующего целого числа

Функция RND+ интерпретирует содержимое аккумулятора accumulator 1 как число типа REAL и преобразует его в число формата DINT.

Функция RND+ возвращает целое число, которое больше исходного числа или равно ему.

Если исходное число в аккумуляторе выходит за пределы допустимого для формата DINT диапазона или исходное число имеет другой формат, отличный от формата REAL, то функция RND+ устанавливает биты состояния OV и OS. Преобразование в таком случае не может быть выполнено.

Таблица 12.2 Режимы округления функции преобразования чисел типа REAL

Входное значение		Результат			
REAL	DW#16#	RND	RND+	RND-	TRUNC
1.00000001	3F80 0001	1	2	1	1
1.00000000	3F80 0000	1	1	1	1
0.99999995	3F7F FFFF	1	1	0	0
0.50000005	3F00 0001	1	1	0	0
0.50000000	3F00 0000	0	1	0	0
0.49999996	3EFF FFFF	0	1	0	0
5.877476e-39	0080 0000	0	1	0	0
0.0	0000 0000	0	0	0	0
-5.877476e-39	8080 0000	0	0	-1	0
-0.49999996	BEFF FFFF	0	0	-1	0
-0.50000000	BF00 0000	0	0	-1	0
-0.50000005	BF00 0001	-1	0	-1	0
-0.99999995	BF7F FFFF	-1	0	-1	0
-1.00000000	BF80 0000	-1	-1	-1	-1
-1.00000001	BF80 0001	-1	-1	-2	-1

Преобразование с округлением "вниз" до следующего целого числа

Функция RND- интерпретирует содержимое аккумулятора accumulator 1 как число типа REAL и преобразует его в число формата DINT.

Функция RND- возвращает целое число, которое меньше исходного числа или равно ему.

Если исходное число в аккумуляторе выходит за пределы допустимого для формата DINT диапазона или исходное число имеет другой формат, отличный от формата REAL, то функция RND- устанавливает биты состояния OV и OS. Преобразование в таком случае не будет выполнено.

Преобразование с округлением до ближайшего целого числа

Функция RND интерпретирует содержимое аккумулятора accumulator 1 как число типа REAL и преобразует его в число формата DINT.

Функция RND возвращает целое число, которое может быть больше или меньше исходного числа, или равно ему.

Функция RND возвращает то целое число, которое ближе к результату преобразования типа исходного числа. Если значение результата лежит точно посредине между четным и нечетным целыми числами, то четное число будет иметь более высокий приоритет, т.е. возвращается четное.

Если исходное число в аккумуляторе выходит за пределы допустимого для формата DINT диапазона или исходное число имеет другой формат, отличный от REAL, то функция RND устанавливает биты состояния OV и OS. Преобразование в таком случае не будет выполнено.

Преобразование без округления

Функция TRUNC интерпретирует содержимое аккумулятора accumulator 1 как число типа REAL и преобразует его в число формата DINT.

Функция TRUNC возвращает целую часть обрабатываемого числа, то есть, дробная часть числа отбрасывается ("усекается").

Если исходное число выходит за пределы допустимого для формата DINT диапазона или исходное число имеет другой формат, отличный от REAL, то функция TRUNC устанавливает биты состояния OV и OS. Преобразование в таком случае не будет выполнено.

12.5 Другие функции преобразования чисел

Существует еще несколько функций, обеспечивающих выполнение операций преобразования чисел:

- INVI нахождение обратного кода двоичного числа формата INT
- INVD нахождение обратного кода двоичного числа формата DINT
- NEGI инвертирование числа формата INT
- NEGD инвертирование числа формата DINT
- NEGD инвертирование числа формата REAL
- ABS нахождение абсолютного значения числа формата REAL

Нахождение обратного кода двоичного числа формата INT

Функция INVI инвертирует содержимое младшего слова в аккумуляторе accumulator 1 (биты с 0 по 15) бит за битом. Функция каждый двоичный ноль заменяет двоичной единицей и наоборот, каждую единицу заменяет нулем. Содержимое старшего слова (биты с 16 по 31) остается без изменений.

Функция преобразования INVI не устанавливает битов состояния.

Нахождение обратного кода двоичного числа формата DINT

Функция INVD инвертирует содержимое в аккумуляторе accumulator 1 бит за битом. Функция каждый двоичный ноль заменяет двоичной единицей и наоборот, каждую единицу заменяет нулем.

Функция преобразования INVD не устанавливает битов состояния.

Инвертирование числа формата INT

Функция NEGI интерпретирует содержимое младшего слова в аккумуляторе accumulator 1 (биты с 0 по 15) как число типа INT и изменяет его знак.

Эта операция идентична умножению на -1. При этом биты старшего слова в аккумуляторе accumulator 1 (биты с 16 по 31) не изменяют своего состояния.

Функция преобразования NEGI устанавливает следующие биты состояния: CC0, CC1, OV и OS.

Инвертирование числа формата DINT

Функция NEGD интерпретирует содержимое аккумулятора accumulator 1 как число типа DINT и изменяет его знак.

Эта операция идентична умножению на -1. При этом биты старшего слова в аккумуляторе accumulator 1 (биты с 16 по 31) не изменяют своего состояния.

Функция преобразования NEGD устанавливает следующие биты состояния: CC0, CC1, OV и OS.

Инвертирование числа формата REAL

Функция NEGR интерпретирует содержимое аккумулятора accumulator 1 как число формата REAL и умножает это число на -1 (функция меняет знак мантиссы, даже если число в аккумуляторе является некорректным действительным числом).

Функция NEGR не устанавливает битов состояния.

Нахождение абсолютного значения числа формата REAL

Функция ABS интерпретирует содержимое аккумулятора accumulator 1 как число формата REAL и формирует для этого числа абсолютное значение, устанавливая знак числа в значение "0" (функция устанавливает знак мантиссы в значение "0", даже если число в аккумуляторе является некорректным действительным числом).

Функция ABS не устанавливает битов состояния.

13 Функции сдвига

Функции сдвига сдвигают влево или вправо содержимое аккумулятора accumulator 1 бит за битом.

В таблице 13.1 представлен обзор доступных пользователю функций сдвига.

Таблица 13.1 Обзор функций сдвига

Функции сдвига	Word (Слово)		Duobleword (Двойное слово)	
	Число поз. как параметр	Число поз. в Accum2	Число поз. как параметр	Число поз. в Accum2
Сдвиг влево (Shift left)	SLW n	SLW	SLD n	SLD
Сдвиг вправо (Shift right)	SRW n	SRW	SRD n	SRD
Сдвиг со знаком (Shift with sign)	SSI n	SSI	SSD n	SSD
Циклический сдвиг влево (Rotate left)	-	-	RLD n	RLD
Циклический сдвиг вправо (Rotate right)	-	-	RRD n	RRD
Циклический сдвиг влево через бит CC1 (Rotate left through CC1)	-	-	RLDA [1)]	-
Циклический сдвиг вправо через бит CC1 (Rotate right through CC1)	-	-	RRDA [1)]	-

[1)] Без параметров, как при смещении только одного бита

В этой главе будут рассмотрены функции сдвига, используемые в языке программирования STL. В языке программирования SCL выполнение функций сдвига обеспечивается с помощью соответствующих стандартных функций SCL (см. раздел 30.4 "Сдвиг и циклический сдвиг").

Примеры, рассматриваемые в данной главе, Вы можете найти на прилагаемой дискете в библиотеке STL_Book library в разделе "Digital Functions" ("Функции для обработки чисел") в функциональном блоке FB 113 или в исходном файле Chap_13.

13.1 Выполнение функций сдвига

Функции сдвига позволяют побитно сдвигать влево или вправо данные, находящиеся в аккумуляторе accumulator 1. При этом в зависимости от функции сдвига аккумулятор содержит либо слово (word), либо двойное слово (duobleword). Биты, сдвигаемые за пределы слова (двойного слова), либо теряются (при операциях сдвига [shift operations]), либо переносятся на другую сторону слова или двойного слова (при операциях циклического сдвига [rotate operations]). Функции сдвига не влияют на содержимое других аккумуляторов.

Функции сдвига всегда выполняются вне всякой связи с какими-либо условиями. Функции сдвига воздействуют только на содержимое аккумулятора accumulator 1. При этом функции не влияют на результат логической операции (RLO).

Параметры для функции сдвига могут быть заданы следующими двумя путями:

- С числом позиций, заданным в аккумуляторе 2
- С числом позиций, заданным в инструкции с помощью параметра

Вы можете программировать операции двумя способами в соответствии со следующими общими схемами:

Загрузка (load) числа позиций Number_of_positions;

Загрузка (load) адреса Address;

Функция сдвига;

Передача (transfer) результата Result;

Загрузка (load) адреса Address;

Функция сдвига с параметром "число позиций" (Number_of_positions);

Передача (transfer) результата Result;

Функции сдвига устанавливают бит состояния CC0 в "0", а бит CC1 устанавливают в состояние, в котором находился последний перемещенный бит (см. рис. 13.1). Биты состояния проверяются с помощью двоичного опроса или в операциях перехода в соответствии с описанием в главе 15 "Биты состояния" и в главе 16 "Функции перехода".

В таблице 13.2 показаны несколько примеров применения функций сдвига для данных типов Word и Duobleword.

Функции сдвига для данных типа Word воздействуют только на содержимое младшего слова в аккумуляторе accumulator 1 (биты с 0 по 15). Содержимое старшего слова аккумулятора остается неизменным.

Функция циклического сдвига с битом состояния CC1 смещает содержимое аккумулятора accumulator 1 на одну разрядную позицию

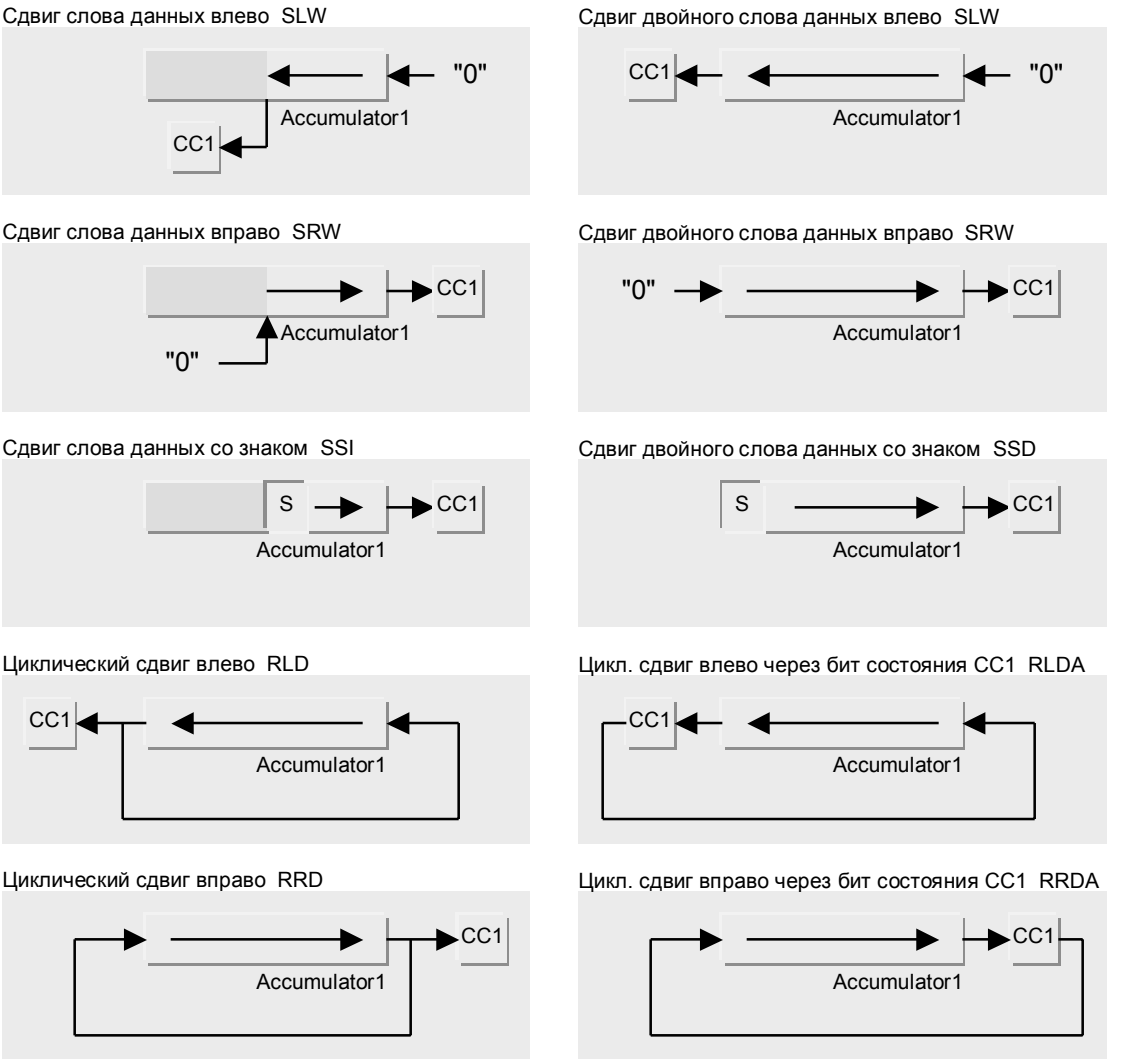


Рис.13.1 Принцип работы функций сдвига

Таблица 13.2 Примеры применения функций сдвига

Сдвиг данных формата Word (слово)	Значение в слове MW 130 сдвигается на 4 позиции влево и сохраняется в слове MW 132. Здесь число позиций задано параметром в инструкции.	`L   MW 130;`
		`SLW 4;`
		`T   MW 132;`

Сдвиг данных формата Duobleword (двойное слово)	Значение в переменной "ShiftOn" сдвигается вправо на "ShiftPos" позиций и сохраняется в переменной "ShiftOff". Здесь число позиций задано посредством аккумулятора accumulator 2.	`L   "Global_DB".ShiftPos;`
		`L   "Global_DB".ShiftOn;`
		`SRD ;`
		`T   "Global_DB".ShiftOff;`

Сдвиг со знаком	Сдвиг переменной #Actval со знаком вправо на 2 позиции и передача в переменную #Display.	`L   #Actval;`
		`SSI 2;`
		`T   #Display;`

Последовательное выполнение функций сдвига

Вы можете выполнять операцию сдвига содержимого аккумулятора accumulator 1 любое необходимое число раз.

Пример:

```
L    Value1;
SSD  4;
SLD  2;
T    Result1;
```

В данном примере в результате выполнения программы содержимое аккумулятора будет сдвинуто со знаком вправо на 2 позиции при этом два правых бита будут сброшены в состояние "0".

13.2 Операции сдвига

Сдвиг слова данных влево

SLW n	Сдвиг слова данных влево на n разрядов
SLW	Сдвиг слова данных влево на количество разрядов, указанное в аккумуляторе accumulator 2

Функция сдвига SLW позволяет бит за битом сдвигать влево данные, находящиеся в младшем слове аккумулятора accumulator 1 (т.е. в битах с 0 по 15). При этом разряды, освобождаемые при сдвиге содержимого указанных битов заполняются нулями. Биты, находящиеся в старшем слове аккумулятора остаются без изменения; нет также переноса данных в бит 16.

Число позиций, на которое выполняется операция сдвига, может быть указано в параметре инструкции SLW или может быть загружено в аккумулятор accumulator 2 в виде положительного числа формата INT, выровненного вправо. Если число позиций равно нулю, то операция сдвига не выполняется (нет операции, т.е. NOP); если число позиций больше 15, то после выполнения операции сдвига SLW все биты младшего слова аккумулятора будут иметь значение "0".

Если содержимое аккумуляторе accumulator 1 (младшего слова) интерпретируется как целое число формата INT, то сдвиг влево эквивалентен умножению на число, равное степенной функции с основанием 2 и показателем степени, равным числу позиций, на которые производится сдвиг.

Сдвиг двойного слова данных влево

SLD n	Сдвиг двойного слова данных влево на n разрядов
SLD	Сдвиг двойного слова данных влево на количество разрядов, указанное в аккумуляторе accumulator 2

Функция сдвига SLD позволяет бит за битом сдвигать влево данные, находящиеся во всех битах аккумулятора accumulator 1. При этом разряды, освобождаемые при сдвиге содержимого аккумулятора

заполняются нулями.

Число позиций, на которое выполняется операция сдвига, может быть указано в параметре инструкции SLD или может быть загружено в аккумулятор accumulator 2, в виде положительного числа формата INT, выровненного вправо. Если число позиций равно нулю, то операция сдвига не выполняется (нет операции, т.е. NOP); если число позиций больше 31, то после выполнения операции сдвига SLD все биты аккумулятора accumulator 1 будут иметь значение "0".

Если содержимое аккумуляторе accumulator 1 интерпретируется как целое число формата DINT, то сдвиг влево эквивалентен умножению на число, равное степенной функции с основанием 2 и показателем степени, равным числу позиций, на которые производится сдвиг.

Сдвиг слова данных вправо

SRW	n	Сдвиг слова данных вправо на n разрядов
SRW		Сдвиг слова данных вправо на количество разрядов, указанное в аккумуляторе accumulator 2

Функция сдвига SRW позволяет бит за битом сдвигать вправо данные, находящиеся в младшем слове аккумулятора accumulator 1 (т.е. в битах с 0 по 15). При этом разряды, освобождаемые при сдвиге содержимого указанных битов заполняются нулями. Биты, находящиеся в старшем слове (биты с 16 по 31) остаются без изменения.

Число позиций, на которое выполняется операция сдвига, может быть указано в параметре инструкции SRW или может быть загружено в аккумулятор accumulator 2 в виде положительного числа формата INT, выровненного вправо. Если число позиций равно нулю, то операция сдвига не выполняется (нет операции, т.е. NOP); если число позиций больше 15, то после выполнения операции сдвига SRW все биты младшего слова аккумулятора будут иметь значение "0".

Если содержимое аккумуляторе accumulator 1 (младшего слова) интерпретируется как целое число формата INT, то сдвиг вправо эквивалентен делению на число, равное степенной функции с основанием 2 и показателем степени, равным числу позиций, на которые производится сдвиг. Так как при сдвиге вправо производится заполнение битов нулевым значением, начиная со старшего бита в слове, то результатом всегда будет положительное число. Результат упомянутого деления соответствует округлению целой части.

Сдвиг двойного слова данных вправо

SRD	n	Сдвиг двойного слова данных вправо на n разрядов
SRD		Сдвиг двойного слова данных вправо на количество разрядов, указанное в аккумуляторе accumulator 2

Функция сдвига SRD позволяет бит за битом сдвигать вправо данные, находящиеся во всех битах аккумулятора accumulator 1. При этом разряды, освобождаемые при сдвиге содержимого этого аккумулятора заполняются нулями.

Число позиций, на которое выполняется операция сдвига, может быть указано в параметре инструкции SRD или может быть загружено в аккумулятор accumulator 2, в виде положительного числа формата INT,

выровненного вправо. Если число позиций равно нулю, то операция сдвига не выполняется (нет операции, т.е. NOP); если число позиций больше 31, то после выполнения операции сдвига SRD все биты аккумулятора accumulator 1 будут иметь значение "0".

Если содержимое аккумуляторе accumulator 1 интерпретируется как целое число формата DINT, то сдвиг вправо эквивалентен делению на число, равное степенной функции с основанием 2 и показателем степени, равным числу позиций, на которые производится сдвиг. Так как при сдвиге вправо производится заполнение битов нулевым значением, начиная со старшего бита в слове, то результатом всегда будет положительное число. Результат упомянутого деления соответствует операции округления целой части.

Сдвиг слова данных со знаком

SSI n	Сдвиг слова данных со знаком на n разрядов
SSI	Сдвиг слова данных со знаком на количество разрядов, указанное в аккумуляторе accumulator 2

Функция сдвига SSI позволяет бит за битом сдвигать вправо данные, находящиеся в младшем слове аккумулятора accumulator 1 (т.е. в битах с 0 по 15). При этом разряды, освобождаемые при сдвиге содержимого указанных битов заполняются значением бита 15 (бита, содержащего знак числа формата INT). Иначе говоря, этом разряды заполняются значением "0", если число положительное и значением "1", если число отрицательное.

Функция сдвига SSI не влияет на содержимое битов с 16 по 31.

Число позиций, на которое выполняется операция сдвига, может быть указано в параметре инструкции SSI или может быть загружено в аккумулятор accumulator 2 в виде положительного числа формата INT, выровненного вправо. Если число позиций равно нулю, то операция сдвига не выполняется (нет операции, т.е. NOP); если число позиций больше 15, то после выполнения операции сдвига SSI все биты младшего слова аккумулятора будут иметь значение, соответствующее знаку исходного числа в аккумуляторе.

Если содержимое аккумуляторе accumulator 1 (младшего слова) интерпретируется как целое число формата INT, то сдвиг вправо эквивалентен делению на число, равное степенной функции с основанием 2 и показателем степени, равным числу позиций, на которые производится сдвиг. Результат упомянутого деления соответствует округлению целой части.

Сдвиг двойного слова данных со знаком SSD

SSD n	Сдвиг слова данных со знаком на n разрядов
SSD	Сдвиг слова данных со знаком на количество разрядов, указанное в аккумуляторе accumulator 2

Функция сдвига SSD позволяет бит за битом сдвигать вправо данные, находящиеся во всех битах аккумулятора accumulator 1. При этом разряды, освобождаемые при сдвиге содержимого указанных битов заполняются значением бита 31 (бита, содержащего знак числа формата DINT). Иначе говоря, этом разряды заполняются значением "0", если число положительное и значением "1", если число отрицательное.

Число позиций, на которое выполняется операция сдвига, может быть указано в параметре инструкции SSD или может быть загружено в аккумулятор accumulator 2, в виде положительного числа формата INT, выровненного вправо. Если число позиций равно нулю, то операция сдвига не выполняется (нет операции, т.е. NOP); если число позиций больше 31, то после выполнения операции сдвига SSD все биты аккумулятора accumulator 1 будут иметь значение, соответствующее знаку исходного числа в аккумуляторе.

Если содержимое аккумуляторе accumulator 1 интерпретируется как целое число формата DINT, то сдвиг вправо эквивалентен делению на число, равное степенной функции с основанием 2 и показателем степени, равным числу позиций, на которые производится сдвиг. Результат упомянутого деления соответствует операции округления целой части.

13.3 Операции циклического сдвига

Циклический сдвиг влево

RLD n	Циклический сдвиг влево на n разрядов
RLD	Циклический сдвиг влево на на количество разрядов, указанное в аккумуляторе accumulator 2

Операция циклического сдвига RLD позволяет бит за битом сдвигать влево данные, находящиеся во всех битах аккумулятора accumulator 1. При этом вновь освобожденный при сдвиге содержимого разряд заполняется значением бита, который был "вытолкнут" из аккумулятора последним.

Число позиций, на которое выполняется операция циклического сдвига, может быть указано в параметре инструкции RLD или может быть загружено в аккумулятор accumulator 2, в виде положительного числа формата INT, выровненного вправо. Если число позиций равно нулю, то операция циклического сдвига не выполняется (нет операции, т.е. NOP); если число позиций равно 32, то содержимое аккумулятора accumulator 1 не изменяется, а бит состояния CC1 принимает состояние бита, перемещенного последним (бит 0). Если указанное число позиций равно 33, то содержимое аккумулятора accumulator 1 смещается на 1 разряд, если указанное число позиций равно 34, то содержимое смещается на 2 позиции и так далее.

Циклический сдвиг вправо

RRD n	Циклический сдвиг вправо на n разрядов
RRD	Циклический сдвиг вправо на количество разрядов, указанное в аккумуляторе accumulator 2

Операция циклического сдвига RRD позволяет бит за битом сдвигать вправо данные, находящиеся во всех битах аккумулятора accumulator 1. При этом вновь освобожденный при сдвиге содержимого разряд заполняется значением бита, который был "вытолкнут" из аккумулятора последним.

Число позиций, на которое выполняется операция циклического сдвига, может быть указано в параметре инструкции RRD или может быть загружено в аккумулятор accumulator 2, в виде положительного числа формата INT, выровненного вправо. Если число позиций равно нулю, то операция циклического сдвига не выполняется (нет операции, т.е. NOP); если число позиций равно 32, то содержимое аккумулятора accumulator 1 не изменяется, а бит состояния CC1 принимает состояние бита, перемещенного последним (бит 31). Если указанное число позиций равно 33, то содержимое аккумулятора accumulator 1 смещается на 1 разряд, если указанное число позиций равно 34, то содержимое смещается на 2 позиции и так далее.

Циклический сдвиг влево через бит состояния CC1

RLDA Циклический сдвиг влево через бит состояния CC1 на 1 разряд

Операция циклического сдвига RLDA позволяет сдвигать на 1 бит влево данные, находящиеся во всех битах аккумулятора accumulator 1. При этом вновь освобожденный при сдвиге содержимого разряд (бит 0) заполняется значением из бита состояния CC1, а бит состояния CC1, в свою очередь принимает состояние бита, который был "вытолкнут" из аккумулятора последним (бит 31). При этом бит состояния CC0 устанавливается в состояние "0".

Циклический сдвиг вправо через бит состояния CC1

RRDA Циклический сдвиг вправо через бит состояния CC1 на 1 разряд

Операция циклического сдвига RRDA позволяет сдвигать на 1 бит вправо данные, находящиеся во всех битах аккумулятора accumulator 1. При этом вновь освобожденный при сдвиге содержимого разряд (бит 31) заполняется значением из бита состояния CC1, а бит состояния CC1, в свою очередь принимает состояние бита, который был "вытолкнут" из аккумулятора последним (бит 0). При этом бит состояния CC0 устанавливается в состояние "0".

14 Логические функции для слов данных (Word Logic)

Логические функции для слов данных (Word Logic) позволяют побитно комбинировать значение, находящееся в аккумуляторе accumulator 1, с константой или с содержимым аккумулятора accumulator 2. Результат выполнения операции сохраняется в аккумуляторе accumulator 1. Логические функции могут выполняться как для данных формата Word (слов), так и для данных фомата Duobleword (двойных слов).

Пользователю доступны следующие логические операции для слов данных:

- AND логическая операция "И",
- OR логическая операция "ИЛИ",
- Exclusive OR логическая операция "Исключающее ИЛИ".

В этой главе будут рассмотрены логические функции для слов данных (Word Logic), используемые в языке программирования STL. В языке программирования SCL выполнение логических функций обеспечивается с помощью программирования соответствующих логических выражений (см. раздел 27.4.3 "Логические операции").

Информацию о том, каким образом логические функции устанавливают биты состояния Вы можете получить из главы 15 "Биты состояния". Примеры, рассматриваемые в данной главе, Вы можете найти на прилагаемой дискете в библиотеке STL_Book library в разделе "Digital Functions" ("Функции для обработки чисел") в функциональном блоке FB 114 или в исходном файле Chap_14.

14.1 Выполнение логических операций для слов данных

Вы можете программировать операции двумя способами в соответствии со следующими общими схемами:

```
Загрузка (load) адреса Address1;
Загрузка (load) адреса Address2;
Логическая операцияции для слова данных без константы;
Передача (transfer) результата Result;
```

```
Загрузка (load) адреса Address;
Логическая операцияции для слова данных с константой;
Передача (transfer) результата Result;
```

Логические функции для слов данных всегда выполняются вне всякой связи с какими-либо условиями. При этом эти функции не влияют на результат логической операции (RLO).

Формирование результата логической операции для слов данных

Логические функции для слов данных формируют результат операции бит за битом таким же образом, как и функции двоичной логики, описанные в главе 4 "Двоичные логические операции" (табл.14.1).

Таблица 14.1 Формирование результата логической операции для слов данных

Содержимое бита в аккумуляторе 2 или бита в константе	0	0	1	1
Содержимое бита в аккумуляторе 1	0	1	0	1
Результат AW, AD	0	0	0	1
Результат OW, OD	0	1	1	1
Результат XOW, XOD	0	1	1	0

Логическая функция комбинирует бит 0 аккумулятора accumulator 1 с битом 0 аккумулятора accumulator 2 или константы, указанной в инструкции. Результат сохраняется в бите 0 аккумулятора accumulator 1. Таким же образом эта же логическая операция выполняется для битов 2, битов 3 и так далее вплоть до старшего бита операндов (до бита 15 в операциях для слов и до бита 31 в операциях для двойных слов). Содержимое аккумулятора accumulator 2 при этом остается неизменным.

Логические функции для слов данных с содержимым accumulator 2

Перед инструкцией логической функции для слов данных должны следовать две операции загрузки, для того, чтобы можно было комбинировать загружаемые значения. После выполнения логической функции результат сохраняется в аккумуляторе accumulator 1.

Пример:

```
L     MW 142;        //Адрес 1
L     MW 144;        //Адрес 2
AW    ;              //Логическая операция
T     MW 146;        //Результат
```

В этом примере логическая функция выполняется с данными формата слово (Word).

Логические функции для слов данных с константой

Перед инструкцией логической функции для слов данных должна следовать операция загрузки, для того, чтобы можно было комбинировать загружаемое значение с константой, которая в свою очередь задается в операторе функции. Результат выполнения логической функции сохраняется в аккумуляторе accumulator 1.

Пример:

```
L     MW 148;
AW    W#16#807F;
T     MW 150;
L     MD 152;
OD    DW#16#8000_F000;
T     MD 156;
```

В этом примере логическая функция выполняется с данными формата двойное слово (Duobleword).

Выполнение логических операций с данными формата слово (Word)

Логические функции, выполняемые с данными формата слово (Word), воздействуют только на данные, находящиеся в младшем слове аккумулятора accumulator 1 (т.е. в битах с 0 по 15). При этом биты, находящиеся в старшем слове аккумулятора (биты с 16 по 31), остаются без изменения (см. рис. 14.1).

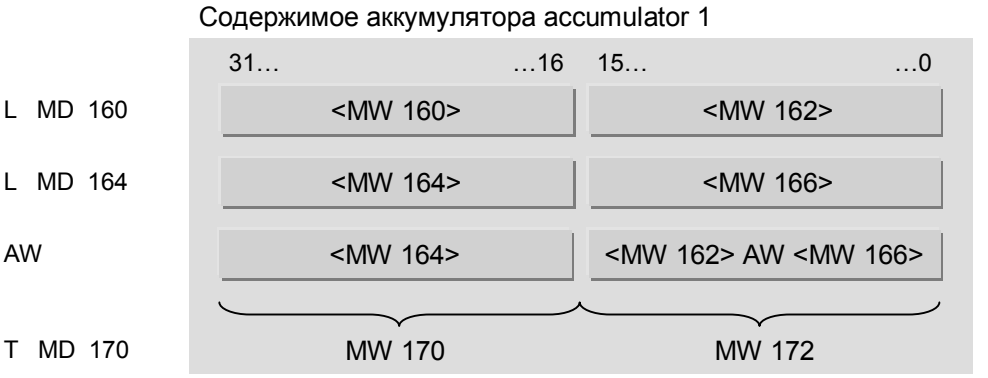


Рис.14.1 Выполнение логических операций с данными формата слово (Word)

Последовательное выполнение логических операций

После завершения логической операции для слов данных Вы можете немедленно перейти к выполнению следующей логической операции (загрузив предварительно данные посредством операции load из соответствующего адреса или задав константу как параметр в инструкции) без необходимости сохранения промежуточного результата (например, в области локальных данных). Аккумуляторы сами служат для временного хранения данных.

Пример:

```
L     Value1;
L     Value2;
AW    ;
L     Value3;
OW    ;
T     Result1;
```

В вышеуказанном примере результат выполнения инструкции AW содержится в аккумуляторе accumulator 1. Во время загрузки значения Value3 этот результат смещается в аккумулятор accumulator 2. Теперь оба значения могут быть обработаны в соответствии с инструкцией OW.

Пример:

```
L     Value4;
L     Value5;
XOW   ;
AW    W#16#FFF0;
T     Result2;
```

В данном примере результат выполнения операции XOW содержится в аккумуляторе accumulator 1. Биты с 0 по 3 аккумулятора сбрасываются в состояние "0" при выполнении операции AW.

В таблице 14.2 показаны примеры по одному для каждого типа логических операций.

Таблица 14.2 Примеры применения логических функций для слов данных

Логическая операция AND (логическое И)	Четыре старших бита слова MW 138 сбрасываются в"0"; результат сохраняется в слове MW 140.	`L   MW 138;`
		`AW  W#16#0FFF;`
		`T   MW 140;`

Логическая операция OR (логическое ИЛИ)	Переменные "WlogicVal1" и "WlogicVal1" побитно обрабатываются функцией OR (ИЛИ); результат сохраняется в переменной "WlogicReslt".	`L   "Global_DB".WlogicVal1;`
		`L   "Global_DB".WlogicVal2;`
		`OD ;`
		`T`
		`"Global_DB".WlogicReslt;`

Логическая операция Exclusive OR (Исключающее ИЛИ)	Переменные #Input и #Mask обрабатываются функцией Exclusive OR (Исключающее ИЛИ); результат сохраняется в переменной #Buffer.	`L   #Input;`
		`L   #Mask;`
		`XOW ;`
		`T   #Buffer;`

14.2 Описание логических операций для слов данных

Операция AND (И) для слов данных

AW		Логическая операция AND (И) для данных формата Word (слово) в accum1 и accum2
AW	W#16#	Логическая операция AND (И) для данных формата Word (слово) в accum1 и константе
AD		Логическая операция AND (И) для данных формата Duobleword (двойное слово) в accum1 и accum2
AD	DW#16#	Логическая операция AND (И) для данных формата Duobleword (двойное слово) в accum1 и константе

Логическая функция для обработки двух чисел в соответствии с логикой AND (И) позволяет последовательно бит за битом комбинировать данные, находящиеся в аккумуляторе accumulator 1, с соответствующими битами аккумулятора accumulator 2 или с соответствующими битами константы, указанной в инструкции функции. Бит, находящийся в слове результата, будет установлен в состояние "1" только в случае, если оба сравниваемых бита исходных данных имеют значение "1".

Так как те биты в аккумуляторе accumulator 2 или в константе, которые имеют значение "0", сбрасывают в "0" соответствующие биты слова результата вне зависимости от состояния соответствующих битов в аккумуляторе accumulator 1, то говорят, что эти биты "маскированы" ("masked"). Этот, так называемый, эффект маскирования битов и является основным назначением логических AND (И) операций для слов данных.

Операция OR (ИЛИ) для слов данных

OW		Логическая операция OR (ИЛИ) для данных формата Word (слово) в accum1 и accum2
OW	W#16#	Логическая операция OR (ИЛИ) для данных формата Word (слово) в accum1 и константе
OD		Логическая операция OR (ИЛИ) для данных формата Duobleword (двойное слово) в accum1 и accum2
OD	DW#16#	Логическая операция OR (ИЛИ) для данных формата Duobleword (двойное слово) в accum1 и константе

Логическая функция для обработки двух чисел в соответствии с логикой OR (ИЛИ) позволяет последовательно бит за битом комбинировать данные, находящиеся в аккумуляторе accumulator 1, с соответствующими битами аккумулятора accumulator 2 или с соответствующими битами константы, указанной в инструкции функции. Бит, находящийся в слове результата, будет сброшен в состояние "0" только в случае, если оба сравниваемых бита исходных данных имеют значение "0".

Так как те биты в аккумуляторе accumulator 2 или в константе, которые имеют значение "1", устанавливают в "1" соответствующие биты слова результата вне зависимости от состояния соответствующих битов в аккумуляторе accumulator 1, то говорят, что эти биты "маскированы" ("masked"). Этот, так называемый, эффект маскирования битов и является основным назначением логических OR (ИЛИ) операций для слов данных.

Операция Exclusive OR (Исключающее ИЛИ) для слов данных

XOW		Операция Exclusive OR (Исключающее ИЛИ) для данных формата Word (слово) в accum1 и accum2
XOW	W#16#	Операция Exclusive OR (Исключающее ИЛИ) для данных формата Word (слово) в accum1 и константе
XOD		Операция Exclusive OR (Исключающее ИЛИ) для данных Duobleword (двойное слово) в accum1 и accum2
XOD	DW#16#	Операция Exclusive OR (Исключающее ИЛИ) для Duobleword (двойное слово) в accum1 и константе

Логическая функция для обработки двух чисел в соответствии с логикой Exclusive OR (Исключающее ИЛИ) позволяет последовательно бит за битом комбинировать данные, находящиеся в аккумуляторе accumulator 1, с соответствующими битами аккумулятора accumulator 2 или с соответствующими битами константы, указанной в инструкции функции. Бит, находящийся в слове результата, будет установлен в состояние "1" только в случае, если только один из сравниваемых битов исходных данных имеет значение "1". Если некоторый бит в аккумуляторе accumulator 2 или в константе, имеет значение "1", то соответствующий бит слова результата в соответствии с логикой функции будет иметь инвертированное значение по отношению к предыдущему значению этого бита в аккумуляторе accumulator 1.

В результате те биты в аккумуляторе accumulator 2 или в константе, которые имеют значение "1", устанавливают в "1" соответствующие биты слова результата вне зависимости от состояния соответствующих битов в аккумуляторе accumulator 1, то говорят, что эти биты "маскированы" ("masked"). Этот, так называемый, эффект маскирования битов и является основным назначением логических Exclusive OR (Исключающее ИЛИ) операций для слов данных.

Управление выполнением программы

STEP 7 обеспечивает пользователя различными средствами для управления ходом выполнения программы. Вы можете выходить из выполняемой линейной программы внутри блока, Вы можете также создать определенную структуру программы посредством вызовов параметрируемых блоков. Вы можете воздействовать на выполнение программы, исходя из текущих значений параметров, рассчитанных во время работы программы, в зависимости параметров процесса или в связи с состоянием установки.

Биты состояния (status bits) призваны обеспечивать информацией, зависящей от результатов выполнения арифметических и математических функций, а также от возникающих ошибок (например, из-за нарушения границ диапазона допустимых значений при расчетах). Вы можете использовать состояния сигналов этих битов в Вашей программе непосредственно, проверяя их с помощью двоичных логических операций.

Функции перехода (jump functions) могут использоваться для организации ветвления программы пользователя. Функции перехода могут быть либо безусловные, либо условные, (по условию определяемому отдельным битом состояния, результатом логической операции RLO или двоичным результатом). В STL можно легко запрограммировать переходы в рассчетные точки программы (распределитель переходов) или выполнить программные циклы (циклический переход).

Главное управляющее реле (Master Control Relay - MCR) обеспечивает еще один способ управления выполнением программы. Изначально разработанный для систем управления с помощью реле, язык STL обеспечивает программную (компьютерную) версию программируемого способа управления.

Функции блоков (block functions) позволяют пользователю структурировать программу. Вы можете снова и снова использовать функции и функциональные блоки, определяя **параметры блоков**.

Более подробную информацию о программировании блоков на STL Вы можете найти в разделе 3.4 "Программирование кодовых блоков на STL". Главы 18 "Функции блоков" и 19 "Параметры блоков" продолжают эту тему. Информацию, касающуюся этих же вопросов для языка SCL, Вы можете отыскать в разделе 3.5 "Программирование кодовых блоков на SCL" и в главе 29 "SCL блоки". Глава 26 "Прямой доступ к переменным" содержит дополнительную информацию по параметрам блоков, такую, например, как сохранение параметров в памяти и использование параметров для сложных типов данных.

15 Биты состояния (Status bits)

Биты состояния RLO, BR, CC0, CC1 и превышения (overflow); проверка битов состояния; слово состояния (status word); EN/ENO.

16 Функции перехода (Jump functions)

Безусловные переходы; переходы по условию для RLO, BR, CC0, CC1 и превышения (overflow); распределитель переходов ("jump distributor"); циклический переход (loop jump).

17 Главное управляющее реле (Master Control Relay - MCR)

Зависимость от MCR; MCR-диапазон; MCR-зона.

18 Функции блоков

Тип блоков, вызовы блоков, окончания блоков; статические локальные данные; обработка блоков данных, регистр блока данных, обработка адресов данных.

19 Параметры блоков

Объявление параметров; формальные параметры и фактические параметры; передача параметров в вызываемые блоки; примеры: ленточный конвейер, счетчик деталей и загрузка.

15 Биты состояния (Status Bits)

Биты состояния - это двоичные флаги (индикаторные биты). CPU использует их для управления двоичными логическими операциями. CPU устанавливает биты состояния при выполнении операций обработки чисел. Вы можете проверять состояние этих битов (например, для выяснения результата выполнения вычислений); Вы можете также влиять на отдельные биты. Биты состояния скомпонованы в слово, в "слово состояния" ("status word").

Примеры, рассматриваемые в данной главе, Вы можете найти на прилагаемой дискете в библиотеке STL_Book library в разделе "Program Flow Control" ("Управление выполнением программы") в функциональном блоке FB 115 или в исходном файле Chap_15.

15.1 Описание битов состояния

В таблице 15.1 представлены биты состояния, доступные при STL-программировании.

Таблица 15.1 Биты состояния

Бит	**Двоичные флаги (binary flags)**	
0	/FC	Первичный опрос (first check)
1	RLO	Результат логической операции
2	STA	Состояние (статус - "status")
3	OR	Бит состояния OR (OR status bit)
8	BR	Двоичный результат (binary result)
Бит	**Числовые флаги (digital flags)**	
4	OS	Для сохранения информации о переполнении (stored overflow)
5	OV	Переполнение (overflow)
6	CC0	Условный код (condition code)
7	CC1	Условный код (condition code)

Первый столбец показывает номер бита в слове состояния. CPU использует "двоичные флаги" ("binary flags") для управления двоичными функциями; "числовые флаги" ("digital flags") используются прежде всего для индикации результатов арифметических и математических функций.

Первичный опрос (first check)

Бит состояния /FC управляет двоичными логическими операциями внутри логического устройства управления (часть АЛУ). Двоичный логический шаг всегда начинается с двоичной инструкции проверки (первичного опроса) при /FC = "0", как показано в описании двоичных логических операций. Первичный опрос устанавливает /FC в состояние "1" (/FC = "1"). Двоичный логический шаг заканчивается присвоение двоичного значения, или условным переходом, или изменением блока. Это сбрасывает бит состояния: /FC = "0". Следующая двоичная проверка (опрос) начинается с новой двоичной логической функции.

Результат логической операции (RLO)

Бит состояния RLO является промежуточным буфером в двоичных логических операциях. При первичном опросе CPU передает результат опроса в RLO, комбинирует результат опроса с хранящимся в RLO значением при каждом последующем опросе и затем сохраняет результат в RLO (как описано в главе 4 "Двоичные логические операции"). Вы можете также устанавливать, сбрасывать или инвертировать значение в RLO непосредственно или сохранять его в BR. Функции счетчика, таймера и операции с памятью управляются с использованием RLO, как и функции перехода.

Состояние (status)

Состояние бита STA соответствует состоянию сигнала указанного двоичного разряда (двоичного адреса) или проверяемого "условного бита" для двоичной логической операции (A, AN, O, ON, X, XN).

В случае операций с памятью (S, R, =) значение бита состояния STA соответствует записанному в память значению или (в случае отсутствия операции записи в память, например, при RLO = "0" или когда главное управляющее реле MCR активно) значение бита STA соответствует значению адресованного (но неизмененного) двоичного адреса.

При проверке наличия фронта сигнала FP или FN значение RLO до операции проверки фронта сохраняется в бите состояния STA. Все остальные двоичные операции устанавливают STA (STA = "1"), как и переходы, зависящие от состояния двоичного флага: JC, JCN, JBI, JNBI (исключение: CLR сбрасывает бит STA: STA = "0").

Бит состояния STA не влияет на обработку операторов STL-операторов. Состояние этого бита отображается тестовыми функциями с помощью программатора PG (такой функцией, например, как функция отображения состояния программы "program status"). Таким образом, Вы можете использовать этот бит состояния для трассировки двоичных логических последовательностей или в целях отладки программы.

Бит состояния OR (OR status bit)

Бит OR status сохраняет результат выполненной (имеется в виду, что условие функции выполнено) двоичной логической операции AND (И) ("1")

и показывает последующей операции OR (ИЛИ), что результат уже зафиксирован (при комбинированной операции OR (ИЛИ) внутри операции AND (И) перед выполнением операции OR (ИЛИ)). Все остальные двоичные операторы сбрасывают бит OR status.

В таблице 15.2 в графах "Двоичные флаги (Binary flags)" приводится пример двоичного логического шага для иллюстрации действия двоичных флагов. Двоичный логический шаг начинается с первичного опроса после операции с памятью и заканчивается с последней операцией с памятью до функции проверки.

Таблица 15.2 Пример влияния битов состояния

STL-операторы	Двоичные флаги (binary flags)				Примечания
	/FC	RLO	STA	OR	
...					
= M 10.0	0	x	x		
A I 4.0	1	1	1	0	I 4.0 сод. "1" Окрашенная
A I 4.1	1	1	0	0	I 4.1 сод. "0" область
O	1	1	1	1	соответствует
O I 4.2	1	1	0	0	I 4.2 сод. "0" двоичному
ON I 4.3	1	1	1	0	I 4.3 сод. "1" логическому
= Q 8.0	0	1	1	0	Q 8.0 --> "1" шагу
R Q 8.1	0	1	0	0	Q 8.1 --> "0"
S Q 8.2	0	1	1	0	Q 8.2 --> "1"
A I 5.0	1	x	x		
...					

STL-операторы	Числовые флаги (digital flags)				Примечания
	CC0	CC1	OV	OS	
...					
T MW 10	x	x	x	x	
L +12	x	x	x	x	
L +15	x	x	x	x	
-I	1	0	0	0	результат отрицательный
L +20000	1	0	0	0	
*I	0	1	1	1	переполнение: OV и OS -> "1"
L +20	0	1	1	1	
+I	0	1	0	1	OV --> "0" OS без изм.="1"
T MW 30	0	1	0	1	
L MW 40	1	1	0	1	
...					

Переполнение (Overflow)

Бит состояния OV индицирует факт превышения диапазона допустимых численных значений (переполнения) или факт использования некорректных действительных (REAL) чисел. На состояние бита OV влияют следующие функции: арифметические, математические, отдельные функции преобразования, функции сравнения действительных (REAL) чисел.

Вы можете проверить состояние бита OV с помощью операций проверки или посредством оператора перехода JO.

Сохранение информации о превышении граничных значений (Stored overflow)

Бит состояния OS фиксирует факт превышения (дублирует и сохраняет установленное состояние бита OV). Когда CPU устанавливает бит состояния OV, он также устанавливает бит состояния OS. При этом, если бит состояния OV в дальнейшем может быть сброшен при выполнении соответствующей операции, то бит OS сохранит свое состояние, т.е. информацию о состоявшемся факте превышения. Это позволит Вам выполнять альтернативную проверку бита состояния, в том числе и отложенную во времени (в более поздней точке программы), для определения факта превышения диапазона допустимых численных значений или факта использования некорректных действительных (REAL) чисел.

Вы можете проверять состояние бита OS посредством операторов проверки (опроса) или с помощью оператора перехода JOS. Оператор перехода JOS или смена блока сбрасывает бит состояния OS.

Биты состояния CC0, CC1 (биты "условных кодов") (condition codes)

Биты состояния CC0, CC1 (условные биты) призваны обеспечивать информацией о результатах выполнения функций сравнения, арифметических и математических функций, логических операций для слов данных или о состоянии "вытолкнутых" битов в случае выполнения операций сдвига.

Вы можете проверять состояние всех числовых флагов с помощью операторов перехода или посредством операторов проверки (опроса) (см. далее в этой главе). В нижней части таблицы 15.2 показаны примеры установки числовых флагов (digital flags).

Двоичный результат (binary result)

Бит состояния BR (двоичный результат) помогает реализовать механизм EN/ENO для вызовов блоков (в сочетании с графическими языками). В разделе 15.4 "Использование двоичного результата" показано, как STEP 7 использует бит двоичный результат. Вы можете также установить или сбросить бит состояния BR или проверить его состояние с помощью операторов проверки (опроса) или посредством операторов перехода.).

Слово состояния (status word)

Слово состояния (status word) содержит в себе все биты состояния. Вы можете загружать слово состояния в аккумулятор accumulator 1, а также считать его значение из этого аккумулятора.

```
L    STW;    //загрузить слово состояния
T    STW;    //загрузить в слово состояния
```

См. в главе 6 "Функции пересылки данных" о том, как используются операторы загрузки (load) и выгрузки (transfer). В таблице 15.1 представлены биты состояния в слове состояния с указанием их назначения.

Вы можете использовать слово состояния для проверки битов состояния или для их установки в соответствии с Вашими требованиями. Таким образом, Вы можете сохранить текущее значение слова состояния или начать выполнение программного блока с требуемыми значениями битов состояния.

Надо учитывать, что S7-300 CPU (кроме CPU 318) не загружают биты состояния /FC, STA и OR в аккумулятор; аккумулятор в соответствующих этим битам позициях содержит "0".

15.2 Описание битов состояния

Функции обработки чисел влияют на биты состояния CC0, CC1, OV и OS, как показано в таблице 15.3. Отдельные операторы STL влияют на биты состояния RLO и BR.

Биты состояния при вычислениях с INT- и DINT-числами

Арифметические функции при использовании данных форматов INT и DINT могут устанавливать все числовые флаги ("digital flags" - биты состояния). В случае результата, равного нулю, биты CC0 и CC1 сбрасываются в "0". Сочетание CC0 = "0", а CC1 = "1" сообщает о положительном результате. Сочетание CC0 = "1", а CC1 = "0" сообщает об отрицательном результате. Превышение (overflow) пределов диапазона допустимых значений устанавливает биты OV и OS (примечание: при других значениях битов состояния CC0 и CC1). Деление на ноль приводит к установлению в состояние "1" всех "числовых битов состояния" ("digital status bits").

Биты состояния при вычислениях с REAL-числами

Арифметические функции при использовании данных формата REAL могут устанавливать все "числовые биты состояния" ("digital status bits"). В случае результата, равного нулю, биты CC0 и CC1 сбрасываются в "0". Сочетание CC0 = "0", а CC1 = "1" сообщает о положительном результате. Сочетание CC0 = "1", а CC1 = "0" сообщает об отрицательном результате. Превышение (overflow) пределов диапазона допустимых значений устанавливает биты OV и OS (примечание: при других значениях битов состояния CC0 и CC1). При обработке некорректного действительного (REAL) числа происходит установка в состояние "1" всех числовых битов состояния.

Действительное (REAL) число относится к "ненормированным" (или "ненормализованным" - "denormalized"), если оно представляется с уменьшенной точностью. Абсолютное значение "ненормированного" ("denormalized") числа формата REAL меньше чем $1.175494 \cdot 10^{-38}$ (см. главу 24 "Типы данных"). S7-300 интерпретируют ненормированные числа формата REAL как числа, равные нулю.

Таблица 15.3 Установка битов состояния

Вычисления с INT-данными				
Результат	CC0	CC1	OV	OS
<-32768 (+I, -I)	0	1	1	1
<-32768 (*I)	1	0	1	1
-32768 ... -1	1	0	0	-
0	0	0	0	-
+1 ... +32767	0	1	0	-
>+32767 (+I, -I)	1	0	1	1
>+32767 (*I)	0	1	1	1
32768 (/I)	0	1	1	1
(-)65536	0	0	1	1
Деление на 0	1	1	1	1

Вычисления с DINT-данными				
Результат	CC0	CC1	OV	OS
<-2147483648 (+D, -D)	0	1	1	1
<-2147483648 (*D)	1	0	1	1
-2147483648 ... -1	1	0	0	-
0	0	0	0	-
+1... +2147483647	0	1	0	-
>+2147483647 (+D, -D)	1	0	1	1
>+2147483647 (*D)	0	1	1	1
2147483648 (/D)	0	1	1	1
(-)4294967296	0	0	1	1
Деление на 0 (/D, MOD)	1	1	1	1

Вычисления с REAL-данными				
Результат	CC0	CC1	OV	OS
+ нормирован	0	1	1	1
± ненормирован	1	0	1	1
± ноль	1	0	0	-
- нормирован	0	0	0	-
+ бесконечность (деление на 0)	0	1	0	-
- бесконечность (деление на 0)	1	0	1	1
± некорр. REAL	0	1	1	1

Операции сравнения				
Результат	CC0	CC1	OV	OS
равен	0	0	0	-
больше чем	0	1	0	-
меньше чем	1	0	0	-
некорр. REAL	1	1	1	1

Преобразование данных NEG_I				
Результат	CC0	CC1	OV	OS
+1 ... +32767	0	1	0	-
0	0	0	0	-
-32768 ... -1	1	0	0	-
(-) 32768	1	0	1	1

Преобразование данных NEG_D				
Результат	CC0	CC1	OV	OS
+1... +2147483647	0	1	0	-
0	0	0	0	-
-2147483648 ... -1	1	0	0	-
(-) 2147483648	1	0	1	1

Функции сдвига				
Результат	CC0	CC1	OV	OS
"0"	0	0	0	-
"1"	0	1	0	-
с числом поз. 0	-	-	-	-

Логические операции для слов данных				
Результат	CC0	CC1	OV	OS
ноль	0	0	0	-
не ноль	0	1	0	-

Биты состояния при работе с функциями преобразования

Из функций преобразования функции инвертирования влияют на все "числовые биты состояния" ("digital status bits"). Кроме того, следующие функции преобразования устанавливают биты состояния OV и OS в случае появления ошибок: (выход за пределы диапазона допустимых значений и обработка некорректного числа формата REAL):

- ITB и DTB: преобразование числа формата INT в формат BCD
- RND+, RND-, RND, TRUNK: преобразование числа формата REAL в формат DINT

Биты состояния при работе с функциями сравнения

Функции сравнения устанавливают биты состояния CC0 и CC1. Флаги устанавливаются независимо от типа выполняемой функции сравнения; зависят они только от отношения между двумя значениями, указанными в выражении функции сравнения. Функция сравнения чисел формата REAL производит проверку корректности данных формата REAL.

Биты состояния при работе с логическими операциями для слов и с функциями сдвига

Логические операции для слов и функции сдвига устанавливают биты состояния CC0 и CC1. Бит OV находится в сброшенном состоянии ("0").

Установка и сброс RLO

Оператор SET устанавливает RLO в состояние "1", а оператор CLR сбрасывает RLO в состояние "0". Одновременно происходит установка в "1" или сброс в "0" бита состояния STA. Рассматриваемые операторы выполняются без всяких условий.

Операторы SET и CLR устанавливают также биты состояния OR и /FC, что означает, что после операторов SET или CLR новая логическая операция начинается со следующей операции проверки (опроса).

Вы можете запрограммировать сброс или установку бита (двоичного адреса) с помощью оператора SET:

```
SET   ;
S     M 8.0;     //Установка меркера
R     M 8.1;     //Сброс меркера
CLR   ;
S     C 1;       //Сброс меркера фронта для установки
                 //счетчика "Set counter"
```

Прямая установка и сброс RLO также полезны при использовании с таймерами и счетчиками. Для запуска таймера и счетчика Вам необходимо изменить RLO со значения "0" на значение "1" (помните также, что Вам еще требуется для этого положительный фронт сигнала разрешения). В разделе программы, в котором преобладают числовые логические операции, RLO обычно не определяется, например, при последующей функции перехода для проверки "числовых флагов" (битов состояния -"digital flags"). При этом Вы можете использовать операторы SET и CLR для определенных установок и сбросов RLO или для программного изменения RLO.

Об инвертировании RLO с NOT см. гл. 4 "Двоичные логические операции".

Установка и сброс BR

С помощью оператора SAVE Вы можете сохранить в бите состояния "двоичный результат" (BR - "binary result"). Оператор SAVE пересылает состояние сигнала RLO в бит состояния BR. Оператор SAVE выполняются без всяких условий; при этом он не влияет на другие биты состояния.

```
SET        ;
SAVE       ;       //Установка бита BR в "1"
...
AN         OV;
SAVE       ;       //Сброс бита BR в "0" при превышении
...
```

15.3 Проверка битов состояния

Биты состояния RLO и BR и все числовые флаги (digital flags) могут быть проверены с помощью операций двоичного опроса и функций перехода. Также можно обрабатывать биты состояния после загрузки в аккумулятор слова состояния.

Проверка битов состояния с помощью операции двоичного опроса

Вы можете использовать все операторы опроса (проверки), описанные в главе 4 "Двоичные логические операции", для проверки числовых флагов (digital flags) и двоичного результата BR (см. ниже).

A	-	Проверка (опрос) выполнения условия и логическое AND (И)
O	-	Проверка (опрос) выполнения условия и логическое OR (ИЛИ)
X	-	Проверка (опрос) выполнения условия и Exclusive OR (исключающее ИЛИ)
AN	-	Проверка (опрос) невыполнения условия и логическое AND (И)
ON	-	Проверка (опрос) невыполнения условия и логическое OR (ИЛИ)
XN	-	Проверка (опрос) невыполнения условия и Exclusive OR (исключающее ИЛИ)

>0	Результат больше 0	[(CC0=0) & (CC1=1)]
>=0	Результат больше или равен 0	[(CC0=0)]
<0	Результат меньше 0	[(CC0=1) & (CC1=0)]
<=0	Результат меньше или равен 0	[(CC1=0)]
<>0	Результат не равен 0	[(CC0=1) & (CC1=0)] v [(CC0=0) & (CC1=1)]
==0	Результат равен 0	[(CC0=0) & (CC1=0)]
UO	Результат не верен	[(CC0=0) & (CC1=1)]
OV	Переполнение (overflow)	[(OV=1)]
OS	Сохраненное переполнение	[(OS=1)]
BR	Двоичный результат	

При этом принципы использования операторов опроса (проверки) такие же, как и при проверке, например, входов.

Проверка битов состояния с помощью функций перехода

Вы можете проверять биты состояния RLO и BR, все комбинации CC0 и CC1, а также биты состояния OV и OS с помощью соответствующих функций перехода (табл. 15.4). В главе 16 "Функции перехода" содержится подробное описание.

Таблица 15.4 Проверка битов состояния с помощью функций перехода

RLO	BR	CC0	CC1	OV	OS	Функции перехода
1	-	-	-	-	-	JC, JCB
0	-	-	-	-	-	JCN, JNB
-	1	-	-	-	-	JBI
-	0	-	-	-	-	JNBI
-	-	0	0	-	-	JZ, JMZ, JPZ
-	-	0	1	-	-	JN, JP, JPZ
-	-	1	0	-	-	JN, JM, JMZ
-	-	1	1	-	-	JUO
-	-	-	-	1	-	JO
-	-	-	-	-	1	JOS

Замечания по проверке битов состояния OV и OS

Если результат вычисления выходит за пределы диапазона допустимых значений для определенного типа данных, то бит состояния OV устанавливается в "1"; одновременно устанавливается в "1" бит состояния OS (бит для сохранения информации о том, что имело место нарушение границы диапазона допустимых значений).

Если результат следующей далее функции (например, в цепочке вычислений) находится в пределах допустимых значений, флаг OV будет сброшен. Однако, при этом флаг OS останется установленным, так что тот факт, что имело место превышения границ разрешенного диапазона значений, останется сохраненным, при этом он может быть обнаружен с помощью проверки (опроса) в конце вычислений.

Флаг OS не может быть сброшен, пока не встретится функция перехода JOS или не сменится блок (т.е., пока не произойдет новый вызов блока или пока не закончится обработка текущего блока).

Далее представлены два варианта программного решения задачи проверки состояния флагов для определения того факта, что имело место превышение пределов диапазона допустимых значений - в первой программе выполняется двоичный опрос (проверка) флагов, а во второй - выполняется проверка флагов с помощью условных переходов.

```
Двоичный опрос
L     Value1;
L     Value2;
+I    ;
A     OV;           //Отдельная проверка
=     Status1;
L     Value3;
+I    ;
A     OV;           //Отдельная проверка
=     Status2;
L     Value4;
+I    ;
A     OS;           //Отдельная проверка
=     Status_overall;
T     Result;

Функции перехода
L     Value1;
L     Value2;
+I    ;
JO    ST1;          //Отдельная проверка
L     Value3;
+I    ;
JO    ST2;          //Отдельная проверка
L     Value4;
+I    ;
JOS   STOV;         //Общая проверка
T     Result;
```

Как видно из последних примеров, опрос (проверку) флагов можно производить либо после каждой операции вычисления (опрос бита состояния OV), либо в конце цепочки вычислений (опрос бита состояния OS).

15.4 Использование двоичного результата (бита состояния BR)

STEP 7 использует бит BR (двоичный результат) для реализации механизма EN/ENO при использовании языков программирования LAD ("контактный план" - ["ladder diagram"]) и FBD ("Функциональная блок-схема" - ["functional block diagram"]).

Вы можете пропустить эту информацию, если Вы используете только язык программирования STL; тогда в Вашем распоряжении двоичный результат BR как дополнительный бит RLO.

Тем не менее, для индикации ошибок при обработке блоков (как это используется в системных блоках SFB и SFC и некоторых стандартных блоках) Вы можете использовать BR как флаг групповой ошибки, даже если Вы используете исключительно программирование на STL.

Механизм EN/ENO

При использовании языков программирования LAD и FBD все функциональные элементы имеют разрешающие входы EN и разрешающие выходы ENO. Если на разрешающем входе EN присутствует единица "1", то функциональный элемент активизируется (выполняется его функция). Если функция выполняется корректно, то на разрешающем выходе ENO также присутствует единичный сигнал "1". Если происходит ошибка во время выполнения функции (например, переполнение при вычислении арифметической функции), выход ENO сбрасывается в состояние "0". Если на разрешающем входе EN присутствует сигнал "0", то на разрешающем выходе ENO также присутствует сигнал "0".

Вы можете использовать такие свойства разрешающих входов/выходов (EN/ENO) при связывании нескольких функциональных элементов в единую цепь; при этом разрешающий выход ENO предыдущего элемента будет управлять разрешающим входом EN последующего функционального элемента (см. рис. 15.1). Такая конфигурация позволит сделать возможным, чтобы происходил разрыв ("switch off") всей цепи (чтобы не был активен ни один функциональный элемент, если в примере на входе I 1.0 присутствует сигнал "0") или чтобы при появлении ошибки в одном из функциональных элементов последующие за ним элементы не могли быть активизированы.

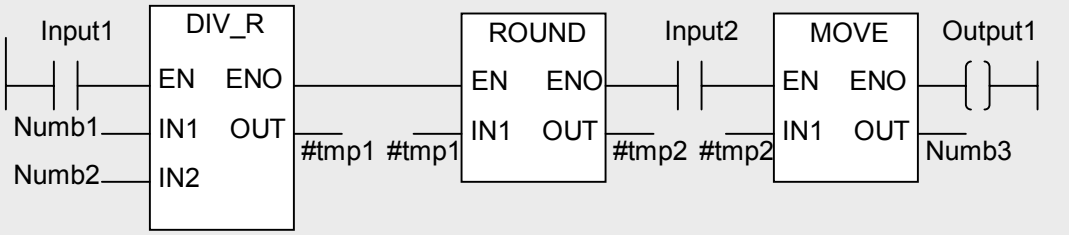
Вход EN и выход ENO не являются параметрами блока, однако, инструкции программы приводят к тому, что LAD/FBD редактор сам генерирует предыдущие и последующие функциональные элементы (даже в случае применения функций и функциональных блоков). В этом случае LAD/FBD редактор использует двоичный результат для хранения состояния сигнала на разрешающем входе EN во время обработки блока или в качестве сигнального бита для индикации ошибки в функциональном элементе.

Вы можете найти программу, указанную на рис. 15.1, в сегменте Network 8 блока FB 115 в программе "Program Flow Control" (в библиотеке STL_Book). Если Вы видите указанный сегмент блока FB 115 на экране монитора, Вы можете переключиться на LAD-представление, выбрав опции: View -> LAD. После этого редактор включит графическое отображение LAD-программы.

Если Вы пишите свои собственные функции или функциональные блоки и хотите их использовать, например, с LAD- или FBD-представлением, Вы должны таким образом управлять BR, чтобы этот бит сбрасывался в "0" при обнаружении ошибки (см. ниже).

Сообщение о групповой ошибке в блоках

Вы можете использовать двоичный результат в качестве сигнального бита для сообщения о групповой ошибке в блоках. Если блок обрабатывается без ошибок, бит BR устанавливается в состояние "1", иначе - BR сбрасывается в состояние "0".




```
        A(
        A(
        A(
        A     Input1
        JNB   M001     //Опрос входа EN
        L     Numb1
        L     Numb2    //функционального элемента
        /R
        T     tmp1
        AN    OV       //Проверка ошибки в элементе
        SAVE           //и установка BR
        CLR
M001:   A     BR       //Проверка BR для установки ENO
        )
        JNB   M002     //Вход EN следующего элемента
        L     tmp1
        RND
        T     tmp2
        AN    OV
        SAVE
        CLR
M002:   A     BR
        )
        A     Input2
        JNB   M003     //Вход EN последнего элемента
        L     tmp2
        T     Numb3
        SET
        SAVE
        CLR
M003:   A     BR
        )
        =     Output1
```

Рис 15.1 Пример реализации механизма EN/ENO

Пример: В начале обработки блока BR устанавливается в состояние "1". На тот случай, если после этого при обработке блока будет обнаружена ошибка, например, результат выходит за рамки определенного диапазона (после чего дальнейшая обработка блока должна быть остановлена) Вы должны запрограммировать сброс BR в состояние "0" с помощью JNB и переход на конец блока, например, (в случае ошибки условие должно выдать сигнал "0").

```
SET   ;
SAVE  ;           //BR = "1"
...
L     10000;
L     Result;     //Если Result > 10000,
<=I   ;           //то BR = "0" и выполнить
JNB   ERR;        //переход к ERR
...
```

В примере "Clock entry" в разделе 26.4 "Краткое описание примера фрейма сообщения" также используется BR для сообщения о групповой ошибке.

16 Функции перехода

С помощью функций перехода (jump functions) Вы можете прервать линейное выполнение программы и продолжить ее выполнение в другой точке блока. Такое ветвление программы может быть организовано либо с проверкой выполнения определенного условия, либо без проверки каких-либо условий (соответственно в программе - либо условный, либо безусловный переход).

Распределитель переходов (распределение переходов с проверкой параметра) и циклический переход также доступны пользователю как особые формы функций перехода.

Обзор

JU	*метка*	Безусловный переход
JC	*метка*	Переход, если RLO = "1"
JCN	*метка*	Переход, если RLO = "0"
JCB	*метка*	Переход, если RLO = "1", и сохранение RLO
JNB	*метка*	Переход, если RLO = "0", и сохранение RLO
JBI	*метка*	Переход, если BR = "1"
JBI	*метка*	Переход, если BR = "0"
JZ	*метка*	Переход, если результат равен "0"
JN	*метка*	Переход, если результат не равен "0"
JP	*метка*	Переход, если результат больше "0"
JPZ	*метка*	Переход, если результат больше или равен "0"
JM	*метка*	Переход, если результат меньше "0"
JMZ	*метка*	Переход, если результат меньше или равен "0"
JUO	*метка*	Переход, если результат неверен
JO	*метка*	Переход при переполнении (при проверке бита OV)
JOS	*метка*	Переход при переполнении (при проверке бита OS)
JL	*метка*	Переход в распределителе переходов
LOOP	*метка*	Переход циклический

В этой главе будут рассмотрены функции перехода, используемые в языке программирования STL. В языке программирования SCL выполнение переходов обеспечивается с помощью различных методов ветвления программы, например, с помощью оператора IF (см. главу 28 "Операторы управления").

Примеры, рассматриваемые в данной главе, Вы можете найти на прилагаемой дискете в библиотеке STL_Book в разделе "Program Flow Control" ("Управление выполнением программы") в функциональном блоке FB 116 или в исходном файле Chap_16.

16.1 Программирование функций перехода

Инструкция для каждой функции перехода содержит оператор перехода, определяющий проверяемое условие, и метку перехода, которая в свою очередь показывает, в какой точке программы следует продолжать ее обработку в случае, когда условие для перехода выполняется.

Метка перехода содержит до 4 символов алфавита и числового ряда, а также символ подчеркивания. Метка перехода не может начинаться с цифры. После метки перехода должно следовать двоеточие. Метка перехода указывает на выражение (строку), которая должна обрабатываться после выполнения операции перехода.

На рис. 16.1 показан пример ветвления программы.

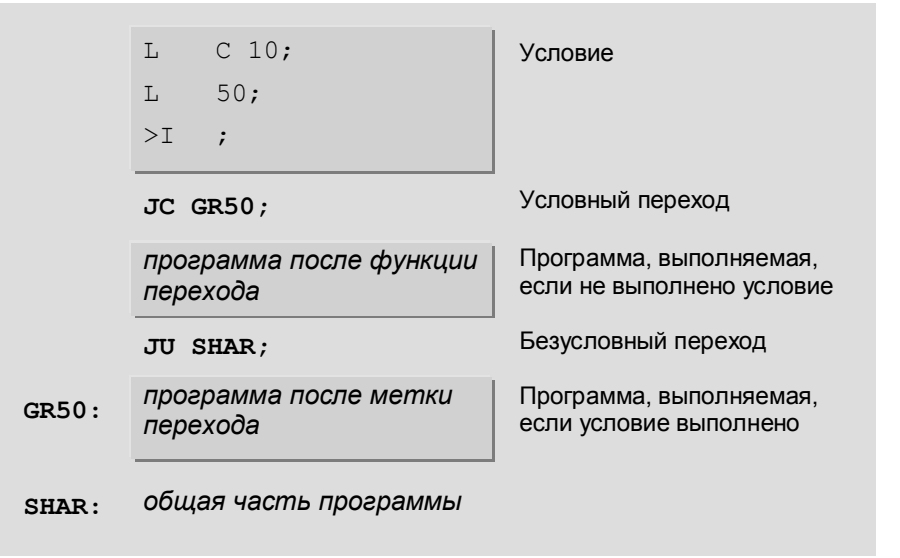


Рис.16.1 Пример ветвления программы.

В примере условием перехода является положительный результат операции сравнения; операция сравнения формирует результат логической операции RLO. Собственно значение RLO и является условием перехода для выражения с JC. При положительном результате операции сравнения (условие операции сравнения выполняется) RLO = "1", при этом выполняется переход к метке GR50. Выполнение программы продолжается с этой метки. При отрицательном результате операции сравнения (условие операции сравнения не выполняется) RLO = "0" и перехода к GR50 нет, программа выполняется со следующего за оператором условного перехода выражения.

Переход может выполняться как вперед (в направлении выполнения программы, т.е., в направлении увеличения номеров строк), так и назад. Переход может выполняться только внутри блока; то есть, назначение точек перехода должно задаваться в том же блоке, где находятся выражения с оператором перехода. Разбиение блока на сегменты не влияет на функцию перехода.

Точки перехода должны иметь уникальный идентификатор, что означает, что Вы можете назначать любую метку перехода в блоке, но только один раз с таким именем.

К одной точке (метке) переход может осуществляться из нескольких точек программы блока. Если Вы используете главное управляющее реле MCR, то метки перехода должны быть в той же MCR-зоне или в той же MCR-области, что и выражение с оператором перехода.

При программировании на языке STL идентификаторы меток сохраняются в соответствующей неисполняемой части блока на носителях данных программатора PG. Только величина перехода хранится в рабочей (work) памяти CPU (в скомпилированном блоке). По этой причине изменения в программе блока, сделанные в интерактивном режиме в CPU, должны также быть выполнены и на дисках программатора PG для сохранения исходных назначений. Если обновления программы не сделаны или если блоки были перенесены из CPU в программатор PG, то соответствующие неисполняемые части блоков переписываются или уничтожаются. Затем редактор сгенерирует свои собственные назначения меток (M001, M002 и т.д.) на экране или в распечатке.

16.2 Безусловный переход

Функция безусловного перехода JU выполняется всегда, т.е. не зависит ни от каких условий. Функция JU прерывает последовательное выполнение программы и продолжает это выполнение с другой точки программы - с метки перехода, указанной в инструкции.

Функция безусловного перехода JU не влияет на биты состояния. Если встречаются операторы проверки (опроса), такие, например, как A I, O I и т.д., располагающиеся непосредственно перед функцией перехода и сразу после метки перехода, то они воспринимаются как отдельные логические операции.

16.3 Функции перехода в зависимости от состояния RLO и BR

Переход в программе может быть сделан зависящим от состояния сигналов битов RLO и BR (см. табл. 16.1). Кроме того, одновременно можно сохранить результаты проверки битов состояния RLO и BR.

Таблица 16.1 Функции перехода с RLO и BR

RLO	BR	Выполняемые переходы	
"1"	-	JC	Переход, если RLO = "1"
"1"	--> "1"	JCB	Переход, если RLO = "1" и сохранить RLO
"0"	-	JCN	Переход, если RLO = "0"
"0"	--> "0"	JNB	Переход, если RLO = "0" и сохранить RLO
-	"1"	JBI	Переход, если BR = "1"
-	"0"	JNBI	Переход, если BR = "0"

Установка битов состояния

Функции перехода, зависящие от RLO, устанавливают биты состояния STA и RLO в "1", а OR и /FC в "0", и при выполнении условия для перехода, и при невыполнении этого условия.

Это приводит к нижеуказанным последствиям при использовании этих функций перехода: RLO всегда устанавливается в состояние "1". Если выражения содержат операции, зависящие от RLO, следующие сразу за такими функциями перехода, то они будут выполняться, если не будет выполнен переход. Если встречаются операторы проверки (опроса), такие, например, как A I, O I и т.д., располагающиеся непосредственно за такими функциями перехода, то эти операторы проверки (опроса) воспринимаются как операции первичного опроса, что означает, что начинается новый логический шаг (новая логическая операция).

Функции перехода, зависящие от BR, устанавливают бит состояния STA в "1", а OR и /FC в "0", и при выполнении условия для перехода, и при невыполнении этого условия. Биты состояния BR и RLO остаются неизменными. Это приводит к нижеуказанным последствиям при использовании таких функций перехода: такие функции перехода завершают логическую операцию; новая логическая операция начинается после таких функций перехода или с метки перехода в программе при выполнении условия для перехода. Бит состояния RLO остается и может быть проверен с помощью операций с памятью сразу после функции перехода.

Переход при RLO = "1"

Функция перехода JC выполняется только в том случае, если RLO = "1" в момент обработки функции. Если RLO = "0", то переход не выполняется и программа продолжает выполняться со следующей инструкции.

Переход при RLO = "0"

Функция перехода JCN выполняется только в том случае, если RLO = "0" в момент обработки функции. Если RLO = "1", то переход не выполняется и программа продолжает выполняться со следующей инструкции.

Переход при RLO = "1" и сохранение RLO

Функция перехода JCB выполняется только в том случае, если RLO = "1" в момент обработки функции. Одновременно функция JCB устанавливает двоичный результат BR в состояние "1". Если RLO = "0", то переход не выполняется и программа продолжает выполняться со следующей инструкции. В этом случае функция JCB устанавливает двоичный результат BR в состояние "0" (т.е., в любом случае RLO переносится в двоичный результат BR).

Переход при RLO = "0" и сохранение RLO

Функция перехода JNB выполняется только в том случае, если RLO = "0" в момент обработки функции. Одновременно функция JNB устанавливает двоичный результат BR в состояние "0". Если RLO = "1", то переход не выполняется и программа продолжает выполняться со следующей инструкции. В этом случае функция JNB устанавливает двоичный результат BR в состояние "1" (т.е., в любом случае RLO переносится в двоичный результат BR).

Переход при BR = "1"

Функция перехода JBI выполняется только в том случае, если двоичный результат BR = "1" в момент обработки функции. Если двоичный результат BR = "0", то переход не выполняется и программа продолжает выполняться со следующей инструкции.

Переход при BR = "0"

Функция перехода JBIN выполняется только в том случае, если двоичный результат BR = "0" в момент обработки функции. Если двоичный результат BR = "1", то переход не выполняется и программа продолжает выполняться со следующей инструкции.

16.4 Функции перехода в зависимости от состояния CC0 и CC1

Переход в программе может быть сделан зависящим от состояния сигналов битов CC0 и CC1 (см. табл. 16.2). Это позволяет, например, проверять, положителен ли результат вычисления, равен ли он нулю или меньше нуля (отрицательный результат). Подробную информацию о том, как устанавливаются биты состояния CC0 и CC1, Вы можете найти в главе 15 "Биты состояния".

Таблица 16.2 Функции перехода с CC0 и CC1

CC0	CC1	Выполняемые переходы	
0	0	JZ	Переход, если результат = 0
		JMZ	Переход, если результат = 0 или < 0
		JPZ	Переход, если результат = 0 или > 0
1	0	JM	Переход, если результат < 0
		JMZ	Переход, если результат = 0 или < 0
		JN	Переход, если результат <> 0
0	1	JP	Переход, если результат > 0
		JPZ	Переход, если результат = 0 или > 0
		JN	Переход, если результат <> 0
1	1	JUO	Переход, если результат неверен

Установка битов состояния

Функции перехода, зависящие от CC0 и CC1, не изменяют никаких битов состояния. Если переход выполнен, значение RLO остается при продолжении программы со строки с меткой перехода и может использоваться далее в программе (без изменения в /FC).

Двоичный опрос является другим способом проверки битов состояния (см. главу 15 "Биты состояния").

Переход при условии, что результат равен нулю

Функция перехода JZ выполняется только в том случае, если CC0 = "0" и CC1 = "0". Это случается, если

- аккумулятор accumulator 1 содержит ноль после арифметической или математической операции;
- аккумулятор accumulator 2 содержит то же значение, что и аккумулятор accumulator 1, при выполнении операции сравнения;
- аккумулятор accumulator 1 содержит ноль после выполнения логической операции для чисел;
- значение сдвинутого последним бита содержит ноль после выполнения операции сдвига.

Во всех остальных случаях условие для перехода для функции JZ не выполняется и выполнение программы продолжается со следующей инструкции.

Переход при условии, что результат не равен нулю

Функция перехода JN выполняется только в том случае, если биты состояния CC0 и CC1 имеют разные значения. Этот случай имеет место, если

- аккумулятор accumulator 1 не содержит ноль после арифметической или математической операции;
- аккумулятор accumulator 2 не содержит то же значение, что и аккумулятор accumulator 1, при выполнении операции сравнения;
- аккумулятор accumulator 1 не содержит ноль после выполнения логической операции для чисел;
- значение сдвинутого последним бита содержит "1" после выполнения операции сдвига.

Во всех остальных случаях условие для перехода для функции JN не выполняется и выполнение программы продолжается со следующей инструкции.

Переход при условии, что результат больше нуля

Функция перехода JP выполняется только в том случае, если CC0 = "0" и CC1 = "1". Этот случай имеет место, если

- содержимое аккумулятора accumulator 1 находится внутри диапазона допустимых положительных значений после арифметической или математической операции (проверить нарушение границ диапазона можно с помощью операторов JO или JOS);
- содержимое аккумулятора accumulator 2 больше значения, находящегося в аккумуляторе accumulator 1, при выполнении операции сравнения;
- аккумулятор accumulator 1 не содержит нуля после выполнения логической операции для чисел;
- значение сдвинутого последним бита содержит "1" после выполнения функции сдвига.

Во всех остальных случаях условие для перехода для функции JP не выполняется и выполнение программы продолжается со следующей инструкции.

Переход при условии, что результат больше нуля или равен нулю

Функция перехода JPZ выполняется только в том случае, если CC0 = "0". Этот случай имеет место, если

- содержимое аккумулятора accumulator 1 находится внутри диапазона допустимых положительных значений или равно нулю после арифметической или математической операции (проверить нарушение границ диапазона можно с помощью операторов JO или JOS);
- содержимое аккумулятора accumulator 2 больше значения, находящегося в аккумуляторе accumulator 1, или равно ему при выполнении операции сравнения;
- после выполнения каждой логической операции для чисел;
- после выполнения каждой функции сдвига.

Во всех остальных случаях условие для перехода для функции JPZ не выполняется и выполнение программы продолжается со следующей инструкции.

Переход при условии, что результат меньше нуля

Функция перехода JM выполняется только в том случае, если CC0 = "1" и CC1 = "0". Этот случай имеет место, если

- содержимое аккумулятора accumulator 1 находится внутри диапазона допустимых отрицательных значений после арифметической или математической операции (проверить нарушение границ диапазона можно с помощью операторов JO или JOS);
- содержимое аккумулятора accumulator 2 меньше значения, находящегося в аккумуляторе accumulator 1, при выполнении операции сравнения.

Во всех остальных случаях условие для перехода для функции JM не выполняется и выполнение программы продолжается со следующей инструкции.

Переход при условии, что результат меньше нуля или равен нулю

Функция перехода JMZ выполняется только в том случае, если CC1 = "0". Этот случай имеет место, если

- содержимое аккумулятора accumulator 1 находится внутри диапазона допустимых отрицательных значений или равно нулю после арифметической или математической операции (проверить нарушение границ диапазона можно с помощью операторов JO или JOS);
- содержимое аккумулятора accumulator 2 меньше значения, находящегося в аккумуляторе accumulator 1, или равно ему при выполнении операции сравнения;

Во всех остальных случаях условие для перехода для функции JMZ не выполняется и выполнение программы продолжается со следующей инструкции.

Переход при условии, что результат некорректен

Функция перехода JUO выполняется только в том случае, если CC0 = "1" и CC1 = "1". Этот случай имеет место, если

- если совершается попытка деления на ноль при арифметической операции;
- если некорректное действительное (REAL) число было определено как исходное значение или было получено при выполнении операции.

Во всех остальных случаях условие для перехода для функции JUO не выполняется и выполнение программы продолжается со следующей инструкции.

16.5 Функции перехода в зависимости от состояния OV и OS

Переход в программе может быть сделан зависящим от состояния сигналов битов OV и OS. Это позволяет проверять, находится ли все еще результат вычисления в диапазоне допустимых значений. Подробную информацию о том, как устанавливаются биты состояния OV и OS, Вы можете найти в главе 15 "Биты состояния".

Переход при условии, что результат вышел за пределы диапазона (переполнение) (проверка бита OV)

Функция перехода JO выполняется только в том случае, если OV = "1". Это случается, если результат вычисления выходит за пределы допустимых значений в предыдущей операции. Бит состояния OV может быть установлен такими функциями, как:

- арифметические функции;
- математические функции;
- функции инвертирования числа;
- функции сравнения для чисел формата REAL;
- функции преобразования чисел форматов INT/DINT в формат BCD и чисел формата REAL в формат DINT.

Если бит состояния OV = "0", то условие для перехода для функции JO не выполняется и выполнение программы продолжается со следующей инструкции.

Если выполняется цепочка последовательных вычислений, то состояние бита OV должно проверяться после выполнения каждого вычисления, так как бит состояния OV вновь будет сброшен после выполнения операции вычисления, результат которой будет находится в диапазоне допустимых значений.

Бит состояния OS может быть опрошен после выполнения цепочки вычислений для проверки, не вышел ли за пределы допустимых значений результат одного из вычислений в рассматриваемой последовательности операций.

Переход при условии, что результат вышел за пределы диапазона (переполнение) (проверка бита OS)

Функция перехода JOS выполняется только в том случае, если OS = "1". Это происходит во всех случаях, когда результат вычисления выходит за пределы допустимых значений, что вызывает установку бита состояния OV (см. выше). В отличие от OV, бит OS сохраняет свое состояние, даже если после его установки результат одной из последующих операций будет в диапазоне допустимых значений. Бит состояния OS может быть сброшен при следующих обстоятельствах:

- при вызове блока и при завершении блока;
- при выполнении перехода JOS.

Если бит состояния OS = "0", то условие для перехода для функции JOS не выполняется и выполнение программы продолжается со следующей инструкции.

16.6 Распределитель переходов (Jump Distributor)

Распределитель переходов (Jump Distributor) JL позволяет определять (вычислять) переходы в разделе программы в блоке к разным точкам, отмеченным метками.

Функция JL работает вместе с набором функций перехода JU. Последовательность выражений с операторами перехода JU следует сразу после функции JL и может содержать до 255 строк (входов). В инструкции после оператора JL следует метка перехода, указывающая на конец списка функций перехода JU (на первую инструкцию, следующую за набором функций перехода JU).

Вы можете запрограммировать распределитель переходов (Jump Distributor) JL в соответствии со следующей общей схемой:

```
    L    Number_of_positions;   //номер позиции
    JL   End;
    JU   M0;
    JU   M1;
    JU   M2;
      ...
    JU   Mx;
End:  ...
```

В этом примере переменная *Number_of_positions* содержит число, загружаемое в аккумулятор accumulator 1. Вслед за операцией загрузки следует распределитель переходов JL с меткой, указывающей на конец списка функций перехода JU.

Номер перехода, который должен быть выполнен, содержится в правом байте аккумулятора accumulator 1. Если этот аккумулятор содержит 0, то выполняется первая функция перехода. Если этот аккумулятор содержит 1, то выполняется вторая функция перехода и так далее. Если число в аккумуляторе превышает размер списка операторов перехода, то

происходит переход на конец списка (на первую инструкцию, следующую за набором функций перехода JU).

JL не зависит ни от каких условий и не изменяет битов состояния.

При этом только выражения с оператором JU, располагающиеся без пробелов, допускаются в списке операторов перехода функции JL. Назначая произвольные метки для этих операторов перехода, Вы должны придерживаться общих правил для меток.

16.7 Циклический переход (Loop Jump)

Функция циклического перехода (Loop Jump) LOOP позволяет просто реализовать в программе программные циклы.

Функция LOOP интерпретирует число в правом слове (word) аккумулятора accumulator 1 как беззнаковое 16-разрядное число из диапазона от 0 до 65535.

При обработке функция LOOP сначала декрементирует (уменьшает) содержимое аккумулятора accumulator 1 на 1. Если значение при этом не равно нулю, то выполняется переход к указанной метке.

Если значение после декрементирования равно нулю, то выполняется следующее за телом цикла выражение.

Значение в аккумуляторе accumulator 1, таким образом, соответствует числу циклов, которые должны быть выполнены. Вы должны сохранить это число в счетчике цикла. Вы можете использовать любое число (адрес) как счетчик цикла.

Вы можете запрограммировать циклический переход LOOP в соответствии со следующей общей схемой:

```
          L     Number;
Next:     T     Counter;
          ...
          ...
          ...
          L     Counter;
          LOOP  Next;
          ...
```

В этом примере переменная *Number* содержит общее число циклов, которое необходимо выполнить. Переменная *Counter* содержит число циклов, которое осталось выполнить.

При первом проходе цикла значению *Counter* задается число циклов, которое необходимо выполнить. В конце программного цикла содержимое переменной *Counter* загружается в аккумулятор и декрементируется с помощью оператора LOOP. Если после этого аккумулятор не содержит нуля, то выполняется переход к метке цикла - здесь к метке Next. После этого происходит обновление переменной *Counter.*

Функция циклического перехода LOOP не изменяет битов состояния.

17 Главное управляющее реле MCR

Управляя методом переключения, главное управляющее реле (Master Control Relay - MCR) активирует или деактивирует отдельные фрагменты схемы управления, которая может состоять из одного или нескольких уровней. При деактивации уровня:

- выключаются все переманентные контакторы и
- остаются неизменными состояния всех реманентных контакторов.

Вы можете вновь изменить состояния этих контакторов только тогда, когда главное управляющее реле MCR будет активировано.

Эти свойства главного управляющего реле MCR определяют его использование особенно в LAD-программах.

В этой главе рассматриваются операторы, необходимые для выполнения функций главного управляющего реле MCR в языке программирования STL. Вы можете использовать эти операторы для эмуляции свойств главного управляющего реле MCR в STL.

Примеры использования главного управляющего реле MCR, рассматриваемые в данной главе, Вы можете найти на прилагаемой дискете в библиотеке STL_Book в разделе "Program Flow Control" ("Управление выполнением программы") в функциональном блоке FB 117 или в исходном файле Chap_17.

Необходимо отметить, что выключение с помощью программного варианта главного управляющего реле (Master Control Relay - MCR) не заменяет антиаварийных или предохранительных устройств. Вы должны трактовать включение с помощью главного управляющего реле MCR, как включение посредством операций с памятью (memory functions)!

Язык программирования STL предоставляет следующие операторы для выполнения функций главного управляющего реле MCR:

- MCRA оператор активации области MCR
- MCR(оператор открытия зоны MCR
-)MCR оператор закрытия зоны MCR
- MCRD оператор деактивации области MCR

Операторы MCRA и MCRD определяют область в Вашей программе, на которую распространяется действие MCR. Внутри этой области Вы можете использовать операторы MCR(и)MCR для задания одной или нескольких зон действия MCR, в которых зависимость от MCR может включаться или выключаться.

Вы можете также использовать вложенные MCR зоны. Результат логической операции RLO, выполняемой непосредственно перед зоной MCR включает или выключает MCR-зависимость внутри этой зоны.

17.1 MCR-зависимость (MCR Dependency)

Главное управляющее реле (MCR) влияет на все операции, которые возвращают (записывают) значения в память. Такие операции, зависящие от MCR, при включении MCR-зависимости, независимо от результатов любых предыдущих двоичных логических операций или логических операций с числами, реагируют следующим образом:

- оператор присвоения (=):
  содержимое адреса сбрасывается в состояние "0"
- операторы установки Set (S) и сброса Reset (R):
  содержимое адреса остается неизменным
- оператор пересылки Transfer (T):
  пересылается ноль "0"

Некоторые функции STL используют операторы пересылки (незаметно для пользователя) для того, например, чтобы записать значение в адресный регистр. Так как оператор пересылки записывает значение нуля "0", если MCR-зависимость включена, то выполнение соответствующей функции (использующей оператор пересылки) не может быть гарантировано.

Поэтому, чтобы во время работы избежать переход CPU в состояние STOP или в неопределенное состояние, Вы должны исключить возможность влияния MCR на соответствующие части программы, содержащие:

- вызовы блоков с параметрами
- операции доступа к параметрам блоков, которые относятся к типам параметров (например, BLOCK_DB)
- операции доступа к параметрам блоков, которые являются компонентами или элементами сложных типов данных или типов, определенных пользователем UDT.

Если MCR-зависимость выключена, то операции, зависящие от MCR, выполняются в "нормальном" режиме, в соответствии с описанием в соответствующих разделах данной книги.

Вы включите MCR-зависимость в зоне, если RLO равен "0" непосредственно перед открытием зоны (аналогично выключению главного управляющего реле MCR). Если открытие MCR-зоны происходит при RLO, равном "1" (главное управляющее реле MCR включается), то обработка программы в такой MCR-зоне выполняется без MCR-зависимости.

Пример:

```
MCRA    ;          //Активация MCR
A       Input0;
MCR(    ;          //Открытие MCR-зоны
A       Input1;
A       Input2;
=       Output0;
)MCR    ;          //Закрытие MCR-зоны
```

```
MCRD    ;           //Деактивация MCR
```

В этом примере переменная *Input0* = "0" сбрасывает содержимое адреса *Output0* в "0". Если *Input0* имеет состояние "1", то содержание адреса *Output0* будет зависеть только от *Input1* и *Input2*.

17.2 MCR-область (MCR Area)

Для того, чтобы использовать функции главного управляющего реле (MCR), Вы должны определить область его действия - MCR-область (MCR Area) с помощью операторов MCRA и MCRD. MCR-зависимость активна внутри MCR-области (но пока еще не включена)

```
MCRA    ;           //Активация MCR
...
...                 //MCR-область
...
MCRD    ;           //Деактивация MCR
```

Оператор MCRA определяет начало MCR-области, а оператор MCRD закрывает эту область. Если Вы вызываете блок внутри MCR-области, то MCR-зависимость является деактивированной в вызванном блоке (см. рис. 17.1). MCR-область начинается снова только с появлением нового оператора MCRA. Когда по окончании обработки вызванного блока управление возвращается в вызывающий блок, MCR-зависимость вновь будет находиться в том состоянии, в котором она находилась до вызова блока, независимо от состояния MCR-зависимости при окончании обработки этого блока.

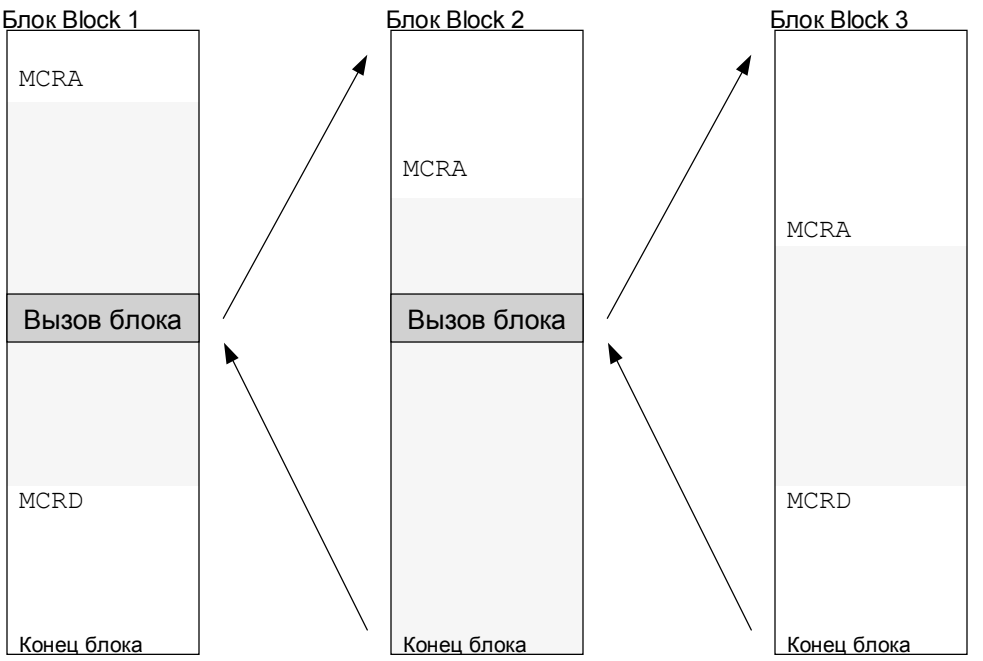


Рис. 17.1 MCR-область в случае вызовов блоков

17.3 MCR-зона (MCR Zone)

Вы можете определить MCR-зону (MCR Zone) с помощью операторов MCR(и)MCR. Внутри MCR-зоны Вы можете включать MCR-зависимость при RLO = "0" и выключать при RLO = "1".

```
...                 //Включение MCR
...                 //при "0"
A     Input3;
MCR(  ;             //Включение MCR-зависимости
...
...                 //MCR-зона
...
)MCR  ;             //Выключение MCR-зависимости
...
```

Операторы MCR(и)MCR заканчивают двоичную логическую операцию (комбинацию).

Вы можете открыть другую MCR-зону внутри MCR-зоны. Глубина вложения для MCR-зон может достигать 8, что означает, что Вы можете открыть до 8 зон перед тем, как должны будете закрыть зону.

Если MCR-зона открыта, Вы можете управлять MCR-зависимостью включенной MCR-зоны с помощью RLO. Однако, если MCR-зависимость включена в MCR-зоне верхнего уровня, Вы не сможете выключить MCR-зависимость в MCR-зоне нижнего уровня. Главное управляющее реле (MCR) первой MCR-зоны управляет MCR-зависимостью во всех включенных зонах (см. рис. 17.2).

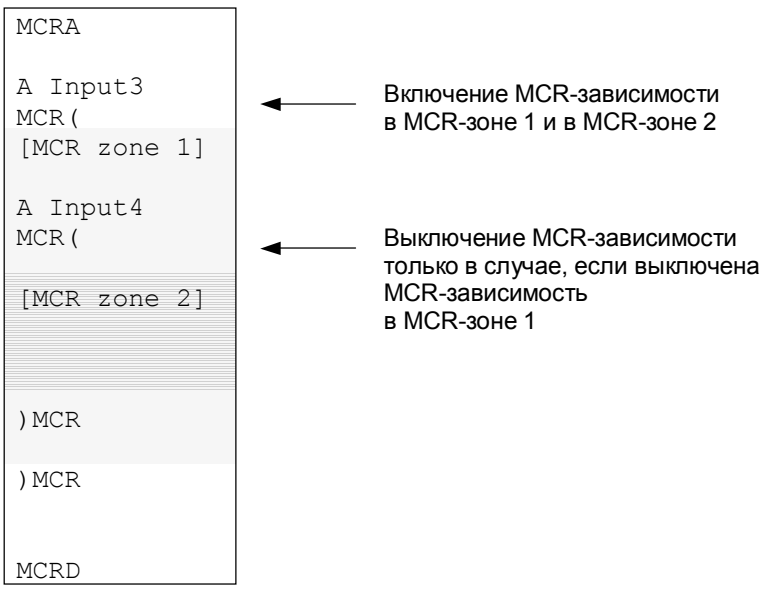


Рис. 17.2 MCR-зависимость в случае вложенных MCR-зон

Вызов блока внутри MCR-зоны не изменяет глубины вложения MCR-зоны. Программа в вызванном блоке находится все еще в MCR-зоне, которая была в открытом состоянии, когда был вызван блок (и управляется из нее). Тем не менее, Вы должны вновь активировать MCR-зависимость в вызванном блоке с помощью открытия MCR-области оператором MCRA (см. рис. 17.3).

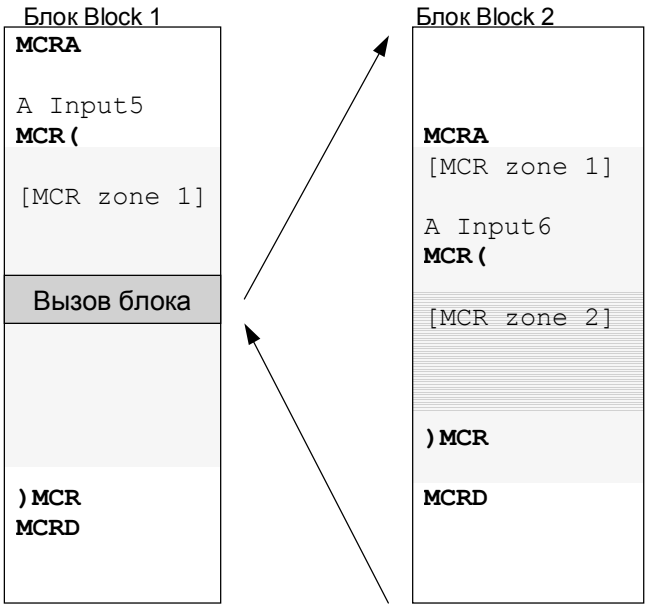


Рис. 17.3 MCR-зоны в случае вызова блока

На рис. 17.3 адреса *Input5* и *Input6* управляют MCR-зависимостью. Посредством адреса *Input5* Вы можете включить MCR-зависимость в обоих зонах (сигналом "0"), независимо от состояния адреса *Input6*. Если MCR-зависимость зоны 1 выключается посредством *Input5* = "1", то Вы можете управлять MCR-зависимостью в зоне 2 посредством адреса *Input6* (см. табл. 17.1).

Таблица 17.1 Пример управления MCR-зависимостью в случае вложенных MCR-зон

Вход *Input5*	Вход *Input6*	Зона Zone 1	Зона Zone 2
"1"	"1"	Нет MCR-зависимости	
"1"	"0"	Нет MCR-зависимости	MCR-зависимость включена
"0"	"1" или "0"	MCR-зависимость включена	

17.4 Установка и сброс битов периферии (I/O битов)

При включенной MCR-зависимости Вы можете устанавливать (Set) и сбрасывать (Reset) биты в области I/O с помощью системных функций.

Для этого требуется, чтобы биты, которыми необходимо управлять, были в области отображения выходов процесса.

Системная функция **SFC 79 SET** предназначена для установки I/O битов, а системная функция **SFC 80 RSET** предназначена для их сброса (см. табл. 17.2). Вы можете вызывать эти функции в MCR-зоне. Системные функции работают только в том случае, когда MCR-зависимость включена; если MCR-зависимость выключена, то вызовы указанных системных функций останутся без всяких последствий.

Таблица 17.2 Параметры SFC для управления I/O-битами

SFC	Параметр	Объявление	Тип	Назначение, описание
79	N	INPUT	INT	Число устанавливаемых битов
	RET_VAL	OUTPUT	INT	Информация об ошибках
	SA	OUTPUT	POINTER	Указатель на первый бит, который должен быть установлен
80	N	INPUT	INT	Число сбрасываемых битов
	RET_VAL	OUTPUT	INT	Информация об ошибках
	SA	OUTPUT	POINTER	Указатель на первый бит, который должен быть сброшен

При выполнении установки и сброса I/O-битов одновременно обновляются выходы из области отображения процесса по выходам.

Периферийные входы/выходы (I/O) управляются побайтно (байт за байтом). Биты, не выбранные с помощью SFC-функций, (в первом и в последнем байте) сохраняют состояние сигнала, так как они доступны в области отображения процесса.

Пример:

```
CALL SFC 79 (
  N         := 8,
  RET_VAL   := MW 10,
  SA        := P#12.0);
CALL SFC 80 (
  N         := 16,
  RET_VAL   := MW 12,
  SA        := P#13.5);
```

В данном примере вызов функции SFC 79 SET устанавливает I/O-биты, соответствующие выходам Q 12.0 ... Q 12.7; вызов функции SFC 80 RSET сбрасывает I/O-биты, соответствующие выходам Q 13.5 ... Q 15.5.

Параметр N определяет число битов, которые должны быть обработаны, а параметр SF определяет первый бит (тип "указатель" - POINTER) из числа битов, которые должны быть обработаны. Указанные функции SFC используют также параметр RET_VAL, в котором они возвращают информацию об ошибках, которые возникли при выполнении функций.

18 Функции блоков (Block Functions)

Из этой главы Вы узнаете, как вызывать и завершать обработку кодовых блоков, а также, как работать с адресами из блоков данных при использовании языка программирования STL. В следующей главе будут рассмотрены параметры блоков и использование этих параметров. Данная глава является продолжением раздела 3.4 "Программирование кодовых блоков на STL" и раздела 3.6 "Программирование блоков данных".

Как вызываются блоки при использовании языка программирования SCL, описано в главе 29 "SCL блоки".

Примеры, рассматриваемые в данной главе, Вы можете найти на прилагаемой дискете в библиотеке STL_Book в разделе "Program Flow Control" ("Управление выполнением программы") в функциональном блоке FB 118 или в исходном файле Chap_18.

18.1 Функции для кодовых блоков

К функциям для кодовых блоков относятся инструкции для вызова и завершения обработки блоков (см. табл. 18.1).

Таблица 18.1 Функции для кодовых блоков

Вызов функционального блока		
С блоком данных и с параметрами блока	Как локальный экземпляр и с параметрами блока	Без параметров блока безусловный вызов и вызов по условию

CALL FB 10, DB 10 (
` In1 := Number1;`
` In2 := Number2;`
` Out := Number3);`
	CALL *name* (
	`  In1 := Number1;`
	`  In2 := Number2;`
	`  Out := Number3);`
		UC FB 11;
		CC FB 11;

Вызов функции		
Со значением функции и с параметрами блока	Без значения функции, но с параметрами блока	Без параметров блока безусловный вызов и вызов по условию

CALL FB 10 (
` In1 := Number1;`
` In2 := Number2;`
` **Ret_Val** := Number3);`
	CALL FB 10 (
	`  In1 := Number1;`
	`  In2 := Number2;`
	`  Out := Number3);`
		UC FB 11;
		CC FB 11;

Операторы завершения блока		
Условное завершение обработки блока	Безусловное завершение обработки блока	Конец блока
BEC	**BEU**	**BE**

Кодовые блоки вызываются для обработки с помощью оператора CALL. Вы можете передавать данные в вызываемый блок и можете также получать данные от вызываемого блока. Такая передача данных выполняется с помощью параметров блока. С оператором CALL в вызываемый блок пересылаются параметры блока, а также открывается экземплярный блок данных в случае вызова функционального блока. Если кодовые блоки не имеют параметров, то их можно вызвать посредством операторов UC или CC. Обработка блока прекращается посредством оператора конца блока.

18.1.1 Вызов блока: общая информация

Если кодовый блок должен быть обработан, он должен быть "вызван".

На рис. 18.1 показан пример вызова функции FC 10 в организационном блоке OB 1.

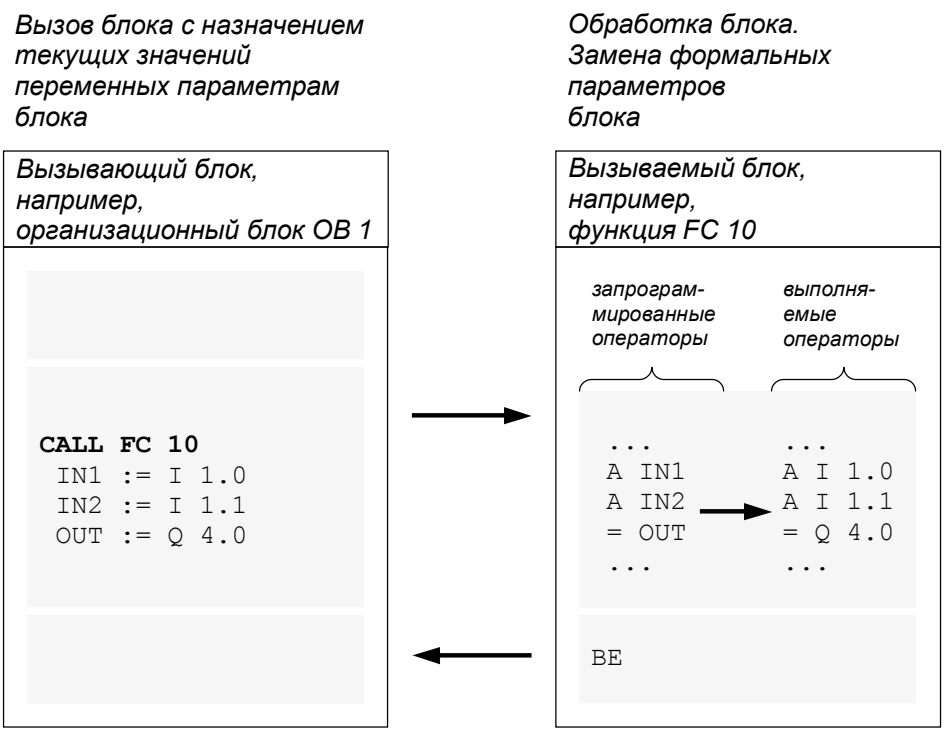


Рис. 18.1 Пример вызова блока

Вызов блока состоит из оператора вызова (в примере: CALL FC 10) и списка параметров. Если вызываемый блок не имеет параметров, инструкция вызова не будет иметь списка параметров. После выполнения оператора вызова CPU продолжает выполнение программы в вызываемом блоке (в примере: FC 10). Программа блока обрабатывается, пока не встретится оператор окончания блока. По окончании вызванного блока CPU возвращается к выполнению программы в вызывающем блоке (в примере: OB 1); выполнение этой программы продолжается со следующего оператора после оператора вызова блока. Если завершается выполнение программы организационного блока, CPU передает

управление операционной системе.

Информация, которая требуется CPU для возврата в вызывающий блок, сохраняется в стеке блоков (B-стек [B stack]). При каждом вызове блока в B-стеке генерируется новый элемент стека, который содержит адрес возврата, содержимое регистра данных и адрес стека локальных данных вызывающего блока. Если CPU переходить в состояние STOP в результате ошибки, Вы можете использовать программатор для того, чтобы увидеть в B-стеке информацию о том, какие блоки обрабатывались до момента возникновения ошибки.

Параметры блока являются интерфейсом данных для вызываемого блока. Рекомендуется избегать передачи данных посредством внутренних регистров (например, посредством аккумуляторов, адресных регистров, RLO), так как содержимое этих регистров может быть изменено при смене текущего блока (в результате "скрытых" операторов, инициированных редактором).

18.1.2 Оператор вызова блока CALL

С помощью оператора вызова блока CALL Вы можете вызывать функциональные блоки FB, функции FC, системные функциональные блоки SFB, системные функции SFC. Оператор вызова блока CALL является безусловным вызовом, что означает, что заданный блок всегда вызывается и обрабатывается, несмотря ни на какие условия. (Вы не можете вызывать организационные блоки; организационные блоки вызываются операционной системой, в зависимости от события).

Вызов функциональных блоков

Вы можете вызывать функциональные блоки FB, задав идентификатор блока после оператора CALL, и отделенный запятой идентификатор экземплярного блока данных, связанного с этим вызовом. Вы можете адресовать оба блока или абсолютным способом, или символьным. Назначение абсолютного адреса символьному адресу выполняется в таблице символов (Symbol Table) для экземплярного блока данных, имеющего связанный функциональный блок как тип данных.

В инструкции вызова может быть список параметров блока. При вводе исходного текста программы список параметров блока помещается между круглыми скобками; в списке параметры блока должны быть разделены запятыми.

При вызове функционального блока нет необходимости инициализировать все параметры вызываемого блока. Неинициализированные параметры блока сохраняют свои текущие значения. Если Вы не задаете никаких параметров, то скобки также не вводятся при вводе исходного текста программы.

Если Вы создаете функциональные блоки с атрибутом "мультиэкземплярные" ("multi-instance-capable"), Вы можете также вызывать эти блоки, как локальные экземпляры внутри других "мультиэкземплярных" функциональных блоков. Имеется в виду, что вызванный функциональный блок использует экземплярный блок данных вызывающего функционального блока для сохранения своих собственных локальных данных.

В таком случае Вы должны объявить локальный экземпляр в статических локальных данных вызывающего функционального блока, и после этого Вы можете вызывать функциональный блок в программе (без определения экземплярного блока данных). Локальный экземпляр трактуется как сложный тип данных внутри функционального блока "верхнего уровня". Более подробно материал изложен в разделе 18.1.6 "Статические локальные данные".

Вызов функций

Вы можете вызывать функции FC, задав идентификатор функции после оператора CALL, или абсолютным способом, или символьным. Далее в инструкции вызова следует список параметров. При вводе исходного текста программы список параметров блока помещается между круглыми скобками; в списке параметры блока должны быть разделены запятыми.

При вызове функции Вы должны инициализировать все параметры; тем не менее, параметры могут следовать в любом порядке. Вызываемые функции с функциональным значением имеют точно такую же форму, как и функции без функционального значения. Единственный выходной параметр, соответствующий функциональному значению, имеет имя RET_VAL.

Вызов системных блоков

Операционная система CPU содержит системные функции SFC и системные функциональные блоки SFB, доступные пользователю. Число и тип системных блоков определяется типом CPU. Все системные блоки вызываются оператором CALL.

Вы можете вызывать системные функциональные блоки таким же образом, как и те блоки, которые Вы можете написать сами; Вы должны расположить связанный экземплярный блок данных в пользовательской памяти (user memory) с типом данных SFB. Вы можете также вызывать системные функции таким же образом, как и функции, которые Вы можете написать сами.

Системные блоки доступны только в операционной системе CPU. При вызове системных блоков во время программирования в автономном (offline) режиме, редактору требуется описание интерфейса вызова для того, чтобы было возможно инициализировать параметры. Описание интерфейса расположено в стандартной библиотеке *Standard library* в системных функциональных блоках *System Function Blocks*. Отсюда редактор копирует описание интерфейса в папку (раздел) автономного режима "Blocks", когда Вы вызываете системный блок. После этого скопированное описание интерфейса вызова появляется как "нормальный" объект блока.

18.1.3 Операторы вызова UC и CC

Вы можете вызывать функциональные блоки и функции с помощью операторов вызова UC и CC. При этом требуется, чтобы вызываемые функции не имели параметров блоков и вызываемые функциональные блоки не имели экземплярного блока данных и, следовательно, не имели параметров блоков и статических локальных данных. Тем не менее, редактор не проверяет, выполняются ли эти условия.

Для вызова Вы можете использовать операторы UC и CC, если блок слишком большой и недостаточно удобен для понимания. При этом блок

просто разбивается на отдельные фрагменты, после чего эти фрагменты вызываются последовательно. Операторы вызова UC и CC не различают функции FC и функциональные блоки FB. Оба типа блоков обрабатываются одинаковым образом.

Оператор вызова UC является безусловным оператором, что означает, что UC вызывает блок вне зависимости от любых условий.

Оператор вызова CC является условным оператором, что означает, что CC вызывает блок только при условии, что результат логической операции (RLO) равен "1". Если RLO равен "0", то блок не вызывается, но RLO устанавливается в "1". После этого выполняется следующий за операцией вызова оператор.

Влияние на индикаторные биты (биты условного кода "condition code"): бит состояния OS сбрасывается при смене текущего блока; на биты состояния CC0, CC1 и OV смена блока не влияет, тогда как бит состояния /FC при смене текущего блока сбрасывается, что означает, что новая логическая операция начинается с операции первичного опроса в новом блоке или следует за вызовом блока.

Влияние смены текущего блока на "стек скобок" ("binary nesting stack"): Вы можете вызывать кодовый блок внутри "двоичного выражения вложения". Текущая глубина стека скобок не изменяется при смене блока. Возможная глубина стека в блоке, который может быть вызван внутри двоичного вложения (binary nest), следовательно, равен разности между максимально возможной глубиной и текущей глубиной вложения при вызове блока.

Влияние смены текущего блока на главное управляющее реле MCR: MCR-зависимость деактивируется при вызове блока. MCR выключается в вызванном блоке, независимо от того, было ли включено или не было включено MCR перед вызовом блока. При окончании обработки вызванного блока MCR-зависимость имеет то состояние, в котором она находилась до вызова блока.

Влияние смены текущего блока на аккумуляторы и адресные регистры: содержимое аккумуляторов и адресных регистров не изменяется при вызове блока с помощью операторов вызова UC и CC.

Влияние смены текущего блока на блоки данных: при вызове блока регистр блока данных сохраняется в B-стеке; оператор окончания блока сохраняет его содержимое при завершении обработки вызванного блока. Блок глобальных данных и экземплярный блок, бывшие текущими перед вызовом блока также открываются после операции вызова блока. Если перед вызовом блока нет открытого блока данных (например, нет экземплярного блока данных в OB 1), то не будет также открытих блоков данных после операции вызова блока, независимо от того, какие блоки данных открыты в вызванном блоке.

Дополнительные возможности:

- косвенная адресация вызовов блоков FB и FC с помощью операторов вызова UC и CC;
- организация вызова с параметрами блока с помощью оператора вызова UC;

- организация вызова с параметрами блока с помощью оператора вызова CC также и в функциональных блоках.

18.1.4 Функции окончания блока (Block End Functions)

Оператор BEC завершает обработку программы в блоке, при этом зависит от состояния RLO, а операторы BEU и BE заканчивают блок независимо от условий.

Оператор завершения блока по условию BEC

Выполнение оператора BEC, завершающего обработку программы в блоке, зависит от состояния RLO. Если результат логической операции RLO = "1" при обработке оператора BEC, то функция окончания блока выполняется и обрабатываемый блок закрывается. При этом выполняется переход в ранее обрабатывавшийся блок, из которого был сделан вызов только что завершенного блока.

Если RLO = "0" при обработке оператора BEC, то функция окончания блока не выполняется. При этом CPU устанавливает RLO в состояние "1" и выполняется следующая за оператором BEC инструкция. Следующий запрограммированный оператор проверки (опроса) в любом случае является первичным опросом.

Оператор безусловного завершения блока BEU

Выполнение оператора BEU не зависит ни от каких условий; оператор BEU выполняется и обрабатываемый блок закрывается. При этом выполняется переход в ранее обрабатывавшийся блок, из которого был сделан вызов только что завершенного блока.

В отличие от оператора BE оператор BEU может быть использован внутри одного блока несколько раз. При этом отдельные части программы, следующие за оператором BEU, могут быть обработаны только в случае выполнения операции перехода к ним.

Оператор безусловного завершения блока BE

Выполнение оператора BE не зависит ни от каких условий; оператор BE выполняется и обрабатываемый блок закрывается. При этом выполняется переход в ранее обрабатывавшийся блок, из которого был сделан вызов только что завершенного блока.

Оператор BE всегда является последним оператором в блоке.

Использование оператора BE является предметом выбора. При "инкрементном" программировании Вы завершаете программирование блока, закрыв блок; при программировании, ориентированном на ввод исходного текста программы, конец блока отмечается ключевым словом, например, END_FUNCTION_BLOCK вместо оператора BE.

18.1.5 Временные локальные данные

Временные локальные данные используются для промежуточного

хранения результатов, получающихся при обработке программы блока. Временные локальные данные доступны только во время обработки блока; после завершения блока эти данные теряются.

Временные локальные данные соответствуют адресам, которые расположены в стеке локальных данных (в L-стеке) в системной памяти CPU. Операционная система CPU обеспечивает размещение временных локальных данных для каждого кодового блока при вызове этого блока. Первоначально при вызове блока значения в L-стеке носят "псевдослучайный" характер. Для того, чтобы эти значения отвечали реалиям, до того, как они будут считаны, необходимо их записать. После завершения обработки блока, L-стек назначается следующему вызываемому блоку.

Объем памяти, требуемый для размещения локальных данных для блока, указывается в заголовке блока. Таким образом операционная система узнает, сколько байт в L-стеке необходимо отвести для вызываемого блока. Вы также можете узнать из заголовка блока, сколько байт локальных данных требуется для блока (если блок открыт, в редакторе с помощью опций меню: *File -> Properties (Файл -> Свойства),* или в оболочке SIMATIC Manager при выбранном блоке с помощью опций меню: *Edit -> Object Properties (Правка -> Свойства объекта),* информация и в том и в другом случае находится на вкладке "General - Part 2" ("Общие - Часть 2")).

Объявление временных локальных данных

Объявление временных локальных данных производится в разделе объявлений кодового блока:

- при инкрементном программировании временные локальные данные помечаются в столбце "Declaration" ("Объявление") словом "temp" (временный);
- при программировании, ориентированном на создание программы путем ввода исходного текста, временные локальные данные объявляются между ключевыми словами: VAR_TEMP и END_VAR.

На рис. 18.2 представлен пример объявления временных локальных данных. В примере переменная *temp1*, расположенная во временных локальных данных относится к типу INT, а переменная *temp2* относится к типу REAL.

Временные локальные данные сохраняются в L-стеке в порядке их объявления в соответствии с типами данных.

Более подробно материал о хранении данных в L-стеке изложен в разделе 26.2 "Хранение переменных".

Символьная адресация временных локальных данных

Адресацию временных локальных данных можно производить с использованием символьных имен. Для этого данным должны быть назначены имена в соответствии с правилами, принятыми для символьных имен (символов) локальных для блока данных.

Все операции, которые действительны для меркеров, разрешены для временных локальных данных. Тем не менее, надо отметить, что биты из временных локальных данных непригодны для использования в качестве

меркеров фронта, так как они не сохраняют состояние своего сигнала после окончания обработки блока.

Инкрементное программирование

Address (адрес)	Declaration (объявление)	Name (имя)	Type (тип)	Initial value (начальное значение)
0.0	in	In	INT	0
	out			
	inout			
2.0	stat	Total	INT	0
0.0 2.0	temp temp	temp1 temp2	INT REAL	

Программирование, ориентирование на создание исходных текстов программы

```
VAR_INPUT
  In   : INT := 0;
END_VAR
VAR_OUTPUT ... END_VAR
VAR_IN_OUT ... END_VAR
VAR
  Total : INT := 0;
END_VAR
VAR_TEMP
  Temp1 : INT;
  Temp2 : REAL;
END_VAR
```

Рис. 18.2 Пример объявления локальных данных в функциональном блоке.

Вы можете получить доступ к временным локальным данным блока только непосредственно в самом блоке. (Исключение: к временным локальным данным вызывающего блока можно получить доступ через параметры блока.)

Размер L-стека

Предельный размер для L-стека определяется типом CPU. Количество байтов временных локальных данных доступное в приоритетном классе, к которому относится программа организационного блока, также фиксировано. Для S7-300 размер L-стека фиксирован, например, 256 байт на приоритетный класс для CPU 314. При работе с S7-400 размер L-стека может настраиваться пользователем. Необходимое число байт отводится для временных локальных данных при параметризации CPU. Эта область памяти должна быть общедоступной для блоков, вызов которых производятся из соответствующего организационного блока и для блоков, которые вызываются, в свою очередь, из этих блоков.

Необходимо заметить в этой связи, что редактор также использует временные локальные данные, например, при передаче параметров блока. Вы не видите эти временные локальные данные в интерфейсе программирования.

Стартовая информация (Start information)

Операционная система CPU передает стартовую информацию (start information) во временных локальных данных при вызове организационного блока. Эта стартовая информация занимает размер памяти 20 байт в каждом организационном блоке и имеет почти идентичную структуру в каждом блоке. В главах 20 "Главная программа", 21 "Управление прерываниями", 22 "Параметры перезапуска" и 23

"Обработка ошибок" описывается назначение стартовой информации для отдельных организационных блоков.

Упомянутые 20 байт стартовой информации должны быть всегда доступны в каждом используемом приоритетном классе. Если Вы программируете обработку синхронных ошибок (ошибки программирования и доступа), необходимо обеспечить дополнительные 20 байт, по крайней мере, для стартовой информации этих организационных блоков обработки ошибок, так как эти OB ошибок должны обрабатываться в том же приоритетном классе.

Вы должны объявить эту стартовую информацию, при программировании организационного блока. Это обязательное условие. В стандартной библиотеке *Standard Library* имеются шаблоны для объявления на английском языке в *Organization Blocks (Организационные блоки).* Если Вам не требуется стартовая информация, то достаточно объявить первые 20 байт, например, как поле (как показано на рис. 18.3).

Инкрементное программирование

Address (адрес)	Declaration (объявление)	Name (имя)	Type (тип)	Initial value (начальное значение)
0.0	temp	SINFO	INT	ARRAY [1..20]
*1.0	temp			BYTE
20.0	temp	Lbyte	INT	ARRAY [1..16]
*1.0	temp		BYTE	BYTE

Программирование, ориентирование на создание исходных текстов программы

```
VAR_TEMP
  SINFO : ARRAY[1..20]
  OF BYTE;
  Lbyte : ARRAY[1..16]
  OF BYTE;
END_VAR
```

Рис. 18.3 Пример объявления локальных данных в организационном блоке.

Абсолютная адресация временных локальных данных

Обычно адресация временных локальных данных производится с использованием символьных имен, а абсолютная адресация используется в исключительных случаях. Если Вы знакомы с тем, как хранятся данные в L-стеке, Вы можете выработать для себя способ адресации статических локальных данных. Вы также можете посмотреть адреса в таблице объявления переменных в скомпилированном блоке.

Идентификатором адреса для временных локальных данных является L; при этом биты адресуются идентификатором L, байты - с помощью идентификатора LB, машинные слова - с помощью идентификатора LW, а двойные слова - с помощью идентификатора LD.

Пример:

Для абсолютной адресации Вы планируете использовать 16 байтов временных локальных данных, к отдельным значениям которых в дальнейшем Вы желаете иметь доступ как в формате байта, так и побитно. Создайте данную область как поле прямо с начала области

локальных данных с начальным адресом, равным 0.

В организационном блоке Вы должны поместить объявление этого поля сразу же за объявлением поля для стартовой информации, таким образом, адресация Вашего поля должна начинаться с адреса 20.

Примечание:

Абсолютная адресация временных локальных данных возможна только при использовании базовых языков программирования STL, LAD и FBD. При использовании языка SCL адресация временных локальных данных возможна только символьным способом.

В главе 26 "Прямой доступ к переменным" показано, как узнать адреса переменных во временных локальных данных во время выполнения программы.

Тип данных ANY

Переменные временных локальных данных могут быть объявлены, как исключение, с типом ANY.

При использовании языка STL Вы имеете возможность сгенерировать указатель типа ANY, который может изменяться во время выполнения программы. Более подробную информацию Вы можете найти в разделе 26.3.3 ""Переменный" указатель ANY".

При использовании языка SCL Вы можете временной переменной типа ANY назначать адрес другой (сложной) переменной во время выполнения программы. Более подробную информацию Вы можете найти в разделе 29.2.4 "Временные локальные данные".

18.1.6 Статические локальные данные

Статические локальные данные - это те адреса, которые функциональный блок сохраняет в своем экземплярном блоке данных.

Статические локальные данные - это "память" функционального блока. Эти данные сохраняются до тех пор, пока программа не изменит их, так же как адреса данных в блоках глобальных данных.

Объем памяти, требуемый для размещения статических локальных данных, ограничивается типом данных используемых переменных и максимальным размером блока, определяемым типом используемого CPU.

Объявление статических локальных данных

Объявление статических локальных данных производится в разделе объявлений функционального блока:

- при инкрементном программировании статические локальные данные помечаются в столбце "Declaration" ("Объявление") словом "stat" (статический);
- при программировании, ориентированном на создание программы путем ввода исходного текста, статические локальные данные объявляются между ключевыми словами: VAR и END_VAR.

На рис. 18.2 в разделе 18.1.5 "Временные локальные данные" представлен пример объявления переменных в функциональном блоке. Сначала объявляются параметры блока, затем объявляются статические локальные данные и, наконец, временные локальные данные.

Статические локальные данные сохраняются в экземплярном блоке после параметров блока в порядке их объявления и в соответствии с типами данных.

Более подробно материал о хранении данных в блоках данных изложен в разделе 26.2 "Хранение переменных".

Символьная адресация статических локальных данных

Адресацию статических локальных данных можно производить с использованием символьных имен. Для этого данным должны быть назначены имена в соответствии с правилами, принятыми для символьных имен (символов) локальных для блока данных.

Имеется возможность доступа к статическим локальным данным с помощью тех операций, которые действительны для использования в сочетании с адресами данных в блоках глобальных данных.

Пример:

Функциональный блок "Totalizer" прибавляет входное значение (input value) к значению, сохраненному в статических локальных данных, (stored value) и после этого сохраняет сумму опять в статических локальных данных. При следующем вызове блока новое входное значение вновь будет прибавлено к сохраненному в статических локальных данных значению, после чего результат вновь будет сохранен, и так далее (см. рис. 18.4. (вверху)).

Total - это переменная в блоке данных "TotalizerData", который является экземплярным блоком данных для функционального блока "Totalizer" (Вы можете самостоятельно задавать имена всех блоков в таблице символов Symbol Table в соответствии с правилами). Экземплярный блок данных имеет структуру данных функционального блока. В данном примере экземплярный блок данных содержит две переменные формата INT с именами *In* и *Total.*

Доступ к статическим локальным данным со стороны функционального блока

Статические локальные данные обычно обрабатываются только в функциональном блоке. Тем не менее, хранятся они в блоке данных. блока данных. Вы можете получить доступ к статическим локальным данным в любое время тем же образом, каким Вы имеете доступ к переменным в блоке глобальных данных с помощью следующей формы адресации: *"ИмяБлокаДанных"."ИдентификаторАдреса".*

В нашем маленьком примере блок данных имеет имя *"TotalizerData",* а переменная адресуется как *Total*. Доступ к переменной может иметь следующую форму:

```
L   "TotalizerData".Total;
T   MW 20;
L   0;
T   "TotalizerData".Total;
```

FB "Totalizer" ("Сумматор")

Address адрес	Declaration объявление	Name имя	Type тип
+0.0	in	In	INT
+2.0	stat	Total	INT

```
      L    #In;
      L    #Total;
      +I   ;
      T    #Total;
```

DB "TotalizerData"

Address адрес	Declaration объявление	Name имя	Type тип
+0.0	in	In	INT
+2.0	stat	Total	INT

FB "Evaluation" ("Опрос")

Address адрес	Declaration объявление	Name имя	Type тип
0.0	in	Add	BOOL
0.1	in	Delete	BOOL
2.0	stat	EM_Add	BOOL
2.1	stat	EM_Del	BOOL
4.0	stat	Memory	Totalizer

```
      A    #Add;
      FP   #EM_Add;
      JCN  M1;
      CALL #Memory
       (In := "Value2");
M1:   A    #Delete;
      FP   #EM_Del;
      JCN  End;
      L    #Memory.Total;
      T    "Result";
      L    0;
      T    #Memory.Total;
```

DB "EvaluationData"

Address адрес	Declaration объявление	Name имя	Type тип
0.0	in	Add	BOOL
0.1	in	Delete	BOOL
2.0	stat	EM_Add	BOOL
2.1	stat	EM_Del	BOOL
4.0	stat:in	Memory.In	INT
6.0	stat	Memory. total	INT

*Выбрав опцию меню: **Data view** (Просмотр данных) для блока данных, можно просмотреть отдельные переменные с их полными именами.*

Одновременно Вы можете видеть соответствующие абсолютные адреса.

Рис. 18.4 Пример статических локальных данных и локальных экземпляров.

Локальные экземпляры

При вызове функционального блока обычно для вызова назначается также экземплярный блок. Функциональный блок сохраняет свои параметры и свои статические локальные данные в этом экземплярном блоке.

Начиная с STEP 7 V2, Вы можете создавать "мультиэкземпляры", что означает, что Вы можете вызывать одни функциональные блоки в других функциональных блоках. Статические локальные данные (и параметры блока) вызванного функционального блока являются подмножеством статических локальных данных вызывающего блока. Для этого требуется, чтобы вызываемый и вызывающий функциональные блоки имели версию 2, чтобы они могли быть обработаны в режиме "мультиэкземпляра". Таким путем можно организовывать вложение вызовов функциональных блоков с глубиной вложения до 8 вызовов.

Пример (рис. 18.4 (внизу)): В статических локальных данных функционального блока "Evaluation" объявлена переменная *Memory*, которая соответствует функциональному блоку "Totalizer" и имеет такую же структуру. Теперь Вы можете вызывать функциональный блок "Totalizer" посредством переменной *Memory*, без определения блока данных, так как данные для *Memory* размещены внутриблочно ("block-local") в статических локальных данных (*Memory* является "локальным экземпляром" блока "Totalizer").

Доступ к статическим локальным данным *Memory* в программе функционального блока "Evaluation" выполняется таким же способом, как и к компонентам структуры с определением имени структуры (*Memory*) и имени компонента (*Total*).

Экземплярный блок данных "EvaluationData", следовательно, содержит переменные *Memory.In* и *Memory.Total*, к которым Вы можете также обращаться как к глобальным переменным, например, следующим образом: *"EvaluationData".Memory.Total*.

Вы можете найти пример использования локального экземпляра в функциональных блоках FB 6, 7 и 8 в программе "Program Flow Control" на прилагаемой дискете. Пример в разделе 19.5.3 "Пример установки" содержит дополнительные варианты применения локальных экземпляров.

Абсолютная адресация статических локальных данных

Обычно адресация статических локальных данных производится с использованием символьных имен, а абсолютная адресация используется в исключительных случаях. Внутри функционального блока экземплярный блок данных открывается посредством регистра DI. Идентификатором для адресов в блоке данных, для статических локальных данных также как параметров блока является идентификатор DI. Биты адресуются идентификатором DIX, байты - с помощью идентификатора DIB, слова - с помощью идентификатора DIW, а двойные слова - с помощью идентификатора DID.

Если Вы знакомы с тем, как хранятся данные блоке данных, Вы можете выработать для себя способ адресации статических локальных данных. Вы также можете посмотреть адреса в таблице объявления переменных в скомпилированном блоке. Но будьте внимательны! *Эти адреса связаны с запуском экземпляра.* Они действительны только в случае, если Вы вызываете функциональный блок с блоком данных. Если Вы вызываете

функциональный блок как локальный экземпляр, соответствующие локальные данные этого локального экземпляра располагаются внутри экземплярного блока данных вызывающего функционального блока. Вы можете просмотреть абсолютные адреса, например, в скомпилированном экземплярном блоке данных, в котором содержатся все локальные экземпляры. Для просмотра адресов отдельных локальных данных выберите опции меню: *Select View -> Data View (Выбор просмотра -> Просмотр данных).*

Вернемся к нашему примеру. В функциональном блоке "Totalizer" переменная *Total* может быть адресована с помощью DIW 2, если FB "Totalizer" вызывается с блоком данных (см. назначение адресов в DB "TotalizerData"), и с помощью DIW 6, если FB "Totalizer" вызывается как локальный экземпляр в блоке FB "Evaluation" (см. назначение адресов в DB "EvaluationData").

Тем не менее, при программировании функционального блока, если пока неизвестно, будет ли он вызываться с блоком данных или будет вызываться как локальный экземпляр, то возникает вопрос: как в этом случае назначать абсолютные адреса статическим локальным данным? Коротко говоря, в этом случае к адресу переменной прибавляется смещение локального экземпляра из адресного регистра AR2. (См. главу 25 "Косвенная адресация" и главу 26 "Прямой доступ к переменным" для получения подробной информации по данному вопросу).

Примечание:

Абсолютная адресация статических локальных данных возможна только при использовании базовых языков программирования STL, LAD и FBD. При использовании языка SCL адресация статических локальных данных возможна только символьным способом.

18.2 Функции для блоков данных

Вы можете хранить данные Вашей программы в блоках данных. В принципе, для хранения данных Вы можете также использовать область меркеров (bit memory area). Тем не менее, используя блоки данных, Вы имеете значительно больше возможностей, в плане объема данных, структурирования данных и типов данных.

В данной главе будет показано

- как работать с адресацией данных,
- как вызывать блоки данных,
- как создавать, удалять и тестировать блоки данных при выполнении программы.

Существует возможность использовать блоки двух видов: *блоки глобальных данных (global data blocks),* которые не назначаются никакому кодовому блоку, и *экземплярные блоки (instance data blocks)*, которые назначаются функциональным блокам. Данные блоков глобальных данных являются, попросту говоря, "свободными" данными, которые может использовать каждый кодовый блок. Вы сами определяете объем, который занимают эти данные, их структуру непосредственно при программировании блоков глобальных данных.

Экземплярные блоки данных содержат только те данные, которые использует связанный функциональный блок; этот функциональный блок определяет структуру данных и место хранения данных в "своем" экземплярном блоке данных.

Число и размер блоков данных определяются типом CPU. Нумерация блоков данных начинается с 1; не может существовать блока данных с именем DB 0. Вы можете использовать каждый блок данных или как блок глобальных данных или как экземплярный блок данных.

Вы должны сначала создать блоки данных, которые Вы будете использовать в Вашей программе или запрограммировав их, так же как кодовые блоки, или используя системную функцию SFC 22 CREAT_DB при выполнении программы.

Блоки данных должны быть сохранены в рабочей (work) памяти для того, чтобы к ним был возможен доступ чтения/записи из пользовательской программы. Вы можете также оставить блоки данных в загрузочной (load) памяти, используя атрибут блока "Unlinked" (ключевое слово UNLINKED используется при создании программы путем ввода исходного текста программы).

Такие блоки данных не занимают место в рабочей (work) памяти. Тем не менее, Вы сможете только считывать блоки данных в загрузочной (load) памяти с помощью системной функции SFC 20 BLCMOV. Такая процедура пригодна для блоков данных с данными параметризации или технологическими (recipe) данными, которые требуются относительно редко для управления установкой или процессом.

Если Вы установили атрибут "The data block is writeprotected in the programmable controller" ("Блок данных защищен от записи в PLC") в свойствах блока (что соответствует использованию ключевого слова READ_ONLY при создании программы путем ввода исходного текста программы), в дальнейшем Вы сможете только считывать данные из этого DB.

18.2.1 Два регистра блоков данных

В CPU есть два "регистра блоков данных" (data block register) для обработки адресованных данных. В этих регистрах содержатся номера обрабатываемых в текущем времени блоков; имеются в виду блоки, в которых находятся обрабатываемые в настоящий момент данные. Перед тем, как получить доступ к адресу, содержащемуся в блоке данных, Вы должны сначала открыть этот блок данных. Если Вы при адресации данных указываете полный их адрес (о способах спецификации блоков данных см. ниже), то Вам нет необходимости заботиться об открытии блока данных и об использовании регистров блоков данных. Редактор сам создаст необходимые инструкции из Ваших данных.

Редактор использует один из регистров блоков данных преимущественно для доступа к блокам глобальных данных, а второй регистр блоков данных используется для доступа к экземплярным блокам данных. В соответствии с назначением регистров, они имеют следующие идентификаторы: регистр для доступа к блокам глобальных данных DB-регистр ("Global data block register") и регистр для доступа к экземплярным блокам данных DI-регистр ("Instance data block register").

Обработка данных регистров CPU происходит абсолютно одинаково. Каждый блок данных может быть открыт с помощью одного из двух регистров (или же с помощью двух регистров одновременно). При загрузке в аккумулятор слова данных, Вы должны задать, какой из двух возможных открытых блоков содержит это слово данных. Если обрабатываемый блок данных открыт с помощью DB-регистра, то слово данных должно содержать DBW; если обрабатываемый блок данных открыт с помощью DI-регистра, то слово данных должно содержать DIW. Остальные форматы данных также имеют соответствующие обозначения (см. табл. 18.2).

Таблица 18.2 Адресация данных

Адресуемые данные	Блок данных открыт с помощью	
	DB-регистра	DI-регистра
Бит данных	DBX y.x	DIX y.x
Байт банных	DBB y	DIB y
Слово данных	DBW y	DIW y
Двойное слово данных	DBD y	DID y

x = адрес бита,
y = адрес байта

18.2.2 Адресация данных

Вы можете использовать следующие способы организации доступа к данным:

- символьная адресация с указанием полного адреса,
- абсолютная адресация с указанием полного адреса,
- абсолютная адресация с указанием неполного адреса.

Дополнительную информацию по способам адресации Вы можете найти в главе 25 "Косвенная адресация".

Символьная адресация с указанием полного адреса глобальных данных в блоке данных требуют минимальной информации для системы. Для использования абсолютной адресации или для использовании сразу двух регистров блоков данных, Вам необходимо ознакомиться с материалом, изложенным ниже.

Символьная адресация данных

Автор рекомендует пользователям использовать символьную адресацию настолько широко, насколько это возможно.

Символьная адресация

- делает более простым чтение и понимание программы (если в качестве символьных имен используются осмысленные обозначения),

- уменьшает ошибки программирования (редактор сравнивает термы, использованные в таблице символов и в программе; "ошибки перестановки чисел", например, DBB 156 и DBB 165, которые могут происходить при использовании абсолютной адресации, не могут произойти при символьной адресации),
- не требует умения программирования на уровне машинных кодов (не требует знания, какой из блоков данных открыт CPU для обработки в настоящий момент).

При символьной адресации указывается полный адрес (указывается и идентификатор блока, и адрес данных), поэтому адрес данных всегда уникален.

Символьный адрес данных формируется в два этапа:

- Назначение блока данных в таблице символов.

  Блоки данных являются глобальными данными, имеющими уникальные адреса в программе. В таблице символов Вы должны назначить символьное имя (например, Motor1) абсолютному адресу блока данных (например, DB 51).

- Назначение адресов данных в блоке данных.

  Вы должны назначить символьные имена адресам данных (и типам данных) во время программирования блока данных. Имена назначаются только в связанном ("associated") блоке (такие имена являются "внутриблочными" - "block-local"). Вы можете назначать такие же имена другим переменным в другом блоке.

Доступ к адресам данных с указанием полного адреса

При организации доступа к адресам данных с указанием полного адреса Вы должны задавать адрес данных совместно с адресом блока. Такой способ указания адреса может быть использован как с символьными, так и с абсолютными адресами.

```
L   MOTOR1.ACTVAL;
L   DB 51.DBW 20;
```

MOTOR1 является символьным адресом, который Вы должны назначить блоку данных в таблице символов. ACTVAL - это адрес данных, который Вы назначаете во время программирования блока данных. Символьное имя MOTOR1.ACTVAL - это такое же уникальное символьное обозначение адреса данных, как и обозначение DB 51.DBW 20.

Доступ к данным посредством полного адреса возможен только при использовании вместе с регистром блоков глобальных данных (с DB-регистром). При адресации данных с указанием полного адреса редактор выполняет две операции: сначала открывает блок данных с помощью DB-регистра, а затем выполняет операцию доступа к адресам данных.

Вы можете использовать доступ к данным посредством полного адреса при всех разрешенных операциях с адресуемыми типами данных. К таким операциям относятся двоичные логические операции, операции с памятью для адресуемых битов, а также операции загрузки (load) и пересылки (transfer) для адресуемых численных данных. Также Вы можете задавать полный адрес для параметров блоков (настоятельно рекомендуется изучить материал главы 19 "Параметры блоков").

Абсолютная адресация данных

При организации доступа к адресам данных посредством абсолютной адресации Вы должны знать адреса, назначенные редактором для данных, при установке блока (setting up). Вы можете найти эти адреса, выведя их после программирования и компилирования блока данных. При этом в колонке адресов Вы увидите значения абсолютных адресов, с которых начинаются соответствующие переменные.

Такая процедура может быть выполнена для всех блоков данных - тех, которые Вы используете как блоки глобальных данных, и тех, которые Вы используете как экземплярные блоки данных. Таким образом Вы также можете видеть, где редактор хранир параметры блоков и статические локальные данные (в случае функциональных блоков).

Если необходимо вычислить адрес, соответствующую информацию Вы можете найти в разделе 26.2 "Хранение переменных".

Организация доступа к адресам данных, имеющих байтовый размер, одинакова, например, с доступом к меркерам; и в том, и в другом случае используются одинаковые операции (см. табл. 18.3), которые выполняются одинаковым способом.

Таблица 18.3 Операции с блоками данных

Оператор	Значение
A –	Проверка на состояние "1", комбинирование по логике AND (И) с адр.
O –	Проверка на состояние "1", комбинирование по логике OR (ИЛИ) с адр.
X –	Проверка на "1", комбинирование по логике Excl.OR (искл.ИЛИ) с адр.
AN –	Проверка на состояние "0", комбинирование по логике AND (И) с адр.
ON –	Проверка на состояние "0", комбинирование по логике OR (ИЛИ) с адр.
XN –	Проверка на "0", комбинирование по логике Excl.OR (искл.ИЛИ) с адр.
= –	Назначение для адр.
S –	Установка адр.
R –	Сброс адр.
FP –	Проверка наличия фронта (переднего) адр.
FN –	Проверка наличия фронта (заднего) адр.
DBXy.x	Адресация бита с помощью DB-регистра
DIXy.x	Адресация бита с помощью DI-регистра
DBz.DBXy.x	Адресация бита с указанием полного адреса
L –	Загрузка из адр.
T –	Пересылка в адр.
DBBy	Адресация байта с помощью DB-регистра
DBWy	Адресация слова с помощью DB-регистра
DBDy	Адресация двойного слова с помощью DB-регистра
DIBy	Адресация байта с помощью DI-регистра
DIWy	Адресация слова с помощью DI-регистра
DIDy	Адресация двойного слова с помощью DI-регистра
DBz.DBBy	Адресация байта с указанием полного адреса
DBz.DBWy	Адресация слова с указанием полного адреса
DBz.DBDy	Адресация двойного слова с указанием полного адреса

x = адрес бита, y = адрес байта, z = номер блока данных

Если Вы намереваетесь назначать исключительно абсолютные адреса данным в блоке данных, Вы должны зарезервировать требуемое количество байтов посредством объявления поля.

18.2.3 Открытие блока данных

OPN DBx	открытие блока данных с помощью DB-регистра с использованием абсолютной адресации;
OPN DB*name*	открытие блока данных с помощью DB-регистра с использованием символьной адресации;
OPN DIx	открытие блока данных с помощью DI-регистра с использованием абсолютной адресации;
OPN DI*name*	открытие блока данных с помощью DI-регистра с использованием символьной адресации;

Блоки данных открываются независимо от любых условий. Открытие блоков не действует ни на RLO, ни на содержимое аккумуляторов; глубина вложения вызовов блоков также не изменяется.

Открываемый блок должен быть в рабочей (work) памяти.

Пример:

Значение слова данных DBW 10 из блока данных DB 12 должно быть перенесено в слово данных DBW 12 из блока данных DB 13 (см. рис 18.5). Значение слов данных DBW 14 из блоков данных DB 12 и DB 13 должны быть сложены; сумма должна быть сохранена в слове данных DBW 14 из блока данных DB 14.

Вы можете запрограммировать данный пример следующими способами организации доступа к данным: с неполной адресацией и с полной адресацией данных (см. таблицу к рис. 18.5).

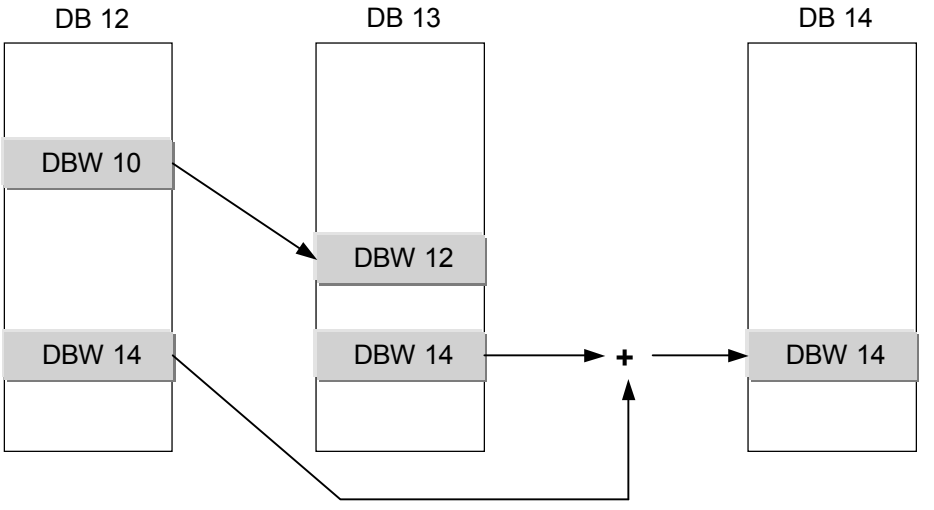


Рис. 18.5 Открытие блоков данных

Таблица к рис. 18.5

Программирование с неполной адресацией данных	Программирование с полной адресацией данных
OPN DB 12; L DBW 10; OPN DB 13; T DBW 12;	L DB 12.DBW 10; T DB 13.DBW 12;
OPN DB 12; L DBW 14; OPN DB 13; L DBW 14; +I ; OPN DB 14; T DBW 14;	L DB 12.DBW 14; L DB 13.DBW 14; +I ; T DB 14.DBW 14;

Когда блок данных открыт, он остается "доступным" ("valid"), до тех пор, пока не будет открыт другой блок данных. При определенных условиях незаметно для пользователя это может обеспечиваться редактором (см. "Особенности, имеющие место при адресации данных"). Например, вызов блока посредством оператора CALL с одновременной передачей параметра может изменять содержимое регистров блоков данных.

При смене обрабатываемого блока с помощью операторов UC и CC содержимое регистров блоков данных сохраняется. При возвращении в вызывающий блок оператор завершения блока восстанавливает содержимое регистров.

18.2.4 Обмен содержимым между регистрами блоков данных

CDB Обмен содержимым между регистрами блоков данных.

Оператор CDB позволяет выполнить обмен содержимым между регистрами блоков данных. Операция обмена данными выполняется независимо от любых условий и не действует ни на биты состояния, ни на другие регистры.

Пример:

С помощью оператора CDB Вы можете "обойти" вызов через регистр DB посредством регистра DI; блок данных пересылается как параметр блока (что невозможно выполнить напрямую).

```
CDB  ;
OPN  #Data2;
CDB  ;
```

В примере с помощью оператора CDB содержимое DB-регистра пересылается в DI-регистр. Затем с помощью параметра блока `#Data2`

открывается блок данных, пересланный как фактический параметр; т.е., номер блока записан в DB-регистр. После нового обмена содержимого регистров, старое значение вновь возвращается в DB-регистр, а DI-регистр содержит параметризированного блока данных.

18.2.5 Размер блока данных и его номер

L	DBLG	Загрузка (Load) размера блока данных, открытого с помощью DB-регистра;
L	DBNO	Загрузка (Load) номера блока данных, открытого с помощью DB-регистра;
L	DILG	Загрузка (Load) размера блока данных, открытого с помощью DB-регистра;
L	DINO	Загрузка (Load) номера блока данных, открытого с помощью DB-регистра;

Инструкция L DBLG загружает (load) размер блока данных, открытого с помощью DB-регистра в аккумулятор accumulator 1. Этот размер представляет собой определенное количество байтов. Инструкция L DILG выполняет аналогичную операцию, только для блока данных, открытого с помощью DI-регистра.

Инструкция L DBNO загружает (load) номер блока данных, открытого с помощью DB-регистра в аккумулятор accumulator 1. Инструкция L DINO выполняет аналогичную операцию, только для блока данных, открытого с помощью DI-регистра.

При выполнении данных операций по внесению номера и размера блока данных в аккумулятор accumulator 1, его предыдущее содержимое пересылается в аккумулятор accumulator 2, как при "нормальном" выполнении операции загрузки (load). Если перед операцией загрузки (load) номера или размера блока данных в аккумулятор не было открыто ни одного блока данных, то в результате в аккумулятор accumulator 1 будут загружены нулевые (0) значения соответственно в качестве номера или размера блока данных.

Невозможно выполнить запись номера блока в регистр блоков данных прямой операцией загрузки; Вы можете влиять на содержимое регистра блоков данных только посредством операций OPN DB, OPN DI и CDB (обмен содержимым между регистрами блоков данных).

18.2.6 Особенности, имеющие место при адресации данных

Изменение назначения в DB-регистре

С помощью следующих функций редактор создает дополнительные операторы, которые могут влиять на содержимое одного или двух регистров блоков данных:

Использование полной адресации данных

Каждый раз, когда при организации доступа к данным используется полная адресация, редактор сначала открывает блок данных с помощью операции OPN DB, затем обеспечивает доступ к адресованным данным. При этом каждый раз производится обновление (переписывание) содержимого DB-регистра. Это же происходит также при инициализации параметров блоков с использованием полной адресации данных.

Операция доступа к параметрам блока

Содержимое DB-регистра также меняется при операциях доступа к следующим параметрам блоков: для функций, для всех параметров блоков сложных типов и функциональных блоков, входных/выходных параметров сложных типов данных.

Операция вызова блока CALL FB

Перед тем как выполнить вызов блока, инструкция CALL FB инициирует сохранение номера текущего экземплярного блока данных в DB-регистре (с помощью обмена содержимого в регистрах блоков данных) и открывает экземплярный блок данных для вызванного функционального блока. Таким образом, связанный экземплярный блок данных всегда открыт в вызванном функциональном блоке. Последующие вызовы блока с помощью инструкции CALL FB вновь изменяют содержимое DB-регистра так, что текущий экземплярный блок данных снова доступен для вызванного функционального блока. Таким образом инструкции CALL FB изменяют содержимое DB-регистра.

Значение DI-регистра для функциональных блоков

Для функциональных блоков данных DI-регистр всегда содержит номер соответствующего экземплярного блока данных. Все операции доступа к параметрам блока или к статистическим локальным данным выполняются с помощью DI-регистра и также посредством адресного регистра AR2 в случае "мультиэкземплярных" функциональных блоков.

Примечание: необходимо учитывать указанное выше постоянное назначение DI-регистра, если Вы изменяете содержимое DI-регистра посредством операций CDB или OPN DI.

Если, к примеру, Вы желаете использовать оба регистра блоков данных одновременно для замены данных, то Вы должны сначала сохранить содержимое регистров, чтобы в последствии восстановить эти значения. Пример, представленный на рис. 18.6, показывает применение соответствующего приема программирования.

Внесение изменений в назначения блоков данных на поздних стадиях

На вкладке "Blocks" ("Блоки") в окне свойств для папки автономных объектов *Blocks* Вы можете определить, какой из способов адресации данных будет иметь преимущество: абсолютная или символьная - при выполнении изменений в назначениях блока данных для уже сохраненных кодовых блоков.

По умолчанию действует установка "Absolute address has priority" ("Абсолютная адресация имеет приоритет") (такие же параметры, как и в предыдущих версиях STEP 7). Такая, принятая по умолчанию, установка означает, что при изменении в разделе объявлений в блоке, абсолютный адрес сохраняется в программе, а символьный адрес соответственно

изменяется. При задании установки "Symbol has priority" ("Символьная адресация имеет приоритет") означает, что при изменении в разделе объявлений в блоке, символьный адрес сохраняется в программе, а абсолютный адрес соответственно изменяется.

```
Var_Temp
   ZW_DB : WORD;  //Промежуточный буфер для блока гло-
                  //бальных данных
   ZW_DI : WORD;  //Промежуточный буфер для экземпярного
                  //блока данных
END_VAR
//Сохранение регистров блоков данных
   L  DBNO;       //Буфер для № блока глобальных данных
   T  ZW_DB;
   L  DINO;       //Буфер для № экземплярн. блока данных
   T  ZW_DI;
//При использовании неполной адресации данных и обоих
//регистров блоков данных
   OPN  DB 12;    //Открыть DB 12 с помощью DB-регистра
   OPN  DI 13;    //Открыть DB 13 с помощью DI-регистра
   L    DBW 16;   //##########
   T    DIW 28;   //# Будьте осторожны с символьной адр.
   L    DID 30;   //# в этой части программы, напр. при
   L    DBD 30;   //# использовании параметров блока,
   +R   ;         //# "внутриблочных" переменных и при
                  //# полной адресации данных
   T    DID 26;   //##########
//Восстановление регистров блоков данных
   OPN  DB[ZW_DB];//Открыть исходный блок глобальных
                  //данных
   OPN  DI[ZW_DI];//Открыть исходный экземплярный блок
```

Рис. 18.6 Пример непосредственного использования двух регистров блоков данных

Пример:

В блоке данных DB 1 слово данных DBW 10 назначено символьному имени *Actual_value*. Если "Data" является символьным именем для блока данных DB 1, то в программе Вы можете загрузить данное слово, например, посредством операции:

```
L    "Data".Actual_value      DB1.DBW 10
```

Если теперь Вы добавите дополнительное слово данных с помощью символьного имени *MaxCurrent* сразу перед словом данных DBW 10, тогда программа будет иметь следующее содержание при последующем открытии (и сохранении) кодового блока:

если "Absolute address has priority" ("Абсолютная адресация имеет приоритет"):

```
L    "Data".MaxCurrent        DB1.DBW 10
```

если "Symbol has priority" ("Символьная адресация имеет приоритет"):

```
L    "Data".Actual_value      DB1.DBW 12
```

Такие же условия имеют место при организации доступа к адресам в блоках глобальных данных (например, доступ к адресам входов), если для этих адресов назначены символьные имена в таблице символов. Подробную информацию по данному вопросу Вы можете найти в разделе 2.5.6 "Приоритет адресов".

18.3 Системные функции для блоков данных

Существуют три системные функции для обработки блоков данных. Параметры этих функций приведены в таблице 18.4.

- SFC 22 CREAT_DB
  С помощью этой функции выполняется создание блока данных
- SFC 23 DEL_DB
  С помощью этой функции выполняется удаление блока данных
- SFC 24 TEST_DB
  С помощью этой функции выполняется тестирование блока данных

Таблица 18.4 Параметры системных функций для обработки блоков данных

SFC	Имя	Объявл.	Тип	Назначение, описание
22	LOW_LIMIT	INPUT	WORD	Нижн. граница для № создаваемого DB
	UP_LIMIT	INPUT	WORD	Верхн. граница для № создаваемого DB
	COUNT	INPUT	WORD	Длина DB в байтах (четное число)
	RET_VAL	OUTPUT	INT	Информация об ошибках
	DB_NUMBER	OUTPUT	WORD	Номер созданного блока данных
23	DB_NUMBER	INPUT	WORD	Номер удаляемого блока данных
	RET_VAL	OUTPUT	INT	Информация об ошибках
24	DB_NUMBER	INPUT	WORD	Номер проверяемого блока данных
	RET_VAL	OUTPUT	INT	Информация об ошибках
	DB_LENGTH	OUTPUT	WORD	Длина DB в байтах
	WRITE_PROT	OUTPUT	BOOL	Если = "1", то DB защищен от записи

18.3.1 Создание блока данных

С помощью системной функции SFC 22 CREAT_DB выполняется создание блока данных в рабочей (work) памяти. В качестве номера блока данных системная функция принимает наименьшее свободное число в диапазоне чисел, заданных входными параметрами LOW_LIMIT (нижний предел) и UP_LIMIT (верхний предел). Граничные значения сами входят в диапазон возможных значений для номера блока. Если оба граничных значения одинаковы, то создаваемый блок будет иметь данное значение. Выходной параметр DB_NUMBER содержит фактический номер созданного блока данных. С помощью входного параметра COUNT Вы задаете размер (длину) создаваемого блока данных.

Длина блока данных соответствует числу байтов данных и при этом должна быть четным числом.

Создание блока данных и вызов блока данных не одно и то же. Текущий (обрабатываемый блок) остается доступным ("valid") блоком. Блок данных, созданный с помощью системной функции, содержит случайные данные. Для использования блока данных, созданного с помощью системной функции, необходимо сначала определить значения данных, содержащихся в блоке. Только после этого можно будет считывать эти данные в программе.

Если при выполнении системной функции происходит ошибка, то блок данных не создается, выходной параметр DB_NUMBER остается неопределенным, а номер ошибки выдается как значение функции.

18.3.2 Удаление блока данных

С помощью системной функции SFC 23 DEL_DB выполняется удаление блока данных в RAM памяти (рабочей [work] и загрузочной [load] памяти). Номер удаляемого блока данных определен в инструкции во входном параметре DB_NUMBER. При выполнении данной системной функции блок данных не должен быть открыт, иначе CPU перейдет в состояние STOP.

Блок данных, созданный с ключевым словом UNLINKED (несвязанный), а также блок данных, расположенный в модуле памяти FEPROM не может быть удален с помощью системной функции SFC 23.

Если при выполнении системной функции происходит ошибка, то блок данных не удаляется, а номер ошибки выдается как значение функции.

18.3.3 Тестирование блока данных

С помощью системной функции SFC 24 TEST_DB выдает число байтов для блока данных в рабочей (work) памяти в выходном параметре DB_LENGTH, а также ID защиты от записи в выходном параметре

WRITE_PROT. Номер тестируемого блока данных Вы задаете в параметре DB_NUMBER.

Если при выполнении системной функции происходит ошибка, то блок данных не создается, выходной параметр остается неопределенным, а номер ошибки выдается как значение функции.

18.4 Null-операции (нуль-операции)

Null-операции (нуль-операции) не создают никакого воздействия при выполнении программы. В языке программирования STL есть операторы NOP 0, NOP 1 и BLD, используемые в качестве Null-операций (нуль-операций).

18.4.1 Операторы NOP

Вы можете использовать операторы NOP 0 (набор битов 16 x "0") и NOP 1 (набор битов 16 x "1") для использования в инструкциях, которые не выполняют никакого действия.

Примечание:

Необходимо отметить, что Null-операции (нуль-операции) занимают в памяти определенное пространство (2 байта) и для выполнения инструкции требуют определенного времени.

Пример:

Необходимо, чтобы в строке с меткой перехода присутствовала инструкция. Если необходимо использовать операцию перехода, но не нужно выполнять никаких операций в строке с меткой перехода, то Вы можете в этой строке использовать оператор NOP 0.

```
          A    I 1.0
          JC   MXX1
          ...
MXX1:     NOP  0
```

Вы можете ввести пустую строку для лучшей читаемости программы простым вводом строки комментария (пустой строки) (это не требует использования пространства памяти пользователя и не добавляет потерь в общее время выполнения программы, так как не содержит кода).

18.4.2 Оператор отображения программы BLD

Редактор использует инструкцию отображения программы BLD nnn для включения информации декомпиляции в программу.

Сами операторы BLD не отображаются.

19 Параметры блоков

Из этой главы Вы узнаете, как использовать параметры блоков; кроме того, Вы узнаете, как

- объявлять (декларировать) параметры блока,
- работать с параметрами блока,
- инициализировать параметры блока,
- последовательно передавать ("pass on") параметры блока.

Параметры блока представляют собой интерфейс передачи данных между вызывающим и вызванным блоками. Все функции, в частности, функции блоков могут выполняться посредством параметров блоков.

19.1 Параметры блока: общая информация

19.1.1 Определение параметров блока

Параметры блока дают возможность передать данные для выполнения инструкций и функций блока.

Пример: необходимо записать блок с функцией сумматора (adder), который Вы хотите использовать в своей программе несколько раз для различных переменных. Переменные будут передаваться как параметры блока; в нашем примере используются три входных параметра и один выходной (см. рис. 19.1). Так как для функционирования сумматора нет необходимости сохранения внутренних значений, то для данной задачи может быть использована функция.

Параметр блока необходимо определять как "входной" параметр (input parameter), если требуется только проверить (Check - т.е., опросить) или загрузить (Load) его значение в программе блока. Если значение параметра только записывается (операции Set, Reset, Assign, Transfer), то Вы имеете дело с "выходным" параметром (output parameter). Если же параметр может быть как записан, так и считан, то Вы должны использовать параметр типа "входной/выходной" (in/out parameter). Редактор не проверяет вариант использования параметров.

19.1.2 Обработка параметров блока

В программе сумматора имена параметров блока являются как бы "вместилищем" для содержимого переменных, которые в дальнейшем будут передаваться в блок для обработки.

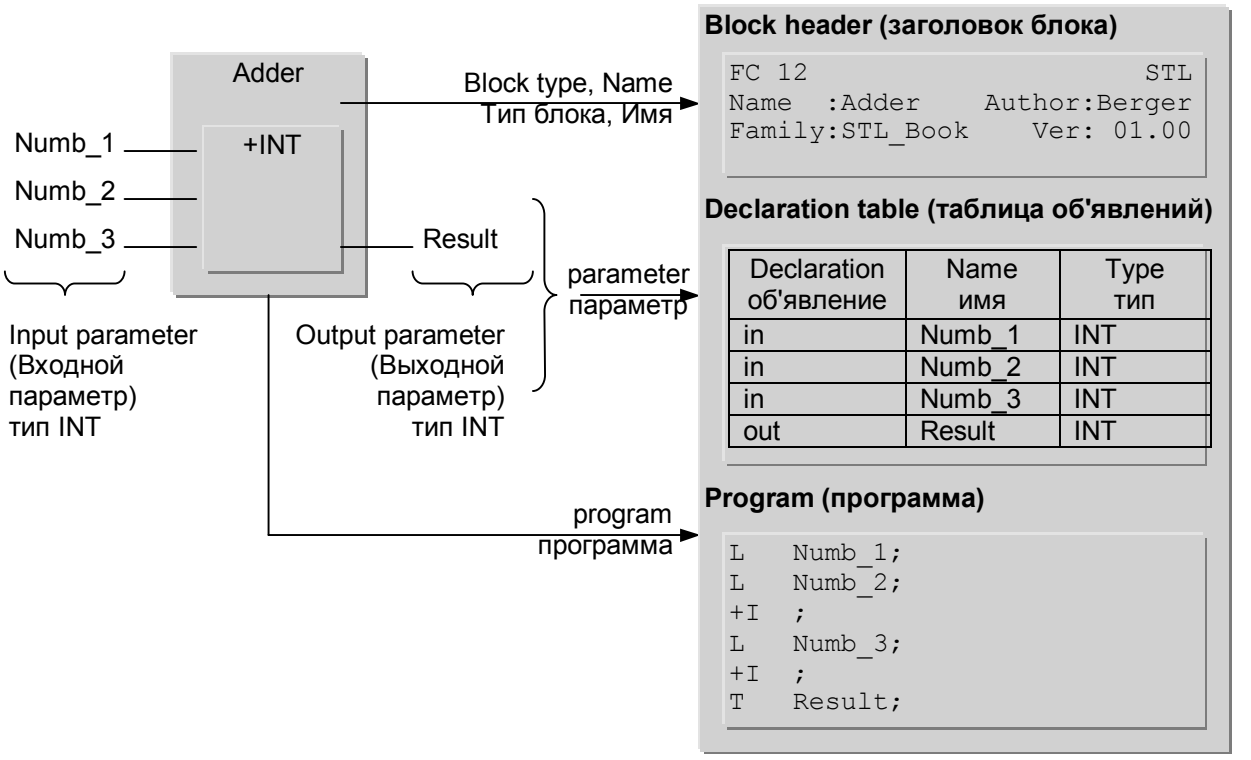


Рис. 19.1 Пример использования параметров блока

Вы можете использовать параметры блока также, как при символьной адресации переменных; в программе они называются *формальными параметрами (Formal parameters).*

Функция "Adder" может быть вызвана в Вашей программе несколько раз. При каждом вызове Вы можете передавать различные значения для операции суммирования с помощью параметров блока (см. рис. 19.2). Значения могут быть константами, адресами или переменными; они называются *фактическими параметрами (Actual parameters).*

Во время выполнения программы CPU заменяет формальные параметры фактическими параметрами. При первом вызове в примере (рис. 19.2) складываются значения из слов MW 30, MW 32 и MW 34, после чего результат сложения записывается в слово MW 40. Другой блок с другими фактическими параметрами вызывает функцию "Adder" второй раз, в результате чего происходит суммирование данных, заключенных в словах DBW 30, DBW 32 и DBW 34 блока данных DB 10, и сохранение суммарного значения в слове данных DBW 40 блока данных DB 10.

19.1.3 Объявление (declaration) параметров блока

Параметры блока должны быть определены в разделе объявлений блока, при программировании этого блока.

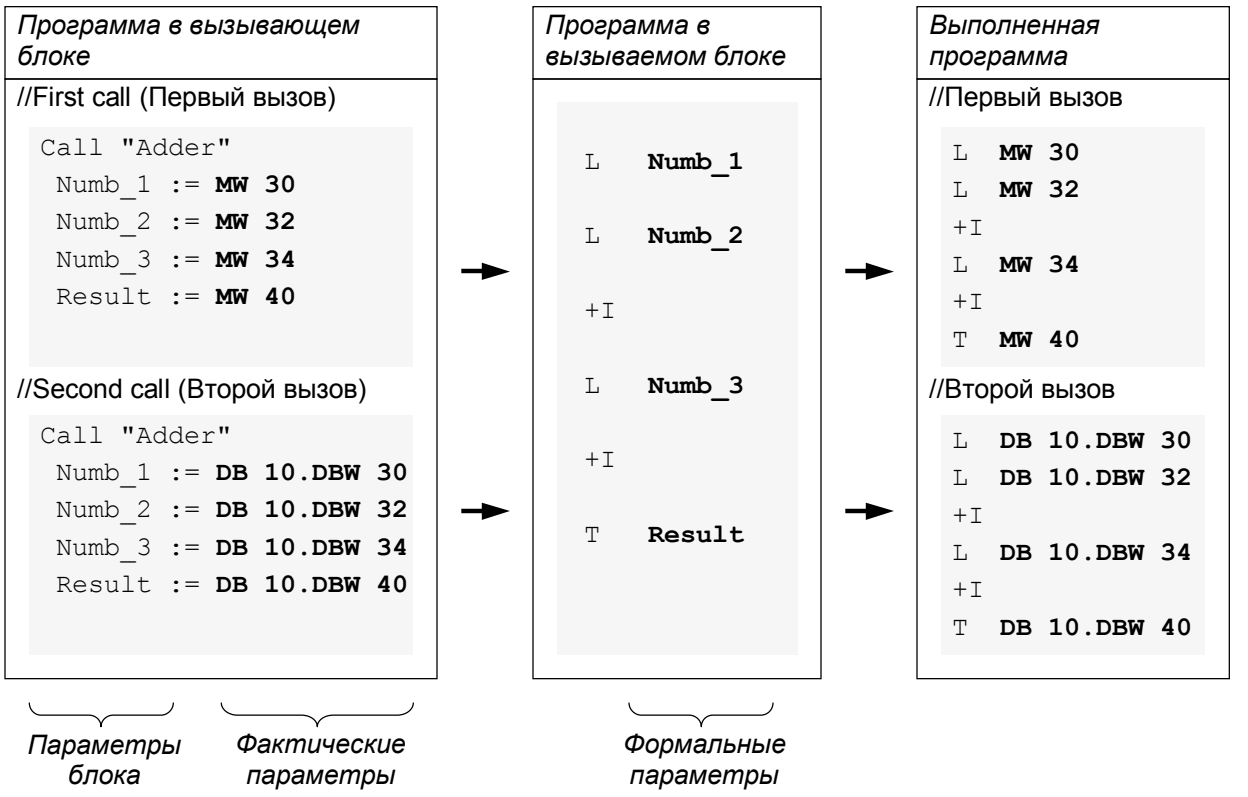


Рис. 19.2 Вызов блока с параметрами

При инкрементном способе программирования Вы просто дополняете список параметров. При способе программирования, ориентированном на создание исходных текстов, Вы определяете параметры блока в особых разделах (рис. 19.3). При этом используются следующие ключевые слова: VAR_INPUT - для входных параметров, VAR_OUTPUT - для выходных параметров и VAR_IN_OUT - для входных/выходных параметров.

Предопределение параметров является необязательным (опциональным) моментом при программировании и имеет смысл только для функциональных блоков, если значение параметра должно быть сохранено. Это касается всех параметров блоков простых типов, а также входных и выходных параметров сложных типов. Заполнение столбца комментариев для параметров необязательно и может быть выполнено в любой момент.

Имя параметра (*Block parameter name*) может содержать до 24 символов. При этом оно должно состоять только из алфавитно-цифровых символов (без национальных элементов, таких как умляут из немецкого алфавита) и символа подчеркивания. Символы в верхнем и нижнем регистрах различаются. Имя не может совпадать ни с одним из ключевых слов.

Символы в верхнем и нижнем регистрах не различаются при вводе имени параметра. При выводе программы редактор использует регистр символов, установленный при объявлении имени параметра блока.

При указании типа данных *Data type* для параметров разрешены все типы: простые *(elementary)*, сложные *(complex)* и пользовательские *(user-defined)*.

Инкрементное программирование

Address (адрес)	Declaration (объявление)	Name (имя)	Type (тип)	Initial value (начальное значение)	Comment (комментарий)
0.0	in	Manual	BOOL	TRUE	Ручное управление
2.0	in	Setpoint	INT	10000	Заданная скорость двигателя Motor 1
4.0	in	Characteristic	ANY		Указатель Point на область данных
14.0	out	Actual_value	INT	0	Скорость двигателя Motor 1
16.0	out	Temperature	REAL	0.000000e+00	Температура двигателя Motor 1
20.0	out	Message	WORD	W#16#0	Сообщение о сбое
22.0	in_out	EM	ARRAY[1..16]		Меркер фронта
*0.1	in_out		BOOL		
*28.0	in_out	Interface	DWORD	DW#16#0	Интерфейс для двигателя Motor 2

Программирование, ориентирование на создание исходных текстов программы

```
VAR_INPUT
  Manual         : BOOL  := TRUE;        //Ручное управление
  Setpoint       : INT   := 10000;       //Заданная скорость для Motor 1
  Characteristic : ANY;                  //Указатель на область данных
END_VAR
```

```
VAR_OUTPUT
  Actual_value   : INT   := 0;           //Скорость двигателя Motor 1
  Temperature    : REAL  := 0.0;         //Температура двигателя Motor 1
  Message        : WORD  := 16#0000;     //Сообщение о сбое
END_VAR
```

```
VAR_IN_OUT
  EM             : ARRAY[1..8] OF BOOL;  //Меркер фронта
  Interface      : DWORD := 16#00000000; //Интерфейс для двиг. Motor 2
END_VAR
```

Рис. 19.3 Пример объявления параметров блока для инкрементного способа программирования и для создания исходных текстов программы

Кроме того, Вы можете задавать типы параметров для параметров блоков.

STEP 7 сохраняет имена параметров блоков в невыполняемых разделах блоков на носителях данных в программаторе PG. В рабочей (work) памяти CPU (в скомпилированном блоке) содержатся только объявленные типы (в столбце *Declaration*) и типы данных (в столбце *Type*). Поэтому изменения в программе блока, которые делаются в интерактивном (online) режиме в памяти CPU, должны быть дополнительно продублированы и на носителях данных в программаторе PG, чтобы сохранить заданные пользователем имена.

Если не обновить данные в PG или блоки перенести из CPU в программатор PG, то соответствующие неисполняемые разделы блока будут либо удалены, либо переписаны. Затем редактор перезаписывает имена, заменяя их, для отображения на экране дисплея или для печати (входные параметры при этом получают имена вида INn, выходные параметры получают имена вида OUTn и входные/выходные параметры получают имена вида INOUTn, где n является порядковым номером, начинающимся с 1).

19.1.4 Объявление (declaration) значения функции

Такой параметр блока как значение функции (в случае применения блока-функции) является особым выходным параметром. Он имеет имя RET_VAL (или ret_val) и определяется как первый выходной параметр.

Тип значения функции может относиться к любому из простых типов данных. Кроме того, к допустимым типам в этом случае также относятся следующие типы данных: DATE_AND_TIME, STRING, POINTER, ANY и пользовательский тип UDT. Однако, тип значения функции не может относиться к таким типам данных, как ARRAY и STRUCT.

В рассмотренном выше примере функция "Adder" может быть также запрограммирована с присвоением значения параметра Result значению функции.

Программирование, ориентированное на создание исходных файлов программы

При программировании, ориентированном на создание исходных файлов программы, Вы должны определить функциональное значение (значение функции), задав тип данных для значения функции после типа блока и разделив эти идентификаторы двоеточием:

```
FUNCTION FC 12 : INT
VAR_INPUT
  Numb_1 : INT;
  Numb_2 : INT;
  Numb_3 : INT;
END_VAR
BEGIN
  L   Numb_1;
  L   Numb_2;
```

```
  +I  ;
  L   Numb_3;
  +I  ;
  T   RET_VAL;
END_FUNCTION
```

В примере значение функции RET_VAL принадлежит к типу INT. Посредством инструкции "T RET_VAL" значение функции принимается равным итоговой сумме параметров Numb_1, Numb_2 и Numb_3.

Инкрементное программирование

При инкрементном программировании Вы должны назначить имя RET_VAL *первому* выходному параметру в списке выходных параметров в разделе объявления переменных. Этим самым Вы определите этот выходной параметр как значение функции для функции FC.

В программе Вы должны рассматривать значение функции как выходной параметр. В примере посредством инструкции "T RET_VAL" значение функции принимается равным итоговой сумме параметров *Numb_1, Numb_2* и *Numb_3.*

19.1.5 Инициализация (Initialization) параметров блока

При вызове блока Вы инициализируете параметры блока значениями фактических параметров. Эти параметры могут быть константами, абсолютными адресами, данными с полной адресацией или переменными с символьной адресацией. Фактические параметры должны относиться к такому же типу данных, что и параметры блока (см. разд. 19.3 "Фактические параметры").

Начиная со STEP 7 V5.1 Вы должны определять параметры блока в исходной программе точно в том порядке, в котором они определены в разделе объявления блока во время программирования. Вы должны инициализировать все параметры функции при каждом ее вызове. В случае использования функциональных блоков инициализация отдельных или всех параметров блока носит опциональный (т.е., выборочный) характер.

19.2 Формальные параметры

В данном разделе Вы узнаете, как получить доступ к параметрам блока внутри блока. В табл. 19.1 показано, что нет ограничений для доступа к параметрам простых типов, к компонентам полей или структур, к функциям таймеров и счетчиков.

Доступ к параметрам сложных типов и к параметрам типов POINTER и ANY в языке программирования STL в настоящее время не поддерживается. Тем не менее, Вы можете инициализировать соответствующие блоки или системные блоки, которые имеют такие параметры, с помощью соответствующих переменных.

Тем не менее, в главе 26 "Прямой доступ к переменным" показано, как Вы можете использовать параметры таких типов данных в блоках, которые написали Вы сами.

Таблица 19.1 Доступ к параметрам блока (общий вид)

Тип данных	Допускается для			Доступ к блоку	
	IN	I_O	OUT	возможен	с помощью
Простые типы данных:					
BOOL	x	x	x	x	двоичного опроса, операции с памятью
BYTE, WORD, DWORD, CHAR, INT, DINT, REAL, S5TIME, TIME, DOT, DATE	x	x	x	x	операции загрузки (load) и пересылки (transfer)
Сложные типы данных:					
DT, STRING	x	x	x	-	Невозможно в STL непосредственно
ARRAY, STRUCT					
Отдельные двоичные компоненты	x	x	x	x	двоичного опроса, операции с памятью
Отдельные двоичные компоненты	x	x	x	x	операции загрузки (load) и пересылки (transfer)
Полные переменные	x	x	x	-	Невозможно в STL непосредственно
Типы параметров:					
TIMER	x	-	-	x	всех операций таймера
COUNTER	x	-	-	x	всех операций счетчика
BLOCK_FC, BLOCK_FB	x	-	-	x	вызова с UC и CC[2)]
BLOCK_DB	x	-	-	x	открытия с OPN DB
BLOCK_SDB	x	-	-	-	Невозможно[3)]
POINTER, ANY	x	x	x[1)]	-	Невозможно в STL непосредственно

[1)] Только для функций

[2)] Оператор CC не применим для функций

[3)] Имеет значение только для системных блоков

Параметры блоков типа BOOL

Параметры блоков типа BOOL могут быть отдельными двоичными переменными или двоичными компонентами полей или структур. Вы можете проверять (опрашивать) входные параметры и входные/выходные параметры с помощью подключения (with contacts) ко входам двоичных функций и влиять на выходные параметры и входные/выходные параметры с помощью операций с памятью.

При использовании FC функций-блоков Вы должны назначать значение двоичному выходному параметру и значению функции в блоке или устанавливать (сбрасывать) его.

В табл. 19.2 показаны разрешенные операции с данными типа BOOL. При программировании функции Вы должны использовать формальные параметры, вместо параметров блока xxxx.

После того, как CPU использовал фактический параметр, определенный посредством параметра блока, он обрабатывает соответствующий оператор (см. табл. 19.2) в соответствии с материалом из глав 4 "Двоичные логические операции" и 5 "Операции с памятью".

Таблица 19.2 Доступ параметрам блоков типа BOOL

A	-	Логическая операция AND (И) при проверки на состояние сигнала "1"
AN	-	Логическая операция AND (И) при проверки на состояние сигнала "0"
O	-	Логическая операция OR (ИЛИ) при проверки на состояние сигнала "1"
ON	-	Логическая операция OR (ИЛИ) при проверки на состояние сигнала "0"
X	-	Логическая операция Exclusive OR (исключающее ИЛИ) при проверки на состояние сигнала "1"
XN	-	Логическая операция Exclusive OR (исключающее ИЛИ) при проверки на состояние сигнала "0"
-	xxxx	входного или входного/выходного параметра типа BOOL
-	xxxx	входного параметра типа TIMER
-	xxxx	входного параметра типа COUNTER
S	-	Операция SET (Установка)
R	-	Операция RESET (Сброс)
=	-	Операция присвоения
-	xxxx	выходного или входного/выходного параметра типа BOOL
FP	-	Операция проверки наличия положительного фронта сигнала
FN	-	Операция проверки наличия отрицательного фронта сигнала
-	xxxx	входного/выходного параметра типа BOOL

Параметры блоков числовых типов

Параметры блоков числовых типов занимают 8, 16 или 32 бита (они могут относиться к любым простым типам, кроме типа BOOL). Они могут быть отдельными численными переменными или численными компонентами полей или структур. Вы можете считывать входные параметры и входные/выходные параметры с помощью функции загрузки (load) ко входам двоичных функций, можете записать выходные параметры и входные/выходные параметры с помощью функций передачи (transfer).

При использовании FC Вы должны передать (transfer) значение в численный выходной параметр и в значение функции. До этого Вы не должны выходить из функции.

```
L   xxxx   Загрузка входного или вх/вых параметра
T   xxxx   Передача в выходной или вх/вых параметр
```

При программировании функции Вы должны использовать формальные параметры, вместо параметров блока xxxx.

После того, как CPU использовал фактический параметр, определенный посредством параметра блока, он обрабатывает соответствующий оператор в соответствии с материалом из главы 6 "Функции пересылки данных".

Параметры блоков типов DT и STRING

Прямой доступ к параметрам блоков типов DT и STRING в настоящее время не поддерживается. Однако, при работе с функциональными блоками Вы можете передавать параметры типов DT и STRING в параметры вызываемых блоков.

В главе 26 "Прямой доступ к переменным" показано, как пользователь может сам запрограммировать операции доступа к параметрам указанных выше типов данных.

Параметры блоков типов ARRAY и STRUCT

Прямой доступ к параметрам блоков типов ARRAY и STRUCT возможен в "покомпонентном" режиме, т.е., доступ обеспечивается к отдельным двоичным или численным компонентам данных вышеуказанных сложных типов посредством соответствующих операций (двоичных логических операций, операций с памятью, функций загрузки [load] или пересылки [transfer]).

Доступ к полным переменным (к целым полям или к целым структурам) в настоящее время не поддерживается, а также нет доступа к отдельным компонентам комбинированного или пользовательского типа. Однако, при работе с функциональными блоками Вы можете последовательно передавать ("pass on") параметры типов ARRAY и STRUCT в параметры вызываемых блоков.

В главе 26 "Прямой доступ к переменным" показано, как пользователь может сам запрограммировать операции доступа к параметрам указанных выше типов данных.

Параметры блоков пользовательского типа

Вы можете работать с параметрами пользовательского типа таким же образом, как и с параметрами типа STRUCT.

Прямой доступ обеспечивается к отдельным двоичным или численным компонентам данных пользовательского типа UDT посредством соответствующих операций (двоичных логических операций, операций с памятью, функций загрузки [load] или пересылки [transfer]).

Доступ к полным переменным в настоящее время не поддерживается и также доступ к отдельным компонентам комбинированного или пользовательского типа невозможен. При работе с функциональными блоками Вы можете последовательно передавать ("pass on") параметры типа UDT в параметры вызываемых блоков.

В главе 26 "Прямой доступ к переменным" показано, как пользователь может сам запрограммировать операции доступа к параметрам указанных выше типов данных.

Параметры блоков типа TIMER

В добавление к функциям проверки (опроса), указанным в табл. 19.2, Вы можете программировать обработку параметров блока типа TIMER с помощью следующих операторов:

SP	-	запуск таймера в режиме "импульса" ("pulse")
SD	-	запуск таймера в режиме "задержки включения" ("ON delay")
SE	-	запуск таймера в режиме "расширенного импульса" ("extended pulse")
SS	-	запуск таймера в режиме "задержки включения с памятью" ("retentive ON delay")
SF	-	запуск таймера в режиме "задержки выключения" ("OFF delay")
R	-	функция сброса ("reset")
FR	-	функция разрешения ("enable")
-	xxxx	входной параметр типа TIMER

При программировании функции Вы должны использовать формальные параметры, вместо параметров блока xxxx.

После того, как CPU использовал фактический параметр, определенный посредством параметра блока, он обрабатывает соответствующий STL оператор точно также, как описано в главе 7 "Функции таймера". При запуске функции таймера значение времени "time value" может также быть параметром формата S5TIME.

Параметры блоков типа COUNTER

В добавление к функциям проверки (опроса), указанным в табл. 19.2, Вы можете программировать обработку параметров блока типа COUNTER с помощью следующих операторов:

S	-	установка счетчика ("set")
CU	-	запуск счетчика в режиме "прямого счета" ("count up")
CD	-	запуск счетчика в режиме "обратного счета" ("count down")
R	-	функция сброса ("reset")
FR	-	функция разрешения ("enable")
-	xxxx	входной параметр типа COUNTER

При программировании функции Вы должны использовать формальные параметры, вместо параметров блока xxxx.

После того, как CPU использовал фактический параметр, определенный посредством параметра блока, он обрабатывает соответствующий STL оператор точно также, как описано в главе 8 "Функции счетчика". При запуске функции счетчика значение счетчика "count value" может также быть, например, параметром типа WORD.

Параметры блоков типа BLOCK_xx

OPN	-	функция открытия блока данных (параметр типа BLOCK_DB)
UC	-	вызов функции (параметр типа BLOCK_FC)
UC	-	вызов функционального блока (параметр типа BLOCK_FB)
CC	-	условный вызов функции (параметр типа BLOCK_FC)
CC	-	условный вызов функционального блока (параметр типа BLOCK_FB) (см. текст)
-	xxxx	входной параметр

При программировании функции Вы должны использовать формальные параметры, вместо параметров блока xxxx.

При открытии блока данных посредством параметра блока, CPU всегда использует регистр блоков глобальных данных (DB-регистр).

Функции и функциональные блоки, которые пересылаются с параметрами блоков, сами не должны содержать параметров блоков. Условный вызов блока посредством параметра возможен, только если это параметр функционального блока.

Вы можете также использовать блок данных, который переслали как параметр блока, в качестве экземплярного блока. Так как редактор не проверяет тип блока данных, используемого при выполнении программы, Вы должны сами обеспечивать, чтобы пересылаемый блок данных также соответствовал экземплярному блоку данных для вызванного функционального блока.

Пример:

Вы можете определить параметр блока типа BLOCK_DB с помощью имени *#Data* как экземплярный блок данных для вызванного функционального блока:

```
CALL FB 10, #Data
```

Параметры блоков типов POINTER и ANY

Прямой доступ к параметрам блоков типов POINTER и ANY невозможен.

В главе 26 "Прямой доступ к переменным" показано, как пользователь может сам запрограммировать операции доступа к параметрам блоков типов POINTER и ANY.

19.3 Фактические параметры

При вызове блока Вы инициализируете параметры блока значениями, которые могут быть константами, адресами или переменными, с которыми программа блока должна быть выполнена. Это фактические параметры. При частых вызовах блока в программе, обычно используются

различные значения фактических параметров.

Фактический параметр должен относиться к такому же типу данных, что и соответствующий параметр блока. Вы можете поставить в соответствие двоичному фактическому параметру (например, меркеру) только параметр блока типа BOOL; Вы можете инициализировать параметр блока типа ARRAY только переменной, занимающей такое же поле в памяти. В табл. 19.3 представлен обзор адресуемых данных, которые Вы можете применять как фактические параметры соответствующих типов.

Табл. 19.3 Инициализация фактических параметров

Тип параметра блока	Допустимые фактические параметры
Простой тип	• Простые адреса, полные адреса, константы • Компоненты полей или структур простых типов • Параметр вызывающего блока • Компоненты параметров блока вызывающего блока простых типов
Сложный тип	• Переменные или параметры вызывающего блока
TIMER, COUNTER, BLOCK_xx	• Таймеры, счетчики и блоки
POINTER	• Простые адреса, полные адреса • Указатель области или DB-указатель
ANY	• Переменные любого типа • Указатель ANY

При вызове функции Вы должны инициализировать все параметры блока фактическими параметрами.

При вызове функциональных блоков нет необходимости инициализировать параметры блока. STEP 7 сохраняет все параметры блока простых типов, входные и выходные параметры сложных типов и входные параметры типов TIMER, COUNTER и BLOCK_xx как значения или как числа. Вх/вых параметры сложных типов и входные параметры типов POINTER и ANY сохраняются как указатели на фактические параметры. Так как при этом вводится значащее значение, Вы должны инициализировать по крайней мере параметры с неполной адресацией, по крайней мере при первом вызове. Имеется также возможность прямого доступа к параметрам блока. Так как параметры блока располагаются в блоке данных, Вы можете обрабатывать их как адреса данных.

Пример:

Функциональный блок с экземплярным блоком "Lift_stat_1" управляет двоичным выходным параметром с именем *Up.* Выполняя обработку программы функционального блока после его вызова, Вы можете проверить (опросить) параметр без инициализации выходного параметра:

```
A   "Lift_stat_1".Up
```

Вы можете запрограммировать такую инструкцию проверки (опроса) параметра вместо его инициализации.

Инициализация параметров блоков простых типов данных

Параметры, перечисленные в табл. 19.4, являются допустимыми для использования в качестве фактических параметров простых типов.

Таблица 19.4 Фактические параметры простых типов данных

Адреса	Допускается для			Адрес двоичных данных или символьное имя	Адрес числовых данных или символьное имя
	IN	I_O	OUT		
Входы (образ процесса)	x	x	x	I y.x	IB y, IW y, ID y
Выходы (образ процесса)	x	x	x	Q y.x	QB y, QW y, QD y
Меркеры	x	x	x	M y.x	MB y, MW y, MD y
Периферийные входы	x	-	-	-	PIB y, PIW y, PID y
Периферийные выходы	-	-	x	-	PQB y, PQW y, PQD y
Глобальные данные неполная адресация полная адресация	 x x	 x x	 x x	 DBX y.x DBz.DBX y.x	 DBB y, DBW y, DBD y DBz.DBB y, DBz.DBW y, DBz.DBD y
Временные локальные данные	x	x	x	L y.x	LB y, LW y, LD y
Статические локальные данные	x	x	x	DIX y.x	DIB y, DIW y, DID y
Константы	x	-	-	TRUE, FALSE	Все числовые константы
Компоненты массивов и структур	x	x	x	Полное имя компонента	Полное имя компонента

x = адрес бита, y = адрес байта, z = номер блока данных

Вы можете назначить или абсолютный, или символьный адрес входу, выходу и меркеру. Входные адреса обычно назначаются входным параметрам; выходные адреса обычно назначаются выходным параметрам (тем не менее, это не обязательно). Адреса меркеров допустимы для параметров, объявленных как входныые, выходные или вх/вых параметры.

При использовании неполной адресации данных Вы должны обеспечить, чтобы соблюдалась "корректность" в открытом блоке данных при обращении к параметру блока (в вызванном блоке). Так как редактор может в отдельных случаях заменить блок данных при вызове блока, то применение неполной адресации данных не рекомендуется. Поэтому используйте только полную адресацию данных.

Для временных локальных данных обычно используется символьная адресация. Они размещаются в L-стеке вызывающего блока (и объявлены в вызывающем блоке).

Если вызывающий блок является функциональным блоком, Вы можете также использовать его статические локальные данные как фактические параметры (см. раздел 19.4 "Передача "Pass On" параметров блока").

Для статических данных обычно используется символьная адресация. Если Вы используете абсолютную адресацию с указанием DI-регистра (DI-адресация), то Вы должны обеспечить "корректность" в открытом блоке данных с указанием DI-регистра при обращении к параметру блока (в вызванном блоке).

Примечание: в связи с вышеизложенным надо заметить, что при использовании вызываемых блоков как локальных экземпляров, абсолютные адреса локальных (block-local - внутриблочных) переменных зависит от объявления локального экземпляра в вызванном блоке.

В качестве параметра блока типа BOOL Вы можете применять константу TRUE (ИСТИНА - для состояния сигнала "1") или константу FALSE (ЛОЖЬ - для состояния сигнала "0"). В качестве параметра блока численных типов Вы можете применять любые константы, относящиеся к численным типам. Инициализация посредством присвоения констант имеет смысл только для входных параметров.

Вы можете также инициализировать параметры блока простых типов посредством присвоения компонентов полей или структур при условии, что компоненты полей или структур имеют соответствущий тип данных.

Инициализация параметров блоков сложных типов данных

Каждый параметр блока может быть сложного типа или пользовательского (UDT) типа. Этим параметрам могут быть поставлены в соответствие адреса переменных соответствующих типов, то есть, эти переменные могут выступать как фактические параметры.

Вы можете для инициализации параметров блока типа DT и STRING использвать отдельные переменные или компоненты полей или структур при условии, что они имеют соответствущий тип данных. Инициализация таких параметров не допустима в STL.

Если Вы инициализируете параметры функционального блока переменной STRING, эта переменная должна иметь такую же длину как и параметр блока STRING.

Когда создается переменная STRING во временных локальных данных, предопределение ее значение невозможно, поэтому при этих условиях содержимое данной переменной содержит "случайные" значения. При использовании такой переменной в качестве фактического параметра для IEC-функции Вы должны сначала определить ее соответствующим корректным значением в программе (перед записью значения в STRING-переменную IEC-функция проверяет его на соответствие типу переменной).

Вы можете для инициализации параметров блока типа ARRAY и STRUCT использвать переменные с точно такой же структурой как и параметры блока.

Назначение параметров сложных типов описано в разделе 26.4 "Краткое описание примера фрейма сообщения" в примерах "Создание фрейма сообщения" и "Считывание времени суток (считывание данных TOD)".

Инициализация параметров блоков пользовательских типов данных

Для сложных и громоздких структур данных рекомендуется ввести и использовать пользовательский (UDT) тип данных. Сначала Вы должны определить UDT-тип данных, затем - использовать его, например, для создания переменной в блоке данных или для объявления параметра

блока. После этого Вы можете использовать переменную при инициализации параметра блока. В этом случае фактический параметр (переменная) должна быть точно такого же типа (такой же UDT-структуры) как и параметр блока.

Назначение параметров пользовательских типов описано в разделе 26.4 "Краткое описание примера фрейма сообщения" в примере "Данные фрейма сообщения".

Инициализация параметров блоков типов TIMER, COUNTER и BLOCK_xx

Вы должны инициализировать параметр блока типа TIMER для функции таймера, параметр блока типа COUNTER для функции счетчика. Для параметров типа BLOCK_FC и BLOCK_FB Вы можете использовать только блоки без своих собственных параметров. После этого эти блоки могут быть вызваны посредством оператора UC (а также CC для функциональных блоков). Инициализировать параметр блока типа BLOCK_DB можно блоком данных, открытом в вызванном блоке посредством DB-регистра.

Назначение параметров пользовательских типов описано в разделе 26.4 "Краткое описание примера фрейма сообщения" в примере "Данные фрейма сообщения".

Параметры блоков типов TIMER, COUNTER и BLOCK_xx могут быть только входными параметрами.

Инициализация параметров блоков типа POINTER

Для параметра блока типа POINTER допустимо использовать только указатели (константы). Эти указатели могут являться указателями либо для области данных (32-разрядные), либо для блока данных DB (48-разрядные).

При этом возможна адресация данных простых типов; также возможна полная адресация.

Для функциональных блоков не допускаются выходные параметры типа POINTER.

Инициализация параметров блоков типа ANY

Для параметра блока типа ANY допустимо использовать переменные любых типов. При программировании вызываемого блока Вы должны определить, какие переменные (адреса или типы данных) должны быть

применены в параметрах блока, какие переменные могут примяться. Вы также можете определить константы в формате указателя ANY

"p#[Data_block.]Address Data_type Number"

и таким образом определить область с помощью абсолютной адресацией.

Исключением является инициализация параметра ANY временными локальными данными типа ANY. В этом случае редактор скорее допустит, что указатель типа ANY уже существует во временных локальных данных, чем создаст указатель на переменную. Это дает Вам возможность применить для параметра ANY указатель ANY, которым можно манипулировать во время выполнения программы. "Переменный указатель ANY" может быть, отчасти, полезен при работе с системной функцией SFC 20 BLKMOV (см. пример "Буфер входа" в разделе 26.4 "Краткое описание примера фрейма сообщения").

Для функциональных блоков не допускаются выходные параметры типа ANY.

19.4 Последовательная передача ("Pass On") параметров блока

Последовательная передача ("Pass On") параметров блока - это особая форма доступа и инициализации параметров блока. При такой форме доступа параметры блока вызывающего передаются ("passed on") в параметры вызванного блока. При этом формальный параметр вызывающего блока становится фактическим параметром вызванного блока.

В общем случае, при этом фактический параметр должен быть того же типа, что и формальный параметр (то есть, параметры блока должны соответствовать типам передаваемых данных). Вдобавок, Вы можете применить входной параметр вызывающего блока только как входной параметр вызываемого блока, и, аналогично, использовать выходные параметры. Вы можете применить вх/вых параметр вызывающего блока во всех (входных, выходных и вх/вых) параметрах вызываемого блока.

Существуют ограничения, зависящие от типов данных, которые возникают из-за различий в способе хранения параметров блоков между функциями и функциональными блоками. Параметры блоков могут последовательно передаваться ("pass on") без ограничений и в соответствии с информацией, изложенной в предыдущих параграфах. Сложные типы данных для входов и выходных параметров могут передаваться, если только вызывающий блок является функциональным блоком. Параметры блоков типов TIMER, COUNTER и BLOCK_xx могут передаваться от одного входного параметра к другому, только если вызывающий блок является функциональным блоком. В табл. 19.5 представлены разрешенные и запрещенные пути передачи данных между входными и выходными параметрами для функций и функциональных блоков в зависимости от типа передаваемых данных.

Вы можете последовательно передавать ("pass on") данные типов TIMER, COUNTER и BLOCK_xx при работе с функциями при использовании косвенной адресации. Соответствующему параметру сначала должен быть присвоен тип данных WORD или INT; затем Вы инициализируете его константой или переменной, которая содержит номер таймера, счетчика или передаваемого блока. Вы можете последовательно передавать ("pass on") этот параметр, так как он относится к параметрам простого типа. В последнем (в цепочке передачи - "last") блоке Вы можете использовать функцию загрузки "load" для передачи содержимого параметра в слово временных локальных данных, после чего может быть выполнена функция таймера, счетчика или блока.

Табл. 19.5 Разрешенные комбинации для последовательной передачи ("pass on") параметров

Вызывающий ---> вызываемый (объявленный тип)	FC вызывает FC			FB вызывает FC			FC вызывает FB			FB вызывает FB		
	E	C	P	E	C	P	E	C	P	E	C	P
Input -> Input (Вх -> Вх)	x	-	-	x	x	-	x	-	x	x	x	x
Output -> Output (Вых -> Вых)	x	-	-	x	x	-	x	-	-	x	x	-
In/Out -> Input (Вх/Вых -> Вх)	x	-	-	x	-	-	x	-	-	x	-	-
In/Out -> Output (Вх/Вых -> Вых)	x	-	-	x	-	-	x	-	-	x	-	-
In/Out -> In/Out (Вх/Вых -> Вх/Вых)	x	-	-	x	-	-	x	-	-	x	-	-

E = простые типы данных
C = сложные типы данных
P = параметрические типы TIMER, COUNTER и BLOCK_xx

19.5 Примеры

19.5.1 Пример: ленточный конвейер

Пример показывает передачу состояний сигналов с помощью параметров. Для этой цели мы будем использовать функцию управления ленточным конвейером, работа которой была объяснена в главе 5 "Операции с памятью". Функция управления ленточным конвейером должна быть размещена в функциональном блоке, и все входы и выходы должны быть запрограммированы как параметры блока, таким образом, что функция управления ленточным конвейером может быть использована многократно (для нескольких конвейеров). На рис. 19.4 показаны входные и выходные параметры блока, так же как и использованные статические локальные данные.

В данном блоке параметры распределяются достаточно просто: все двоичные адреса, соответствующие входам должны стать входными параметрами, все двоичные адреса, соответствующие выходам должны стать выходными параметрами, все меркеры должны стать статическими локальными данными.

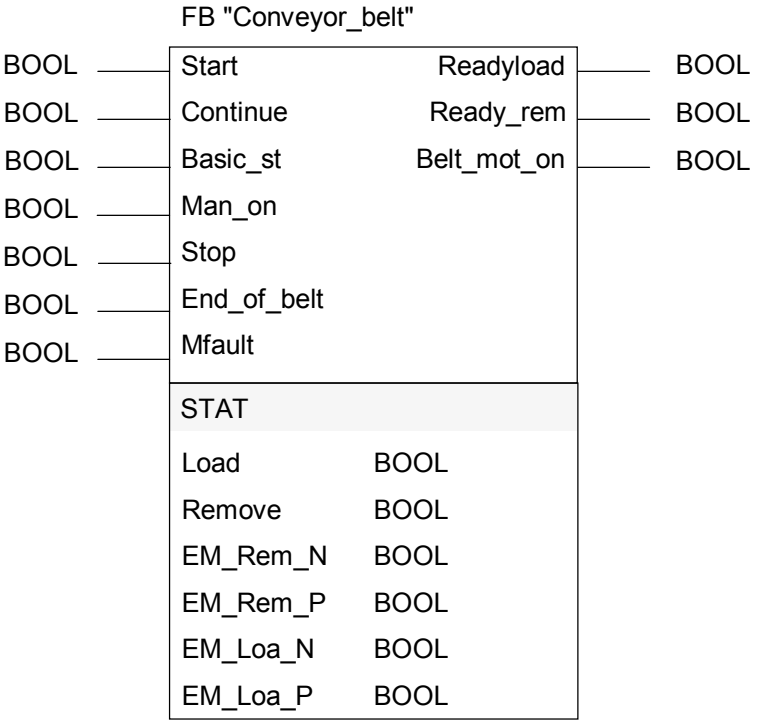


Рис. 19.4 Функциональный блок для примера ленточного конвейера

Вы можете заметить, что имена также слегка изменены, так как при написании имен переменных могут использоваться только алфавитно-цифровые символы, а также символ подчеркивания.

Функциональный блок "Conveyor_belt" предназначен для управления двумя ленточными конвейерами. Поэтому он вызываеися дважды: первый раз - для обработки входов и выходов конвейера 1, второй раз - для обработки входов и выходов конвейера 2. Для каждого вызова функционального блока требуется экземплярный блок данных, в котором хранятся данные конвейера для соответствующего случая. Блок данных для конвейера 1 называется "Belt_data1", блок данных для конвейера 2 называется "Belt_data2" и т.д.

Вы можете найти пример исполняемой программы в библиотеке STL_Book с названием "Conveyor Example" на дискете, приложенной к книге. Исходная программа содержит код функционального блока с входными параметрами, с выходными параметрами и статическими локальными данными. За этой программой следуют программы экземплярных блоков данных; в них достаточно определить функциональный блок в разделе объявлений. Вы можете использовать любой блок данных в качестве экземплярного, например, DB 21 "Belt_data1" и DB 22 "Belt_data2". В таблице символов эти блоки данных имеют тип данных функционального блока.

В конце исходной программы Вы можете увидеть другие два вызова функциональных блоков, такие, например, как в OB 1. Входы и выходы из таблицы символов используются как фактические параметры. В тех случаях, когда такие глобальные символьные имена содержат особые

символы, в программе эти имена должны быть заключены в кавычки.

19.5.2 Пример: счетчик деталей

Пример демонстрирует обработку параметров блока простых типов. Пример "Parts Counter" ("счетчик деталей") из главы 8 "Функции счетчиков" взят за основу для нашей функции. В нашем случае функция выполняется как функциональный блок со всеми глобальными переменными, объявленными или как параметры блока, или как статические локальные данные (Рис. 19.5).

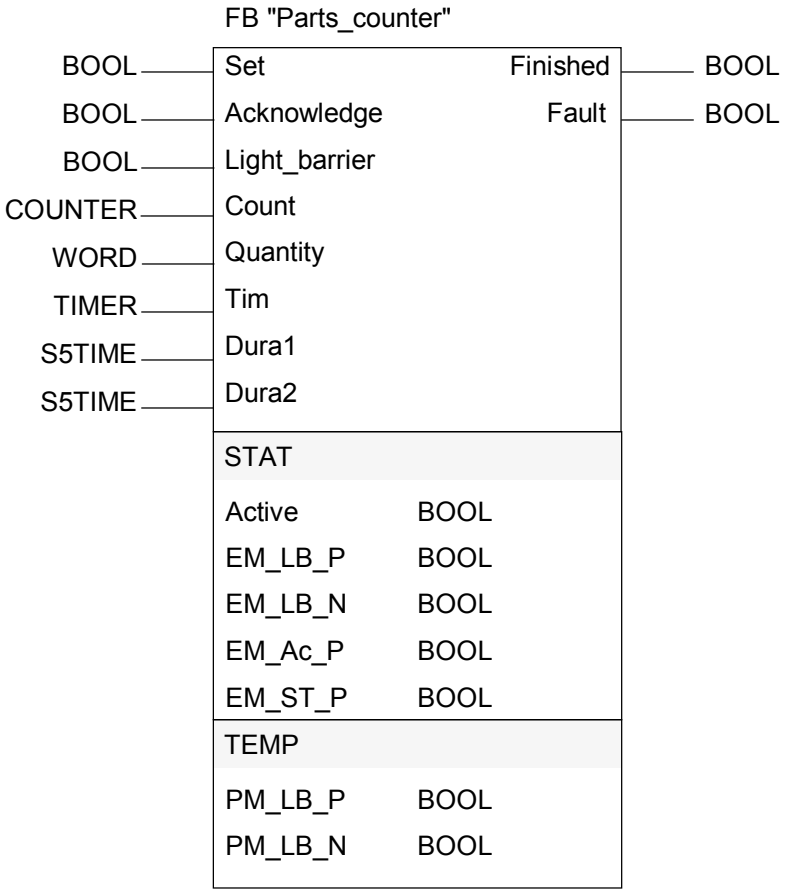


Рис. 19.5 Функциональный блок для примера счетчика деталей

Функции таймера и счетчика передаются с помощью параметров блока типов TIMER и COUNTER. Эти параметры блока должны быть входными параметрами. Начальные значения для функции счетчика (Quantity) и для функции таймера (Dura1 и Dura2) могут также быть переданы как параметры блока; тип данных параметров блока здесь соответствует фактическим параметрам.

Меркеры фронта сохраняются в статических локальных данных, а меркеры импульса сохраняются во временных локальных данных.

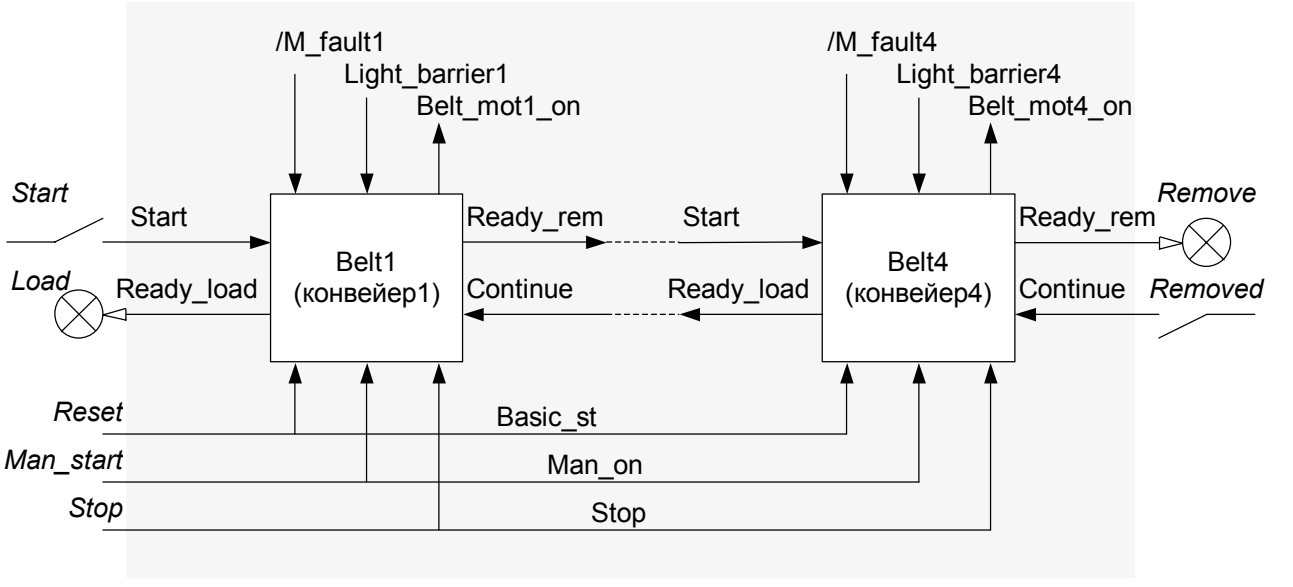
Вы можете найти пример исполняемой программы в библиотеке STL_Book с названием "Conveyor Example" на дискете, приложенной к книге. Исходная программа содержит функциональный блок "Parts_counter", связанный с ним экземплярный блок данных "Count_Dat", а также вызов функционального блока с экземплярным блоком данных.

19.5.3 Пример: подающий механизм

Такие же функции как те, что описаны в предыдущих двух примерах, могут также быть вызваны как локальные экземпляры. В нашем примере это означает, что нам предстоит запрограммировать функциональный блок "Feed" ("Подающий механизм") для управления четырьмя ленточными конвейерами и для подсчета деталей, переданных с помощью этих конвейеров. В данном функциональном блоке блок FB "Conveyor_Belt" ("Ленточный конвейер") вызывается четыре раза, а блок FB "Parts_Counter" ("Счетчик деталей") - только один раз. В нашем случае не для каждого вызова FB вызывается свой экземплярный блок, а все вызываемые функциональные блоки хранят свои данные в экземплярном блоке функционального блока "Feed".

На рис. 19.6 показано, как отдельные блоки управления конвейерами соединяются в единую систему управления (блок FB "Parts_Counter" не показан на рисунке).

Функциональный блок "Feed"



Belt1, ... , Belt4 - функциональные блоки "Conveyor_Belt"

Рис. 19.6 Пример функционального блока программы "Feed"

Запускающий (Start) сигнал подается на вход *Start* блока управления конвейера Belt 1, выход *Ready_rem* подается на вход *Start* блока управления конвейера Belt 2, и т.д. И, наконец, *Ready_rem* от блока управления Belt 4 подается на выход *Remove* системы управления "Feed". Такая же цепочка сигналов проходит в обратном направлении *Removed -> Continue -> Ready_load -> ... -> Load.*

Сигналы *Belt_motX_on, Light_barrierX* и */M_faultX* (отказ мотора) - это отдельные сигналы в каждом из блоков управления конвейерами. Входы *Reset, Man_start* и *Stop* позволяют управлять всеми блоками посредством сигналов *Basic_st, Man_on* и *Stop* соответственно.

Следующий раздел программы функционального блока "Feed" создан таким же путем. Входные и выходные параметры функционального блока показаны на рис. 19.5. Кроме того, числовые значения для счетчика деталей Quantity и таймера Dura1 и Dura2 используются здесь как входные параметры. Мы объявляем данные отдельных блоков управления конвейерами и данные счетчика деталей в статических локальных данных таким же образом, как объявляются данные пользовательского типа UDT, т.е., с указанием имени и типа данных.

Переменная *Belt1* должна содержать структуру данных функционального блока "Conveyor_Belt", так же как и переменные *Belt2, ..., Belt4;* переменная *Check* должна содержать структуру данных функционального блока "Parts_counter".

Программа функционального блока начинается с инициализации общих сигналов для всех блоков управления конвейерами. Здесь мы используем тот факт, что параметры блока функциональных блоков, вызываемых как локальные экземпляры, являются статическими локальными данными в текущем блоке и, поэтому, могут обрабатываться как таковые. Параметр блока *Man_start* текущего функционального блока управляет входным параметром *Man_on* всех четырех блоков управления конвейерами посредством простой операции присвоения. Таким же образом проводится инициализация входных параметров *Basic_st* и *Stop* с помощью параметров блока *Reset* и *Stop* соответственно. Теперь общие сигналы для блоков управления конвейерами инициализированы. (Конечно же Вы можете инициализировать эти сигналы и при вызове функционального блока).

Последовательные вызовы функциональных блоков управления конвейерами содержат параметры блока только для отдельных сигналов соответствующего конвейера, связанные с параметрами функционального блока "Feed". Эти отдельные сигналы представляют собой сигналы от фотодатчиков (*Light_barrierX*), сигналы для и от моторов приводов конвейеров (соответственно сигналы для аварийной остановки двигателей */M_faultX* и сигналы о состоянии моторов приводов *Belt_motX_on*). (Здесь имеется в виду, что при вызове функционального блока не все параметры блока должны быть инициализированы).

Программирование связи между отдельными блоками управления конвейерами производится с использованием операций присвоения.

Функциональный блок FB "Parts Counter" ("счетчик деталей") вызывается как локальный экземпляр, даже несмотря на то, что он не связан с сигналами управления конвейерными лентами. Экземплярный блок данных функционального блока "Feed" содержит в себе данные FB.

Входные параметры для счетчика деталей *Quantity* и таймера *Dura1* и *Dura2* функционального блока "Feed" должны быть установлены только один раз. Это может быть сделано путем задания значений, принимаемых по умолчанию (как в данном примере), или путем прямого присвоения значений при перезапуске программы в OB 100 (если, к примеру, эти три параметра трактуются как глобальные данные).

Данный функциональный блок "Feed" и связанный с ним экземплярный блок "FeedDat" содержатся в библиотеке STL_Book в программе с именем "Conveyor Example". Ниже представлен функциональный блок с экземплярным блоком данных для основной программы.

```
FUNCTION_BLOCK "Feed"
TITLE = Control of several conveyor belts //Управление несколькими
                                          //ленточными конвейерами
//Пример локальных экземпляров "local instances"
//declaration (объявления), calls (вызовы)
NAME        : Feed
AUTHOR      : Berger
FAMILY      : STL_Book
VERSION     : 01.00
VAR_INPUT
  Start     : BOOL      := FALSE;    //Запуск лент конвейеров
  Removed   : BOOL      := FALSE;    //Детали удалены с ленты
  Man_start : BOOL      := FALSE;    //Ручной запуск конвейеров
  Stop      : BOOL      := FALSE;    //Остановка лент конвейеров
  Reset     : BOOL      := FALSE;    //Установка в основное
                                     //  состояние (basic_st)
  Count     : COUNTER;               //Счетчик деталей
  Quantity  : WORD      := W#16#0200;//Число деталей
  Tim       : TIMER;                 //Функция таймера
  Dural     : S5TIME    := S5T#5s;   //Контрольное время д/деталей
  Dura2     : S5TIME    := S5T#10s;  //Контрольное время д/перерыва
END_VAR
VAR_OUTPUT
  Load      : BOOL      := FALSE;    //Загрузка ленты деталями
  Remove    : BOOL      := FALSE;    //Удаление деталей с ленты
END_VAR
VAR
  Beltl : "Conveyor_belt";    //Управление лентой belt 1
  Belt2 : "Conveyor_belt";    //Управление лентой belt 2
  Belt3 : "Conveyor_belt";    //Управление лентой belt 3
  Belt4 : "Conveyor_belt";    //Управление лентой belt 4
  Check : "Parts_counter";    //Управление подсчетом деталей
                              //и контролем времени
END_VAR

BEGIN

NETWORK
TITLE = Initializing the common signals //инициализация общих
                                        //сигналов
  A   Man_start;
  =   Beltl.Man_on;
  =   Belt2.Man_on;
  =   BeIt3.Man_on;
  =   Belt4.Man_on;

  A   Stop;
  =   Belt1.Stop;
  =   Belt2.Stop;
  =   Belt3.Stop;
  =   Belt4.Stop;

  A   Reset;
  =   Beltl.Basic_state;
  =   Belt2.Basic_state;
  =   Belt3.Basic_state;
  =   Belt4.Basic_state;
```

(продолжение на следующей странице)

```
NETWORK
TITLE = Calling the conveyor belt controls //Вызов блоков управления
                                           //отдельными конвейерами
CALL Beltl (
   Start          := Start,
   Readyload      := Load,
   End_of_belt    := Light_barrierl,
   Mfault         := "/Mfaultl",
   Belt_mot_pn    := Belt_motl_on);
A  Belt2.Readyload;
=  Beltl.Continue;
A  BeIt1.Ready_rem;
=  Belt2.Start;

CALL Belt2 (
   End_pf_belt    := Light_barrier2,
   Mfault         := "/Mfault2",
   Belt_mot_on    := Belt_mot2_on);
A  Belt3.Readyload;
=  Belt2.Continue;
A  Belt2.Ready_rem;
=  Belt3.Start;

CALL Belt3 (
   End_of_belt    := Light__barrier3,
   Mfault         := "/Mfault3",
   Belt_mot_on    := Belt_mot3_on);
A  Belt4.Readyload;
=  Belt3.Continue;
A  Belt3.Ready_rem;
=  Belt4.Start;

CALL Belt4 (
   Continue       := Removed,
   Ready_rem      := Remove,
   End_of_belt    := Light_barrier4,
   Mfault         := "/Mfault4",
   Belt_mot_on    := Belt_mot4_on);

NETWORK
TITLE = Call for counting and monitoring //Контроль времени/режима
CALL Check (
    Set            := Start,
    Acknowledge    := "Acknowledge";
    Light_barrier  := Light barrier 1,
    Count          := #Count,
    Quantity       := #Quantity,
    Tim            := #Tim,
    Dura1          := #Dural,
    Dura2          := #Dura2,
    Finished       := Finished,
    Fault          := "Fault");

NETWORK
TITLE = Block end
   BE
END FUNCTION BLOCK
```

20 Выполнение программы

В данном разделе книги обсуждаются различные методы, применяемые для обработки программы.

Основная программа (main program) выполняется циклически. После каждого прохода программы CPU возвращается к началу программы и вновь запускает ее на выполнение. Это "стандартный" метод выполнения программ для PLC.

Многочисленные системные функции поддерживают использование системных служб, таких, как управление системными часами или связь посредством системной шины. В отличие от статических установок, выполненных при параметризации CPU, системные функции могут использоваться в динамическом режиме в процессе выполнения программы.

Основная программа может быть на время приостановлена для того, чтобы выполнить **обслуживание прерываний** (interrupt servicing). Различные типы прерываний: аппаратные прерывания (hardware interrupts), таймерные прерывания (watchdog interrupts), временные (по времени суток) прерывания (time-of-day interrupts), прерывания с задержкой обработки (time-delay interrupts), прерывания мультипроцессорного режима (multiprocessor interrupts) разбиты по приоритетным классам. Приоритеты обработки Вы можете в большей степени определять самостоятельно. Обработка прерываний позволяет пользователю оперативно реагировать на сигналы, поступающие от управляемого процесса, а также позволяет периодически выполнять процедуры контроля (мониторинга и управления) вне зависимости от времени обработки основной программы.

Перед запуском основной программы на выполнение CPU активизирует **программу запуска** (start-up program), в которой Вы можете определить параметры, относящиеся к выполнению основной программы, определить значения переменных, принимаемые по умолчанию, а также Вы можете параметризировать модули.

Обработка ошибок (error handling) является также необходимой частью обработки программы. В STEP 7 различаются синхронные ошибки (ошибки, возникающие во время обработки инструкций) и асинхронные ошибки (ошибки, которые случаются вне зависимости от выполнения программы). Для тех и других ошибок Вы можете создать программу обработки ошибок (error routine) в соответствии с Вашими требованиями.

20 Основная программа (main program)

Структура программы; управление циклом сканирования; время отклика; программные функции; многопроцессорный режим работы; обмен данными с помощью системных функций; стартовая информация (start information).

21 Обработка прерываний (interrupt handling)

Аппаратные прерывания (hardware interrupts), таймерные прерывания (watchdog interrupts), временные (по времени суток) прерывания (time-of-day interrupts), прерывания с задержкой обработки (time-delay interrupts), прерывания мультипроцессорного режима (multiprocessor interrupts), обработка прерываний.

22 Параметры запуска (start-up characteristics)

Включение питания (power-up); сброс памяти (memory reset); реманентность (retentivity); полный перезапуск (complete restart); теплый перезапуск (warm restart); установка адресов модулей; параметризация модулей.

23 Обработка ошибок (error handling)

Синхронные ошибки (программные ошибки, ошибки доступа); обработка синхронных ошибок; асинхронные ошибки; системная диагностика.

20 Основная программа (main program)

Основная программа (main program) - это циклически выполняемая (иначе говоря, "сканируемая" - "scanned") программа пользователя. "Циклическое сканирование" - это обычный метод выполнения программ в программируемых логических контроллерах. Подавляющее большинство систем управления использует только такой способ выполнения программ. Если применяется сканирование программы, управляемой событиями, то в большинстве случаев такая программа лишь дополняет основную программу.

Основная программа вызывается в организационном блоке OB 1. Основная программа имеет самый низкий приоритет, и ее выполнение может быть прервано любыми другими процедурами обработки программы. При выполнении программы переключатель режимов на передней панели CPU должен быть в положении RUN или RUN-P. Если переключатель режимов находится в положении RUN-P, то возможно программирование посредством программатора PG. Если переключатель режимов находится в положении RUN, то Вы можете удалить ключ, так что никто не сможет изменить рабочий режим без подтверждения прав авторизации. Если переключатель режимов находится в положении RUN, то программа защищена от изменений - возможно только ее считывание.

20.1 Организация программы

20.1.1 Структура программы

Проанализировать сложную задачу автоматизации - это значит разделить эту задачу на ряд мелких задач или функций в соответствии со структурой процесса, для которого необходимо построить систему управления. Затем необходимо спланировать те отдельные функциональные части, которые получены при разбиении общей задачи автоматизации процесса, чтобы определить заложенные в них функции и сигналы связи (интерфейса) с процессом и другими функциональными частями. Это разделение общей задачи управления на отдельные функциональные части может быть выполнено в Вашей программе. Таким образом, структура Вашей программы должна соответствовать структуре задачи автоматизации.

Структурированность программы пользователя может сделать программу более легко конфигурируемой. Такая программа может быть написана отдельными блоками (в том числе и несколькими программистами, в случае большой объемистости программы). И, в конце концов, не менее важный момент - структурированная пользовательская программа всегда

является более простой в пуско-наладочных работах и обслуживании. Структурированность программы зависит от ее размера и ее функций. Различаются три варианта структуры программы:

- Линейная программа
- Фрагментированная программа
- Структурированная программа

Линейная программа характерна тем, что основная программа целиком располагается в организационном блоке OB 1. Программа разбивается на сегменты (networks). STEP 7 последовательно нумерует сегменты. При редактировании или отладке программы Вы можете ссылаться на сегмент непосредственно по его номеру.

Фрагментированная программа характерна тем, что, по существу оставаясь линейной, программа разбивается на отдельные блоки. Причиной такого разбиения программы могут быть или слишком большой размер программы для размещения ее в организационном блоке OB 1, или необходимость улучшения читаемости программы. Блоки программы вызываются последовательно. Программа в отдельном блоке может быть таким же образом разбита, как и в организационном блоке OB 1. Такой способ программирования позволяет вызывать отдельные программы, связанные с отдельными функциями процесса, из разных блоков. Преимущество такой структуры программы заключается также в том, что несмотря на то, что программа остается линейной, ее можно отлаживать и выполнять отдельными фрагментами (просто добавляя или пропуская отдельные вызовы тех или иных блоков).

Структурированная программа используется, если концептуальная формулировка задачи управления особенно широка, если необходимо повторно использовать функции программы, если требуется решать сложные задачи. Структурирование программы означает разбиение программы на части (блоки), которые включают в себя независимые функции или которые имеют особое функциональное назначение и при этом обмениваются минимально возможным количеством сигналов с другими блоками.

Назначение для каждого блока программы особой функции (связанной с процессом) сделает программируемые блоки легко читаемыми и с более простым интерфейсом с другими блоками программы.

Языки программирования STL и SCL поддерживают структурное программирование посредством функций, с помощью которых Вы можете создавать блоки (автономные части программы). В главе 3 "SIMATIC S7-программа" в параграфе с названием "Блоки" обсуждаются различные виды блоков и их использование. Вы можете также найти подробное описание функций для вызова и для окончания обработки блоков (т.е. для выхода из блоков) в главе 18 "Функции блоков". Блоки получают сигналы и данные для обработки посредством "интерфейса вызова" ("call interface"), которым по сути являются параметры блоков. Блоки также формируют некоторые результаты в результате обработки данных; и эти результаты также должны быть возвращены посредством того же интерфейса для последующей обработки. Возможные способы передачи параметров подробно описаны в главе 19 "Параметры блоков". В главе 29 "SCL-блоки" содержится описание способов обработки блоков при использовании языка программирования SCL.

20.1.2 Организация программы

Способ организации программы определяет, будет ли и в каком порядке будет CPU обрабатывать блоки программы, которые Вы создали. Чтобы организовать соответствующим образом Вашу программу, необходимо запрограммировать вызовы блоков в требуемой последовательности в вызывающих блоках. Необходимо также определить порядок, в котором блоки должны вызываться, и который отображает функциональную организацию установки (процесса), для которой требуется построить систему управления.

Глубина вложения (Nesting depth)

Максимальная глубина вложения вызовов - это параметр, который зависит от приоритетного класса (касается программ в организационном блоке), а также зависит от типа CPU. В CPU 314, например, глубина вложения имеет значение восемь (8), что означает, что, начиная с организационного блока (1-й уровень вложения), Вы можете добавить еще 7 блоков "по горизонтали" (это называется вложением). Если будет последовательно (блок из блока) вызвано более 7 блоков, тогда CPU перейдет в режим STOP с индикацией ошибки переполнения стека блоков ("Block overflow"). При подсчете глубины вложения не забывайте учитывать вызовы системных функциональных блоков (SFB), а также вызовы системных функций (SFC).

Вызовы блоков данных, которые фактически открывают или выбирают области данных, не влияют на глубину вложения блоков, как не влияет на глубину вложения последовательный вызов нескольких блоков "по вертикали" (последовательный вызов блоков, скажем, в организационном блоке OB 1).

Практическая организация программы

В организационном блоке OB 1 Вы должны таким образом выстроить последовательность вызовов блоков в основной программе (main program), чтобы в первом приближении выполнялась логическая организация Вашей программы. При этом в основе своей программа может быть организована или с ориентацией на процесс ("process-related"), или с ориентацией на выполнение функций ("function-related").

Последующие моменты обсуждения этого вопроса могут дать только приблизительное, очень общее представление с целью дать начинающему пользователю некоторые идеи относительно структурирования программы и относительно перенесения его задачи управления в практическую плоскость. Продвинутые программисты обычно имеют уже достаточный опыт для того, чтобы сразу организовать структуру программы, отвечающую требованиям задачи управления.

Структура программы с ориентацией на процесс ("process-related") близка к структуре установки, для которой требуется система управления. Отдельные части программы соответствуют отдельным частям установки или управляемого процесса. В общих чертах такая структура предполагает и сканирование устройств блокировки, и использование панелей операторов, и управление приводами, и устройствами отображения (в различных частях установки). При этом для обмена

сигналами между различными частями установки используются меркеры или глобальные данные.

Структура программы с ориентацией на выполнение функций ("function-related") базируется на обеспечении функций управления. Изначально (в общей схеме) этот метод структурного программирования вообще не берет в расчет установку, для которой проектируется система управления. Структура установки начинает обнаруживать себя лишь в программных блоках более высокого (следующего) уровня разбиения (при дальнейшем разбиении структуры программы).

На практике обычно используется смешанный, гибридный подход, содержащий оба вышеуказанных метода построения логической схемы программы для системы управления. На рис. 20.1 представлен пример структуры программы. Функциональная структура отражается в программе режимов работы ("operating mode program") и в программе обработки данных ("data processing program"), которые "сверху" и "снизу" ограничивают блоки, относящиеся собственно к установке. Разделы программы Загрузка конвейера 1 (Feeding Conveyor 1), Загрузка конвейера 2 (Feeding Conveyor 2), Процесс (Process), Разгрузка конвейера (Discharge Conveyor) являются фрагментами программы с ориентацией на процесс ("process-related").

В примере на рис. 20.1 также показано использование различных типов блоков. Основная программа находится в организационном блоке OB 1. Именно в этой программе организованы вызовы блоков рабочих режимов, блоков обслуживания различных частей оборудования установки, блоков для обработки данных. Эти блоки являются функциональными блоками с экземплярными блоками для хранения данных. Разделы программы Загрузка конвейера 1 (Feeding Conveyor 1) и Загрузка конвейера 2 (Feeding Conveyor 2) имеют одинаковую структуру. Блок FB 20 с экземплярным блоком DB 20 для блока Загрузка конвейера 1 (Feeding Conveyor 1) и с экземплярным блоком DB 21 для блока Загрузка конвейера 2 (Feeding Conveyor 2) используется для управления конвейером.

В программе управления конвейером функция FC 20 обслуживает систему блокировок; функция сканирует входы или меркеры и управляет локальными данными блока FB 20.

Функциональный блок FB 101 содержит программу управления лентой конвейера. Он вызывается по одному разу для каждого конвейера. Для вызовов блоков предназначается экземплярный блок DB 20, в котором хранятся локальные данные. Этот же блок применяется для обслуживания вызова блока сбора данных FB 29.

Программа обработки данных в блоке FB 50 с экземплярным блоком DB 50 обрабатывает данные, полученные при выполненнии блока сбора данных FB 29 (и других блоков). Эти данные сохраняются в блоке глобальных данных DB 60. Функция FC 51 служит для подготовки этих данных для передачи. Передачей данных управляет блок FB 51 (с экземплярным блоком DB 51); из этого блока вызываются системные блоки SFB 8, SFB 9, SFB 62. В этом же блоке данных DB 51 системные блоки SFB сохраняют свои "экземплярные" данные.

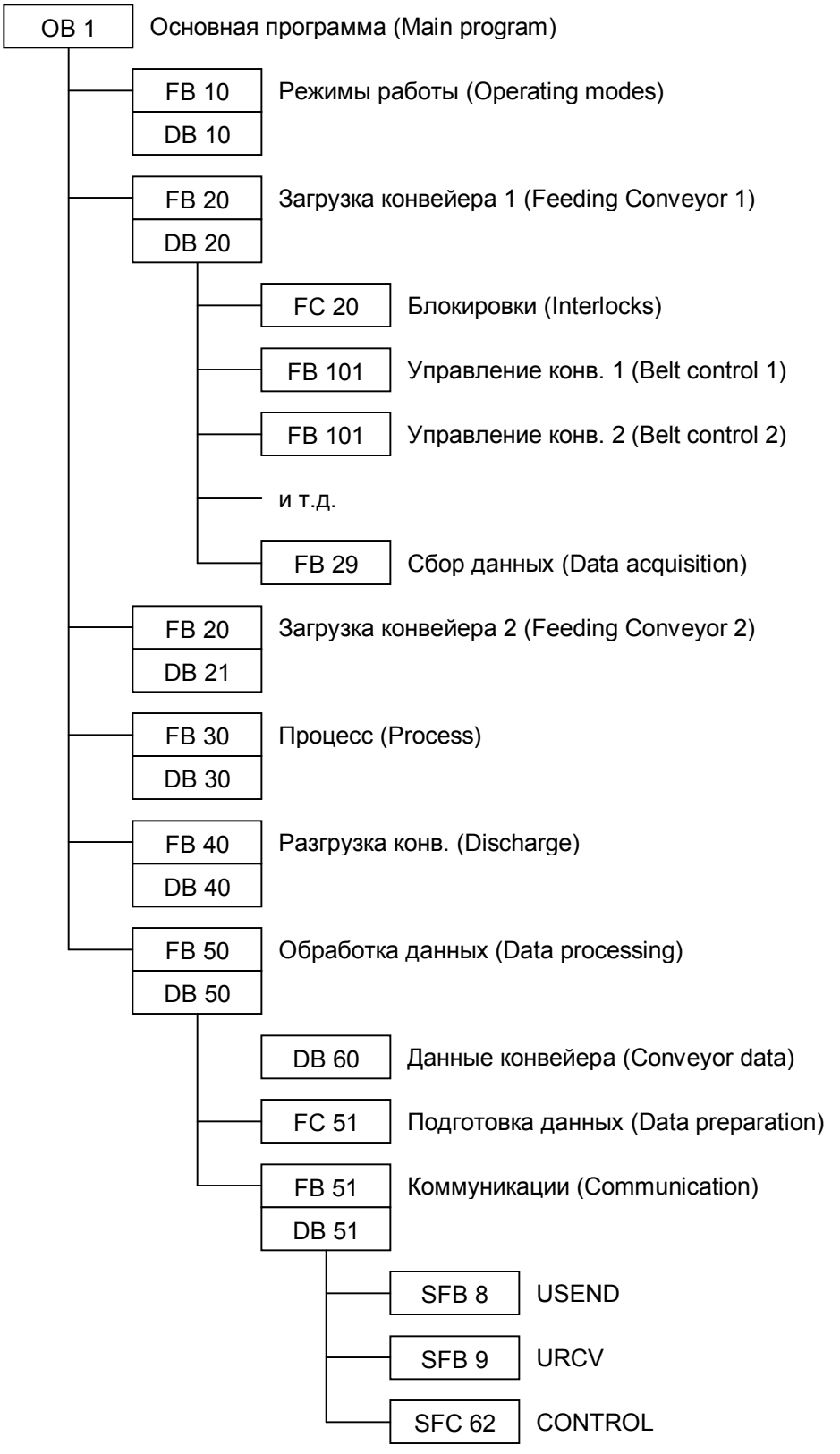


Рис. 20.1 Пример структуры программы

20. 2 Управление циклом сканирования

20.2.1 Обновление отображения состояния процесса

Область памяти для отображения состояния процесса находится во внутренней системной памяти CPU (см. раздел 1.1.4 "Области памяти CPU"). Область памяти для отображения состояния процесса начинается с I/O адреса 0 и заканчивается некоторым верхним предельным значением адресом, который зависит от типа CPU. Отдельные типы CPU позволяют пользователю самостоятельно определять верхнее значение адреса области отображения состояния процесса.

Обычно все дискретные модули отображаются в адресной области отображения состояния процесса, в то время как аналоговые модули отображаются в адресной области за пределами области отображения состояния процесса. Если CPU позволяет свободное размещение адресов, Вы можете использовать таблицу конфигурации (configuration table) для размещения любого модуля в адресной области отображения состояния процесса или за пределами области отображения состояния процесса.

Образ процесса состоит из таблицы отображения входов процесса (inputs I) и из таблицы отображения выходов процесса (outputs Q).

После перезапуска CPU перед первым выполнением организационного блока OB 1 операционная система пересылает состояния сигналов таблицы отображения выходов процесса в выходные модули и считывает сигналы входных модулей в таблицу отображения входов процесса. За этим следует выполнение организационного блока OB 1; в процессе которого обычно происходит обработка входных сигналов от входов процесса (inputs I) и вырабатываются сигналы управления (выходные сигналы) для выходов процесса (outputs Q). После этого следует завершение обработки блока OB 1 и обновление образа процесса перед началом нового цикла выполнения организационного блока OB 1 (см. рис. 20.2).

Если случается ошибка во время обновления образа процесса, например, если недоступен какой-либо модуль, то вызывается организационный блок обработки ошибок, возникающих при выполнении программы, OB 85 "Program Execution Errors". Если блок OB 85 недоступен, то CPU переходит в состояние STOP.

Отображение состояния подпроцесса (Subprocess images)

Для отдельных, специально предназначенных для этого CPU, Вы можете разбивать область отображения состояния процесса на отдельные подобласти (для отображения состояния подпроцессов) общим числом от 9 до 16. Такое разбиение области отображения состояния процесса на отдельные подобласти выполняется во время параметризации сигнальных модулей, в результате чего модули должны получить адреса. Вы можете выполнить разбиение области отображения в соответствии с разбиением на таблицы отображения входов и таблицы отображения выходов.

Все модули, для которых не будет сделано назначение в одном из адресных полей для образов подпроцессов с номерами 1 ... 8 или 1 ... 15,

будут адресоваться в адресном поле для образа подпроцесса с номером 0. Этот образ подпроцесса 0 обновляется автоматически операционной системой CPU в процессе циклического выполнения программы.

Для отдельных, специально предназначенных для этого CPU, Вы можете также назначать области отображения состояния для организационных блоков обработки прерываний, которые автоматически обновляются при вызове этих OB.

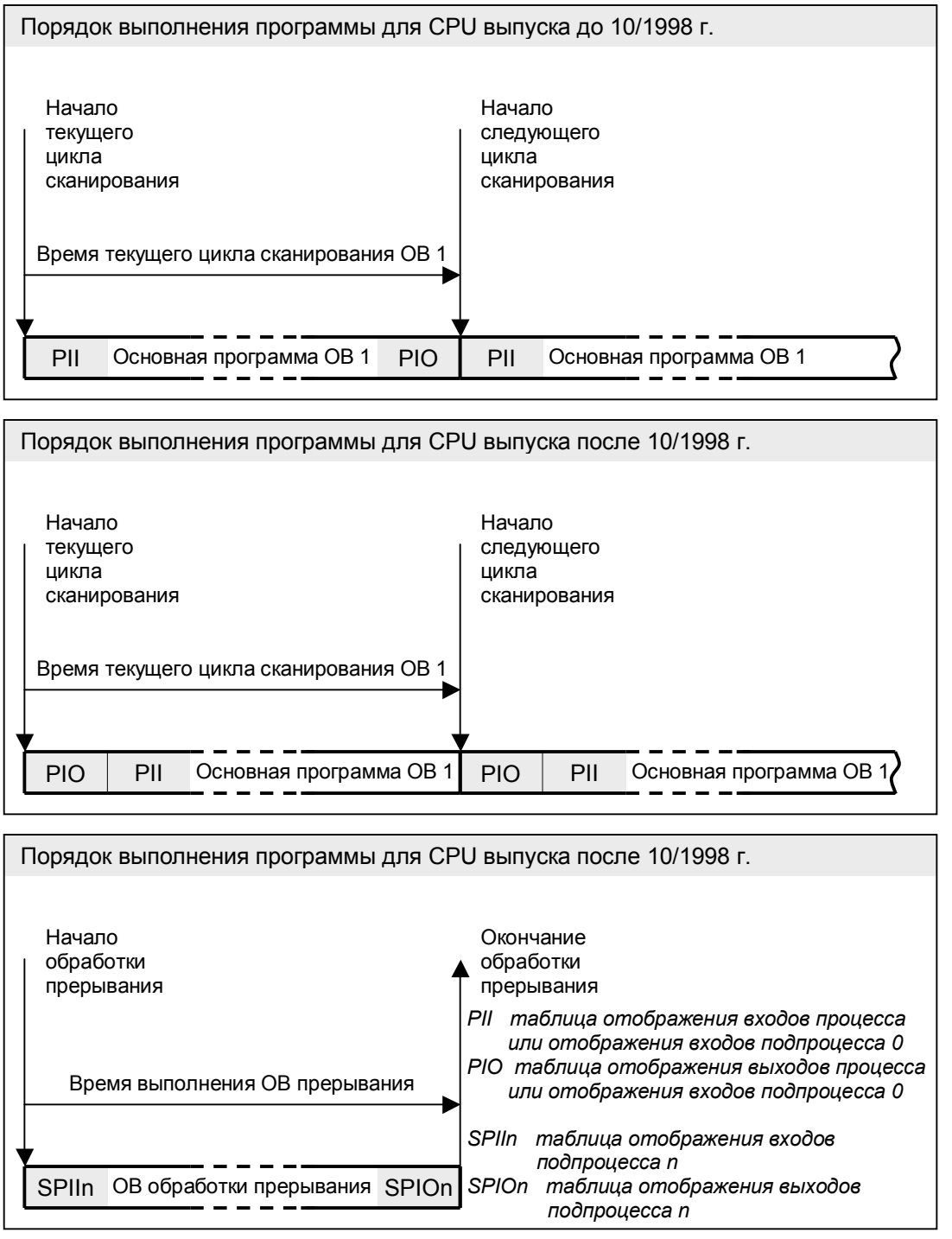


Рис. 20.2 Обновление образа процесса

SFC 26 UPDAT_PI

SFC 27 UPDAT_PO

Для обновления образа подпроцесса используются системные функции, которые вызываются из программы пользователя. Для обновления таблицы отображения входов подпроцессов используется системная функция SFC 26 UPDAT_PI, а для обновления таблицы отображения выходов подпроцессов используется системная функция SFC 27 UPDAT_PO. С помощью этих функций Вы можете также обновлять образы подпроцессов 0.

В таблице 20.1 представлены параметры указанных функций SFC.

Таблица 20.1 Параметры SFC-функций для обновления образа процесса

Имя параметра	SFC		Объявл.	Тип данных	Содержание, описание
PART	26	27	INPUT	BYTE	Номер образа подпроцесса (0 ... 15)
RET_VAL	26	27	OUTPUT	INT	Информация об ошибках
FLADDR	26	27	OUTPUT	WORD	В случае появления ошибки доступа указывается адрес первого байта

Вы можете выполнить обновление образа отдельного подпроцесса, вызывая указанные системные функции SFC в любое время и в любом месте. Например, Вы можете определить образ подпроцесса для приоритетного класса (для уровня выполнения программы), затем Вы можете вызывать обновление данного образа подпроцесса в начале и в конце выполнения соответствующего организационного блока при обработке этого приоритетного класса.

Обновление образа процесса может быть прервано вызовом более высокого приоритетного класса. Если случается ошибка во время обновления образа процесса, например, если более недоступен модуль, то об этом сообщает значение функции SFC.

20.2.2 Время мониторинга цикла сканирования

Сканирование программы в организационном блоке OB 1 отслеживается с помощью так называемого "монитора цикла сканирования" ("scan cycle monitor") или "таймера цикла сканирования" ("scan cycle watchdog"). Значение, которое принимается по умолчанию для времени мониторинга цикла сканирования, равно 150 мс. Вы можете изменять это значение в пределах от 1 мс до 6 с путем соответствующей параметризации CPU.

Если сканирование основной программы выполняется за больший промежуток времени, чем установленное время мониторинга цикла сканирования, тогда CPU вызывает OB 80 ("Timeout" - "Превышение времени"). Если организационный блок OB 80 не запрограммирован, то CPU переходит в состояние STOP.

Время мониторинга цикла сканирования соответствует полному времени

сканирования для OB 1. В это время также входит время сканирования блоков с более высоким приоритетным классом, которые вызываются в основной программе (в текущем цикле). Процессы связи, выполняемые операционной системой, например, GD-коммуникации или операции доступа программатора PG к CPU, также увеличивают время выполнения основной программы. Время выполнения основной программы может быть уменьшено в некоторой степени при параметризации CPU (на вкладке "Cycle/Clock memory bits" ["Цикл/Тактовые меркеры"] - "Cyclic load from communication" ["Циклическая загрузка посредством коммуникаций"]).

Статистика циклов

В случае наличия интерактивной (online) связи программатора PG с CPU с помощью выбора опций: *PLC -> Module Information (PLC -> Информация о модуле),* Вы можете вызвать диалоговое окно с несколькими вкладками. На вкладке "Cycle Time" ("Время цикла") будет показано текущее значение длительности времени цикла, а также длительности самого короткого и самого продолжительного циклов сканирования. На этой же вкладке отображаются также параметризованная минимальная длительность цикла и время мониторинга цикла сканирования ("scan cycle monitoring time").

Кроме того, продолжительность последнего цикла сканирования также как и продолжительности минимального и максимального по длительности циклов с момента последнего запуска PLC, могут быть считаны из временных локальных данных в стартовой информации организационного блока OB 1.

SFC 43 RE_TRIGR

Перезапуск времени мониторинга цикла сканирования

Вызов системной функции SFC 43 RE_TRIGR перезапускает время мониторинга цикла сканирования ("scan cycle monitoring time"); таймер перезапускается с новым значением, установленным посредством параметризации CPU. Функция SFC 43 не имеет параметров.

Продолжительность процедур операционной системы (Operating system run times)

Во время мониторинга цикла сканирования также включается продолжительность процедур операционной системы. К этим процедурам относятся:

- Системное управление циклическим сканированием ("no-load cycle" - "незагружаемый цикл"); фиксированное значение;
- Обновление образа процесса; зависит от числа байт, которые необходимо обновлять;
- Обновление таймеров; зависит от числа таймеров, которые необходимо обновлять;
- Загрузка коммуникаций.

Функции связи (коммуникации) для CPU включают в себя функции передачи блоков программы пользователя и обмен данными между модулями CPU с использованием системных функций. Время, которое CPU будет затрачивать на выполнение этих функций, может быть

ограничено при параметризации CPU.

Все значения продолжительностей процедур операционной системы зависят от типа CPU и являются его характеристиками.

20.2.3 Минимальное время цикла сканирования Сканирование в фоновом режиме ("background scanning")

Для отдельных, специальным образом оборудованных, CPU Вы можете задавать минимальное время цикла сканирования ("minimum scan cycle time"). В случае если продолжительность выполнения основной программы (включая обработку прерываний) меньше минимального времени цикла сканирования, CPU будет ожидать, пока не истечет заданное минимальное время, и только после этого начнется следующий цикл с повторного вызова OB 1.

Значение, принимаемое по умолчанию для минимального времени цикла сканирования, составляет 0 мс, иначе говоря, данная функция неактивна (disabled). Для параметра "минимальное время цикла сканирования" ("minimum scan cycle time") Вы можете установить значение в пределах от 1 миллисекунд до 6 секунд на вкладке "Cycle/Clock memory bits" ("Цикл/Тактовые меркеры") при параметризации CPU.

Сканирование в фоновом режиме ("background scanning") OB 90

В интервале между фактическим окончанием цикла сканирования и моментом истечения времени минимальной продолжительности цикла сканирования CPU обрабатывает организационный блок OB 90 "Background scanning" ("Сканирование в фоновом режиме") (см. рис. 20.3). Организационный блок OB 90 выполняется "по частям". Когда операционная система вызывает блок OB 1, выполнение блока OB 90 прерывается; при этом, когда обработка блока OB 1 закончится, выполнение блока OB 90 продолжится в точке прерывания.

Выполнение организационного блока OB 90 может быть прервано после каждого оператора; тем не менее, любой системный блок, вызванный в блоке OB 90, перед обработкой прерывания будет выполнен полностью.

Размер отдельных выполняемых фрагментов блока OB 90 зависит от продолжительности текущего цикла сканирования блока OB 1. Чем меньше разница между продолжительностью текущего цикла сканирования и минимальной продолжительностью цикла сканирования, тем меньше времени остается для выполнения OB 90. Мониторинг времени сканирования программы в OB 90 не проводится.

Блок OB 90 выполняется (сканируется) только в режиме RUN. Этот процесс сканирования может быть прерван, если возникает событие прерывания или ошибки, так же как прерывается обработка блока OB 1.

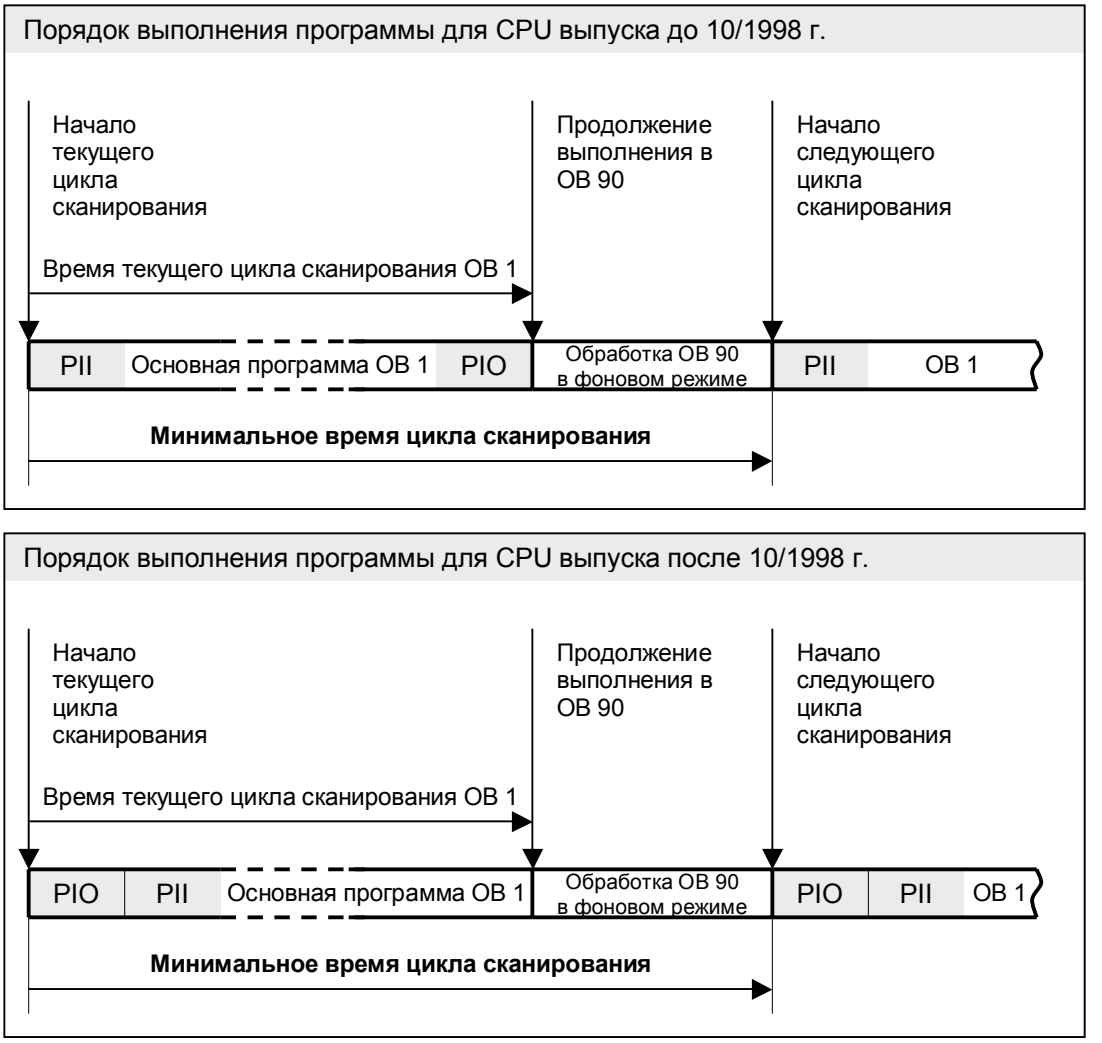


Рис. 20.3 Минимальная длительность цикла и сканирование в фоновом режиме

В стартовой информации (во временных локальных данных) организационного блока (байт 1), указывается причина запуска обработки блока OB 90:

- B#16#91
  После перезапуска CPU
- B#16#92
  После того, как блок, обрабатывавшийся в OB 90, был удален или заменен.
- B#16#93
  После (пере)загрузки OB 90 в режиме RUN.
- B#16#95
  После того, как программа в OB 90 была запущена на выполнение и начался новый цикл ее обработки в фоновом режиме.

20.2.4 Время отклика ("Response Time")

Если программа пользователя в OB 1 работает со значениями (с состояниями сигналов) из области отображения процесса, это сказывается на таком параметре, как время отклика ("response time"), который зависит от времени выполнения программы (от времени сканирования цикла). Величина времени отклика имеет значение, лежащее между длительностью одного и двух циклов сканирования, это показано в следующем примере.

Если, например, "концевой выключатель" ("limit switch") активирован, то состояние его сигнала изменяется со значения "0" на значение "1". Программируемый контроллер обнаруживает это изменение сигнала во время последующего обновления образа процесса и устанавливает входы (inputs), соответствующие этому концевому выключателю в состояние "1". В программе происходит проверка состояния сигнала от концевого выключателя, и при обнаружении состояния "1" происходит сброс некоторого выхода (output) для выключения соответствующего двигателя. Новое состояние сигнала этого выхода, который был сброшен, передается в конце цикла сканирования программы; только после этого соответствующий бит в выходном дискретном модуле будет сброшен.

Наилучший случай имеет место, когда образ процесса обновляется сразу же после изменения состояния сигнала, поступающего от концевого выключателя. При этом проходит период времени, равный только одному циклу сканирования, от момента изменения сигнала выключателя до момента изменения состояния соответствующего выхода (см. рис. 20.4).

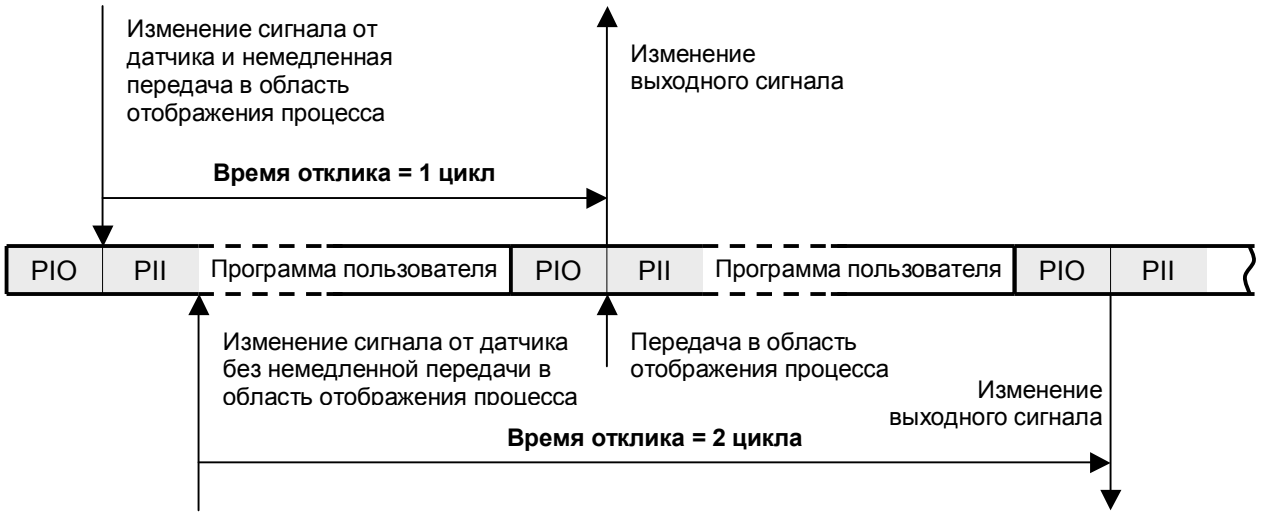


Рис. 20.4 Время отклика программируемого контроллера

Наихудший случай имеет место, когда изменение состояния сигнала, поступающего от концевого выключателя, происходит сразу же после того, как произошло обновление образа процесса. При этом проходит период времени, приблизительно равный одному циклу сканирования, до момента, когда PLC обнаруживает изменение сигнала от выключателя

и устанавливает соответствующий ему вход (input). После этого должен пройти еще один период времени, равный одному циклу сканирования, прежде чем будет изменено состояние соответствующего выхода.

Время, необходимое для выполнения программы пользователя, включает в себя время выполнения всех процедур выполняемых в одном программном цикле (включая, например, обслуживание прерываний, выполнение операционной системой таких функций, как обновление таймеров, управление MPI-интерфейсом, обновление образа процесса).

Величина времени отклика на изменение входного сигнала, таким образом, может иметь значение, лежащее между длительностью одного и двух циклов сканирования. Величина времени отклика PLC увеличивается также за счет времени задержки во входных модулях, времени переключения контакторов и т.п.

В некоторых случаях Вы можете уменьшить время отклика посредством прямой адресации I/O периферии или с помощью организации вызовов разделов программы на основе управления событиями.

20.2.5 Стартовая информация ("Start Information")

Операционная система CPU первоначально обращается к стартовой информации в организационном блоке OB 1, как и при обработке любого другого организационного блока. Стартовая информация ("start information") занимает первые 20 байт во временных локальных данных. Вы можете самостоятельно создать объявление для стартовой информации или использовать для этого информацию из стандартной библиотеки *Standard Library* в разделе для организационных блоков *Organization Blocks*. В таблице 20.2 представлены стартовая информация для организационного блока OB 1, назначение символьных имен, принимаемых по умолчанию, и типы данных. Вы можете изменять назначения в любое время и можете, также, выбрать более приемлемые для Вас имена. Даже если Вы не используете стартовую информацию, Вы должны зарезервировать первые 20 байтов во временных локальных данных для стартовой информации (например, в форме массива размером 20 байтов).

В SIMATIC S7 все сообщения о событиях имеют фиксированную структуру, которая определяется классом события. В стартовой информации блока OB 1, например, сообщение о событии B#16#11 стандартный вызов OB. Из содержания следующего байта Вы можете узнать, находится ли основная программа в первом цикле после включения, а также были ли после этого вызваны, например, программы инициализации в циклической программе.

Приоритет и номер OB основной программы имеют фиксированные значения. В стартовой информации содержатся три числа в формате INT, показывающие величину предыдущего цикла сканирования, а также величины максимального и минимального циклов с момента последнего включения контроллера. Последнее значение в формате DATE_AND_TIME показывает, когда программа управления обнаружила событие для вызова организационного блока OB 1.

Таблица 20.2 Стартовая информация для организационного блока OB 1

Имя	Тип данных	Описание	Содержание
OB1_EV_CLASS	BYTE	Класс события	B#16#11
OB1_SCAN_1	BYTE	Стартовая информация	B#16#01 = 1-й цикл после полного перезапуска B#16#02 = 1-й цикл после теплого перезапуска B#16#02 = каждый другой цикл
OB1_PRIORITY	BYTE	Приоритет	B#16#01
OB1_OB_NUMBR	BYTE	Номер OB	B#16#01
OB1_RESERVED_1	BYTE	Зарезервировано	-
OB1_RESERVED_2	BYTE	Зарезервировано	-
OB1_PREV_CYCLE	INT	Время предыдущего цикла	в мс
OB1_MIN_CYCLE	INT	Время минимального цикла	в мс
OB1_MAX_CYCLE	INT	Время максимального цикла	в мс
OB1_DATE_TIME	DT	Время прихода события для вызова OB 1	Время вызова OB (в цикле)

Надо отметить, что прямое считывание стартовой информации организационного блока возможно выполнять только в этом организационном блоке, так как эта информация содержится во временных локальных данных. Если Вам требуется стартовая информация, которая находится в более "глубоких" уровнях вложения вызовов, то Вы должны использовать вызов системной функции SFC RD_SINFO в соответствующем месте программы.

SFC 6 RD_SINFO

Считывание стартовой информации

Системная функция SFC 6 RD_SINFO обеспечивает считывание стартовой информации в текущем организационном блоке (то есть, в OB, находящемся на вершине "дерева вызовов") и в последнем выполненном организационном блоке запуска даже на более глубоком уровне вызовов (см. табл. 20.3).

Выходной параметр TOP_SI содержит первые 12 байтов стартовой информации текущего организационного блока OB, выходной параметр START_UP_SI содержит первые 12 байтов стартовой информации последнего выполненного стартового блока OB. Однако в обоих случаях в этой информации нет отметок времени.

Системная функция SFC 6 RD_SINFO может быть вызвана не только в любом месте основной программы, но и в каждом приоритетном классе, даже в организационном блоке обработки ошибок или в программе запуска. Если системная функция SFC 6 RD_SINFO вызывается, например, в организационном блоке обработки прерывания, то параметр TOP_SI содержит стартовую информацию организационного блока обработки прерывания. В случае вызова SFC при перезапуске параметры TOP_SI и START_UP_SI содержат одинаковую информацию.

Таблица 20.3 Параметры для системной функции SFC 6 RD_SINFO

SFC	Имя параметра	Объявление	Тип данных	Содержание, описание
6	RET_VAL	OUTPUT	INT	Информация об ошибках
	TOP_SI	OUTPUT	STRUCT	Стартовая информация для текущего OB (с такой же структурой как у параметра START_UP_SI)
	START_UP_SI	OUTPUT	STRUCT	Стартовая информация для последнего выполненного стартового блока OB:
	.EV_CLASS		BYTE	ID и класс события
	.EV_NUM		BYTE	номер события
	.PRIORITY		BYTE	приоритет выполнения (номер уровня выполнения)
	.NUM		BYTE	номер блока OB
	.TYP2_3		BYTE	ID дополнительной информации 2_3
	.TYP1		BYTE	ID дополнительной информации 1
	.ZI1		WORD	дополнительная информация 1
	.ZI2_3		DWORD	дополнительная информация 2_3

20.3 Функции программы (Program Functions)

Кроме параметризации CPU с помощью утилиты конфигурирования оборудования Hardware Configuration Вы можете также выбирать номер системной функции динамически в процессе выполнения программы посредством встроенных системных функций.

20.3.1 Управление часами реального времени (Real-Time Clock)

Следующие системные функции могут использоваться для управления часами реального времени CPU (Real-Time Clock):

- SFC 0 SET_CLK
  Функция позволяет установить дату и время.
- SFC 1 READ_CLK
  Функция позволяет считать дату и время.
- SFC 48 SNC_RTCB
  Функция позволяет синхронизировать часы CPU.

Вы можете найти список параметров этих SFB в таблице 20.4.

Таблица 20.4 Параметры системных функций для управления часами реального времени CPU (Real-Time Clock)

SFC	Имя параметра	Объявление	Тип данных	Содержание, описание
0	PDT	INPUT	DT	Дата и время (новые значения)
	RET_VAL	OUTPUT	INT	Информация об ошибках
1	RET_VAL	OUTPUT	INT	Информация об ошибках
	CDT	OUTPUT	DT	Дата и время (текущие значения)
48	RET_VAL	OUTPUT	INT	Информация об ошибках

Если несколько CPU связаны между собой посредством подсети, "часы" одного из них должны быть инициализированы как "master clock" ("главные часы"). При параметризации CPU также необходимо задать интервал синхронизации, по прошествии которого все часы в подсети должны автоматически синхронизироваться с "master clock" ("главными часами").

При вызове функции SFC 48 SNC_RTCB в CPU с назначенными "master clock" ("главными часами") происходит синхронизация всех часов в подсети, независимо от автоматической синхронизации. При установке времени в "master clock" (в главных часах) посредством системной функции SFC 0 SET_CLK происходит автоматическая синхронизация всех часов в подсети - все часы будут установлены на заданное время.

20.3.2 Системные часы (System Clock)

Системные часы CPU (system clock) запускаются при включении питания или при полном перезапуске. Системные часы (system clock) CPU идут (включены) пока CPU выполняет программу перезапуска или находится в режиме RUN. Когда CPU переходит в режим STOP или HOLD, текущее "системное время" (system time) "замораживается".

Если Вы активируете "теплый" ("warm") перезапуск в системе с S7-400 CPU, системные часы вновь начинают отсчет с ранее запомненного "замороженного" значения времени.

При "холодном" перезапуске или полном перезапуске происходит сброс системного времени.

Системное время (system time) имеет формат данных TIME, поэтому его величина может принимать только положительные значения в диапазоне от TIME#0ms до TIME#24d20h31m23s647ms.

В случае переполнения значения системного времени, оно вновь начинает отсчитываться с 0. В CPU 3xx (кроме CPU 318) обновление системного времени происходит каждые 10 миллисекунд, а в CPU 4xx, а также в CPU 318 обновление системного времени происходит каждую миллисекунду.

SFC 64 TIME_TCK

Считывание системного времени

Системная функция SFC 64 TIME_TCK обеспечивает считывание текущего системного времени. Параметр RET_VAL содержит значение системного времени в формате TIME.

Вы можете использовать системные часы, например, для считывания текущего времени выполнения программы CPU или, вычисляя разность их показаний в двух точках, рассчитать величину промежутка времени между двумя вызовами функции SFC 64. Разность между двумя значениями в TIME-формате вычисляется с использованием вычисления для DINT-чисел.

20.3.3 Измеритель времени наработки (Run-Time Meter)

Измеритель времени наработки (run-time meter) в CPU отсчитывает наработанные аппаратурой часы. Вы можете определять наработанное время CPU или подключенных к нему устройств. Возможное число измерителей наработки для каждого CPU определяется его типом. Если CPU переходит в режим STOP или HOLD, измеритель времени наработки также останавливается; когда CPU вновь запускается, измеритель времени наработки продолжает отсчет времени с запомненного в момент остановки значения.

Когда показания измерителя времени наработки достигают значения 32767 часов, измеритель времени наработки останавливается и сигнализирует о переполнении значения. Измеритель времени наработки может быть установлен на новое значение или сброшен в нулевое состояние только при вызове соответствующей системной функции SFC.

Следующие системные функции используются для управления измерителем времени наработки (run-time meter):

- SFC 2 SET_RTM
  Функция позволяет установить измеритель времени наработки
- SFC 3 CTRL_RTM
  Функция позволяет остановить или запустить измеритель времени наработки
- SFC 4 READ_RTM
  Функция позволяет считать показания измерителя времени наработки

В таблице 20.5 представлен список параметров указанных системных функций. Параметр NR устанавливает номер измерителя времени наработки для CPU; он имеет тип BYTE. Этот параметр может быть инициализирован с помощью константы или переменной (аналогично тому, как инициализируются все входные параметры простых типов данных). Параметр PV (тип данных INT) используется для установки измерителя времени наработки на начальное значение. Параметр S функции SFC 3 при значении "1" инициирует запуск выбранного измерителя времени наработки, а при значении "0" инициирует его остановку. Параметр CV индицирует состояние измерителя времени наработки при сканировании программы: значение "1" - активен; значение

"0" - остановлен. Параметр CV содержит текущее значение измерителя времени наработки в формате INT.

Таблица 20.5 Параметры системных функций для управления измерителем времени наработки (Run-Time Meter)

SFC	Параметр	Объявление	Тип данных	Содержание, описание
2	NR	INPUT	BYTE	Номер измерителя времени наработки (B#16#01 ... B#16#08)
	PV	INPUT	INT	Новое значение измерителя времени наработки
	RET_VAL	OUTPUT	INT	Информация об ошибках
3	NR	INPUT	BYTE	Номер измерителя времени наработки (B#16#01 ... B#16#08)
	S	INPUT	BOOL	При значении "1" происходит запуск измерителя времени наработки; при значении "0" происходит его остановка
	RET_VAL	OUTPUT	INT	Информация об ошибках
4	NR	INPUT	BYTE	Номер измерителя времени наработки (B#16#01 ... B#16#08)
	RET_VAL	OUTPUT	INT	Информация об ошибках
	CQ	OUTPUT	BOOL	Параметр-индикатор при сканировании: значение "1" - измеритель времени наработки активен; значение "0" - измеритель времени наработки остановлен.
	CV	OUTPUT	INT	Текущее значение измерителя времени наработки

20.3.4 Сжатие информации в памяти CPU (Compressing CPU Memory)

Многократное удаление и повторная загрузка блоков (что часто происходит во время интерактивного изменения блоков) могут приводить к появлению незанятых участков в рабочей (work) памяти CPU и в загрузочной (load) RAM-памяти, что в свою очередь, уменьшает количество полезного пространства памяти. Для устранения таких незанятых участков в памяти служит функция "Compress" ("Сжатие"), при запуске которой происходит заполнение незанятых участков путем сдвига блоков друг к другу. Вы можете инициировать функцию "Compress" ("Сжатие") с помощью программатора PG, подключенного к CPU, или с помощью вызова системной функции SFC 25 COMPRESS. Параметры для функции SFC 25 представлены в таблице 20.6.

Процедура сжатия распределяется на несколько программных циклов. Функция SFC 25 в параметре BUSY возвращает значение "1", если процедура еще не закончена, а в параметре DONE возвращает значение "1", если процедура сжатия полностью завершена.

Таблица 20.6 Параметры системной функции SFC 25 COMPRESS

SFC	Параметр	Объявление	Тип данных	Содержание, описание
25	RET_VAL	OUTPUT	INT	Информация об ошибках
	BUSY	OUTPUT	BOOL	Значение "1", если процедура сжатия еще не закончена
	DONE	OUTPUT	BOOL	Значение "1", если процедура сжатия полностью выполнена

Системная функция SFC 25 COMPRESS не может быть активизирована, если запущен процесс сжатия извне, если активна функция удаления блоков "Delete Block" или если функции из PG получили доступ к блоку, который должен быть сдвинут (например, функция статуса блока "Status Block").

Необходимо отметить, что отдельные блоки максимальной длины (определяется типом CPU) не могут быть сдвинуты, т.е. незанятые участки в памяти CPU остаются. Только функция "Compress" ("Сжатие"), запущенная из программатора PG, в то время, пока CPU находится в режиме STOP, может полностью устранить незанятые участки в памяти CPU.

20.3.5 Режимы ожидания и остановки

Системная функция SFC 47 WAIT обеспечивает остановку сканирования программы на заданный период времени.

Функция SFC 47 WAIT имеет входной параметр WT (тип INT), с помощью которого Вы можете определить время ожидания (waiting time) в микросекундах (μs). Максимальная величина этого параметра составляет 32767 микросекунд; минимальная величина времени ожидания равна времени выполнения системной функции, которое определяется типом CPU.

Системная функция SFC 47 WAIT может быть прервана событием с более высоким приоритетом. В случае использования S7-300 это увеличивает время ожидания на величину времени сканирования программы обработки прерывания с более высоким приоритетом.

Системная функция SFC 48 STP обеспечивает завершение сканирования программы, в результате чего CPU переходит в состояние STOP.

Системная функция SFC 48 STP не имеет параметров.

20.3.6 Мультипроцессорный режим

Станция S7-400 может использоваться в мультипроцессорном режиме. В одной стойке при этом могут работать до четырех допускающих такой

режим процессоров.

Станция S7-400 автоматически запускается в мультипроцессорном режиме после того, как в центральной стойке Вы установили посредством утилиты конфигурирования аппаратной части Hardware Configuration больше, чем один CPU. При этом слоты для установки CPU произвольны. CPU различаются по автоматически назначаемым номерам, идущим в порядке возрастания в соответствии с местом (слотом) установки. Пользователь также может назначить номер CPU самостоятельно используя вкладку "Multicomputing" ("Мультипроцес-сорный режим").

Данные конфигурирования для всех CPU должны быть загружены в PLC, даже если Вы делаете изменения в конфигурации только одного CPU. После назначения параметров для CPU Вы должны назначить каждый модуль станции соответствующему CPU.

Это делается с помощью параметризации модуля на вкладке "Address" ("Адреса") в окне "CPU assignment" ("Назначение CPU") (см. рис. 20.5). Одновременно с назначением адресной области модуля Вы должны назначить прерывания модуля для этого CPU. С помощью опций: *View -> Filter -> CPU No.x-Modules (Вид -> Фильтр -> CPU No.x)* Вы можете выделить модули, назначенные для CPU в таблицах конфигурации.

Все CPU в мультипроцессорной сети имеют одинаковый режим работы. Это означает, что:

- все они должны быть параметризированы с одинаковым режимом перезапуска;
- все они должны переходить в режим RUN одновременно;
- все они должны быть переведены в режим HOLD, когда Вы делаете отладку на одном из CPU;
- все они должны переходить в режим STOP, как только один из CPU переходит в режим STOP.

Как только происходит сбой в одной стойке в станции, в каждом CPU должен вызываться организационный блок OB 86. Программы пользователя в таких CPU выполняются независимо одна от другой; они не синхронизированы.

Вызов системной функции SFC 35 MP_ALM запускает организационный блок OB 60 "Multiprocessor interrupt" ("Обработка прерывания мультипроцессорного режима ") на всех CPU одновременно (см. раздел 21.6 "Прерывания мультипроцессорного режима").

20.4 Связь (communications) посредством распределенной периферии I/O

Таким же путем, каким модули с централизованной компоновкой назначаются CPU, распределенные модули (станции, ведомые (slave) DP-устройства) назначаются ведущему (master) DP-устройству. Ведущее (master) DP-устройство вместе с относящимися к нему ведомыми (slave) DP-устройствами образуют "систему ведущего (master) DP-устройства". Одна S7-станция может содержать несколько систем ведущих (master) DP-устройств.

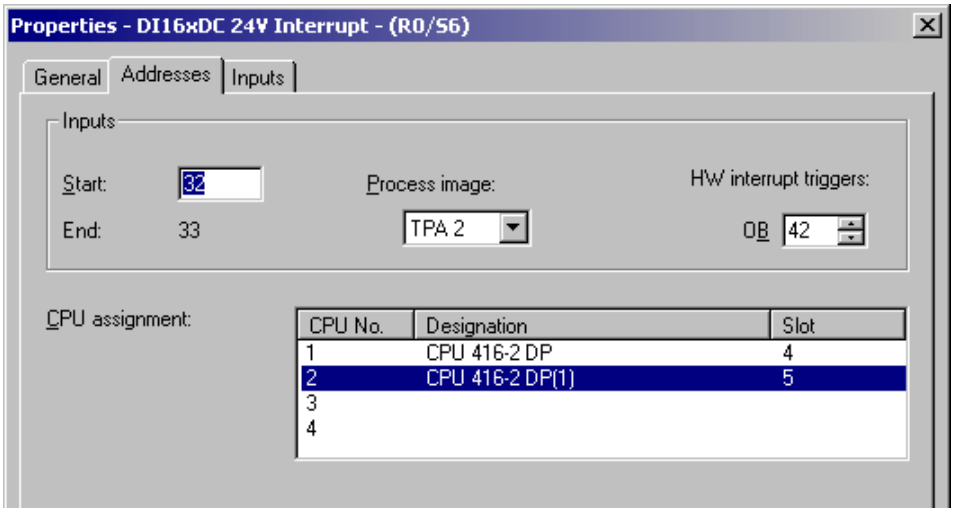


Рис. 20.5 Назначение модулей

Также как и центральные модули, ведомые (slave) DP-устройства получают адреса в I/O области CPU (пространство логических адресов). Ведущее (master) DP-устройство, так сказать, "прозрачно" в отношении адресов его ведомых (slave) DP-устройств: CPU "видит" адреса ведомых (slave) DP-устройств, поэтому эти адреса не должны перекрываться с адресами центральных модулей. Адреса ведомых (slave) DP-устройств также не должны перекрываться с адресами ведомых (slave) DP-устройств из других систем ведущих (master) DP-устройств, назначенных CPU.

Существуют ведомые (slave) DP-устройства, которые ведут себя аналогично центральным модулям, т.е., они содержат области данных пользователя и области системных данных, они могут инициировать процесс и вызывать диагностические прерывания, в случае, если оснащены соответствующим образом. Такие станции относятся к "DP-S7 slaves" (к "ведомым DP-S7 устройствам"). "DP V0 standard slaves" ("ведомые DP-устройства стандарта V0") совместимы со стандартом EN 50170, том 2, PROFIBUS. Существенные различия между ними состоят в способе считывания и в структуре диагностических данных.

Конфигурирование распределенной периферии (I/O) также очень похоже на конфигурирование центральных модулей. Исходным пунктом является ведущее (master) DP-устройство в графическом представлении системы ведущего DP-устройства. Станции устанавливаются в систему, затем получают адреса и параметризируются.

Особые требования к консистентности пользовательских данных требуют специальных системных функций для распределенной периферии (I/O). Так что, несмотря на последовательный характер передачи данных для ведомых (slave) DP-устройств, консистентность данных за пределами 4 байтов может гарантироваться системой S7, и Вы можете группировать ведомые (slave) DP-устройства таким образом, что они могут принимать и передавать данные синхронно.

20.4.1 Адресация распределенной периферии (I/O)

Каждое ведомое (slave) DP-устройство может адресоваться в соответствии с

- адресом узла (node address),
- географическим адресом (geographical address),
- начальным адресом модуля (module starting address),
- диагностическим адресом (diagnostics address).

Адресация в системе ведущего (master) DP-устройства показаны на рис. 20.6).

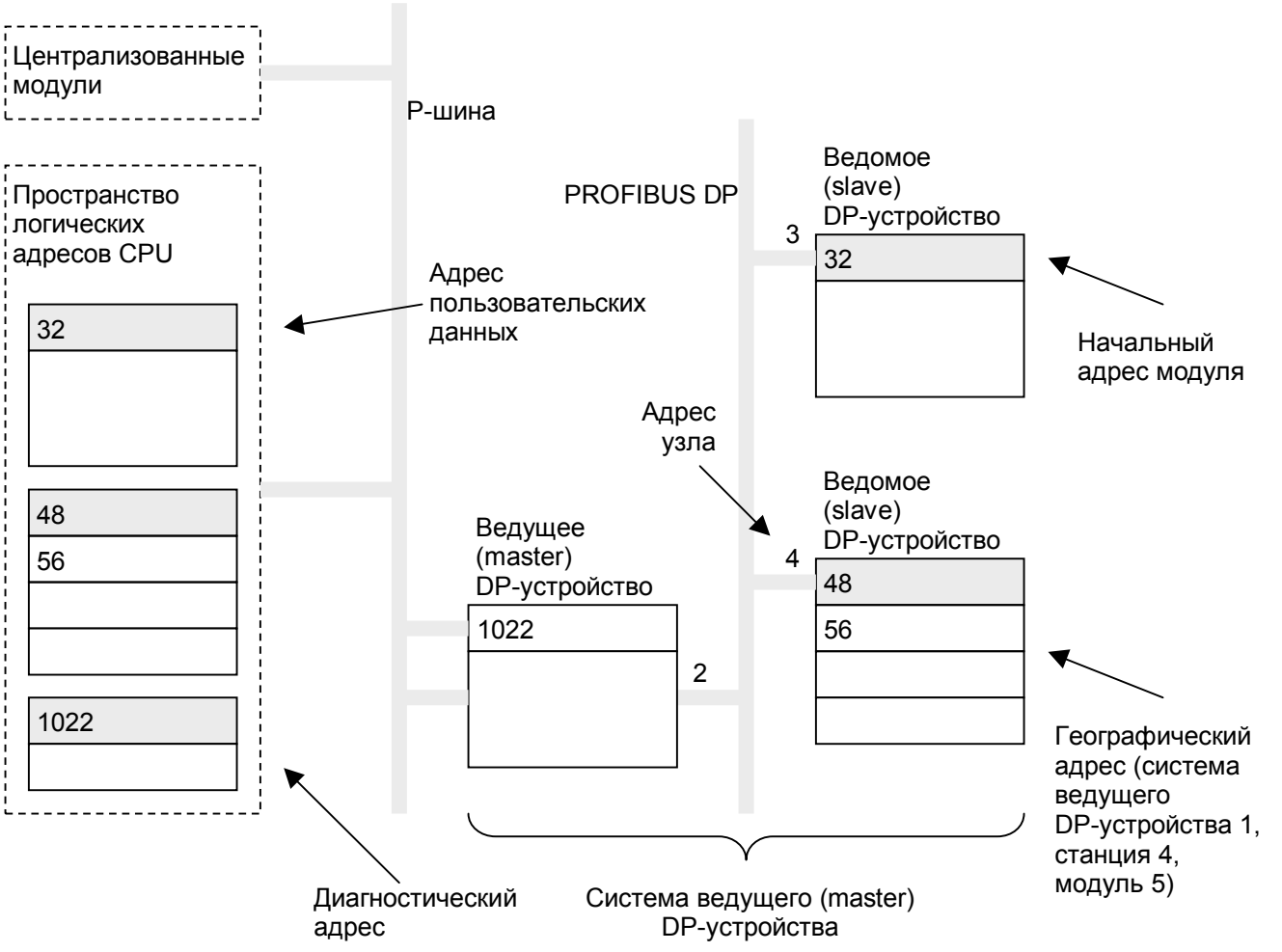


Рис. 20.6 Адреса в системе ведущего (master) DP-устройства

Адрес узла (Node Address)

Каждый узел в подсети PROFIBUS имеет уникальный адрес, именуемый адресом узла (node address) или номером станции (station number), по которому система отличает его от других узлов подсети. Доступ к станции в подсети PROFIBUS осуществляется с помощью такого адреса узла.

Необходимо отметить, что между адресами активных узлов шины должен быть разрыв, равный, по крайней мере, единице (1), (например, в случае перекрестного обмена данными между ведущим (master) DP-устройством и узлами). Система S7 учитывает это требование при автоматическом назначении адресов узлам подсети.

Географический адрес (Geographical Address)

Географический адрес (geographical address) ведомого (slave) DP-устройства соответствует адресу слота центрального модуля. Этот адрес состоит из идентификатора (ID) системы ведущего (master) DP-устройства, определяемого во время конфигурирования, и адреса узла в подсети PROFIBUS (соответствующего номеру стойки).

В случае модульной организации ведомых (slave) DP-устройств номер слота также добавляется к адресу, а если модули сами составлены из модулей (подмодулей), номер слота подмодуля также должен добавляться к адресу (что касается станций S7-300, то нумерация слотов начинается с 4).

Начальный адрес модуля (Module Starting Address)
Консистентность данных

По начальному адресу модуля (module starting address) ("logic base address" - "базовый логический адрес") Вы можете получить доступ к пользовательским данным компактного ведомого (slave) DP-устройства (compact DP slave) или к модулю модульного ведомого DP-устройства (modular DP slave). Как следует из названия, этот адрес соответствует начальному адресу модуля для центрального модуля. Если адреса ведомого (slave) DP-устройства обеспечивают консистентность данных для 1, 2 или 4 байтов, Вы можете использовать для доступа к пользовательским данным операторы загрузки (load) и пересылки (transfer). Если начальный адрес модуля лежит в области отображения процесса, то пользовательские данные пересылаются во время передачи данных образа процесса. Таким же образом возможно назначение данных в область отображения подпроцесса.

Если должна обеспечиваться консистентность данных длиной 3 или больше, чем 4 байтов (до некоторого максимального размера, определяемого типом CPU), то Вам потребуется использовать системные функции SFC 14 DPRD_DAT и SFC 15 DPWR_DAT. Эти функции предназначены для считывания и записи пользовательских данных соответственно, с сохранением их консистентности.

Функции SFC в качестве исходной области данных или области назначения данных используют адреса указанные в параметре RECORD. Рис. 20.7 иллюстрирует передачу пользовательских данных.

Ведущее (master) DP-устройство пересылает пользовательские данные из "его" ведомых (slave) DP-устройств циклически; в примере оно пересылает из станции с начальным адресом модуля 32 и с гарантией консистентности данных для 4 байтов (ведомое DP-устройство 1) и из

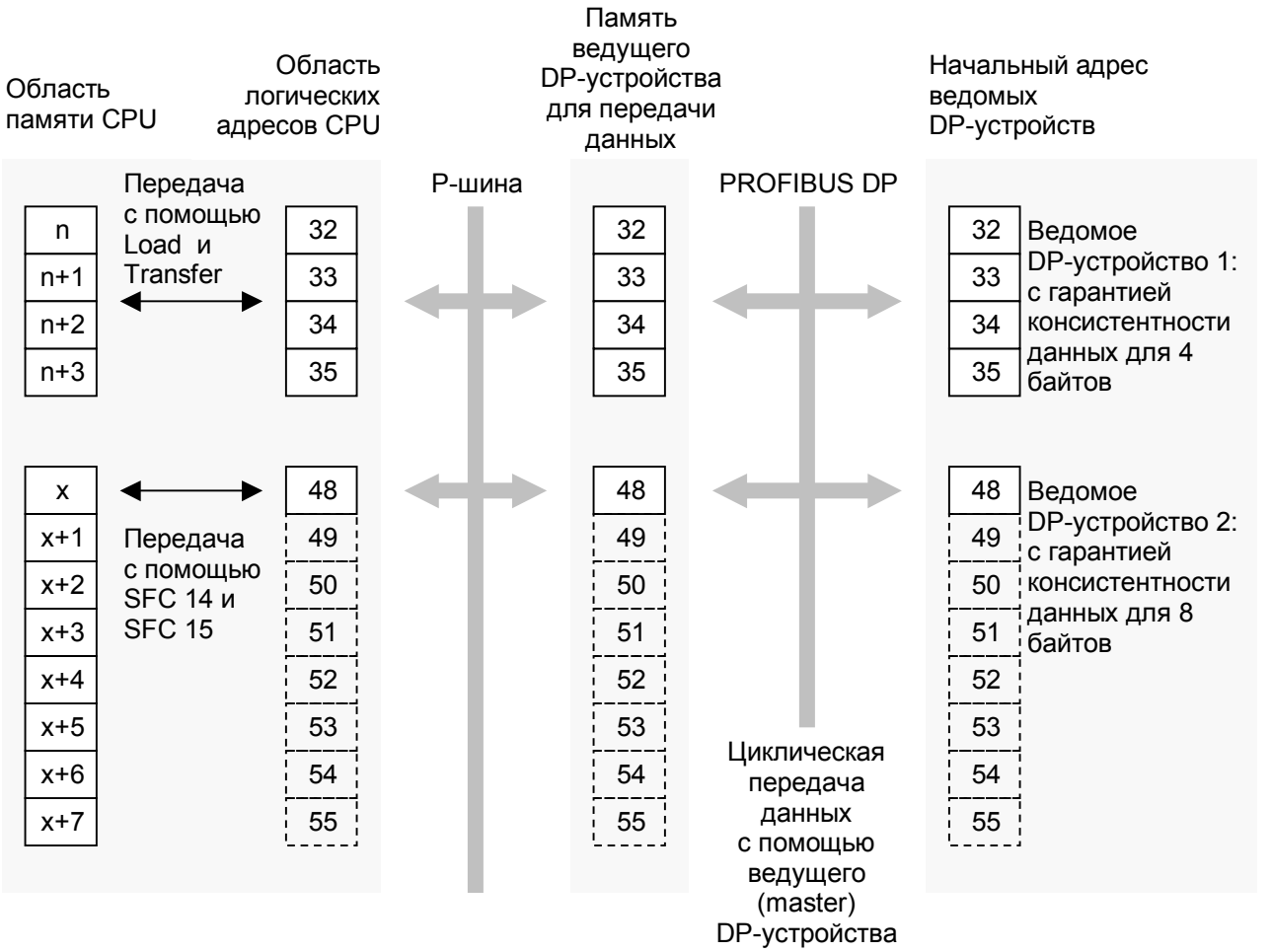


Рис. 20.7 Передача пользовательских данных в распределенной периферии.

станции с начальным адресом модуля 48 и с гарантией консистентности данных для 8 байтов (ведомое DP-устройство 2).

Указанные байты пользовательских данных ведомого (slave) DP-устройства 1 размещаются в области для пересылки (transfer area) ведущего (master) DP-устройства на P-шине или в CPU и могут быть пересланы, например, в блок данных в памяти CPU с помощью операторов Load или Transfer, также как это выполняется в центральных модулях.

Указанные байты пользовательских данных ведомого (slave) DP-устройства 2 также все размещаются в области для пересылки (transfer area) ведущего (master) DP-устройства, но доступ к ним возможен только с начальным адресом модуля (48, согласно примеру) на P-шине.

Оставшиеся адреса могут (теоретически) быть свободно назначены, но они блокируются (disabled) Hardware Configuration для иного применения.

С помощью системных функций SFC 14 и SFC 15 организуется доступ к пользовательским данным ведомого (slave) DP-устройства 2 с использованием начального адреса модуля (48) и выполняется обмен

данными с областями памяти CPU, например, с блоком данных.

Этот адрес не доступен при использовании операторов пересылки Load и Transfer, а также даже для операционной системы при обновлении областей отображения процесса. В нашем примере образ процесса не обновляется в области между адресами 48 и 55. Тем не менее, Вы можете определять для системных функций SFC 14 и SFC 15 области-источники и области назначения и таким образом использовать эти адреса.

Память для передачи данных в интеллектуальных ведомых DP-устройствах (Transfer memory)

В случае использования компактных или модульных ведомых (slave) DP-устройств адреса входов и выходов локализуются в I/O области центрального CPU (упоминаемого ниже как "CPU ведущего (master) DP-устройства" или "master CPU").

В случае использования интеллектуальных ведомых (slave) DP-устройств CPU ведущего (master) DP-устройства не имеет прямого доступа ко входным/выходным модулям ведомого (slave) DP-устройства. Поэтому каждое интеллектуальное ведомое (slave) DP-устройство имеет "память для передачи" ("transfer memory"), которая может быть разделена на несколько дополнительных областей различной длины и консистентности данных. Тогда с точки зрения ведущего (master) DP-устройства интеллектуальное ведомое (slave) DP-устройство выступает как компактное или модульное ведомое (slave) DP-устройство, в завистимости от разделения.

Размер "памяти для передачи" ("transfer memory") в целом зависит от ведомого (slave) DP-устройства. Вы можете определить адреса "памяти для передачи" в конфигурации: адреса с точки зрения CPU ведомого (slave) DP-устройства при конфигурировании интеллектуальных ведомых (slave) DP-устройств и адреса с точки зрения CPU ведущего (master) DP-устройства при вставке интеллектуальных ведомых (slave) DP-устройств в систему ведущего (master) DP-устройства. Исключение: если DP-интерфейс для интеллектуальных ведомых устройств формирует коммуникационный процессор CP 342-5DP, то разбиение его "памяти для передачи" ("transfer memory") не конфигурируется, пока ведомое устройство не будет подключено к системе ведущего (master) DP-устройства.

С точки зрения ведущего (master) DP-устройства (или точнее, с точки зрения центрального CPU ведущего (master) DP-устройства) адреса "памяти для передачи" не должны перекрываться с адресами других модулей в центральной S7-станции. С точки зрения CPU ведомого (slave) DP-устройства адреса "памяти для передачи" не должны перекрываться с адресами других модулей, встроенных в интеллектуальное ведомое (slave) DP-устройство.

Области адресов "памяти для передачи" представляют собой как бы отдельные модули с точки зрения доступа к пользовательским данным и консистентности данных, т.е., самый младший адрес адресного пространства является начальным (базовым) адресом модуля ("module starting address"). В соответствии с уровнем установки консистентности Вы можете передавать пользовательские данные для адресных областей либо с помощью операторов Load / Transfer, либо с помощью системных функций SFC 14 / SFC 15, в обоих случаях используя программу для CPU

ведущего (master) DP-устройства и программу для CPU ведомого (slave) DP-устройства. В программе для CPU ведомого (slave) DP-устройства Вы также можете использовать системную функцию SFC 7 DP_PRAL для запуска прерывания процесса в CPU ведущего (master) DP-устройства для адресного пространства.

На рис. 20.8 показан пример использования "памяти для передачи" интеллектуального ведомого (slave) DP-устройства для двух областей адресов.

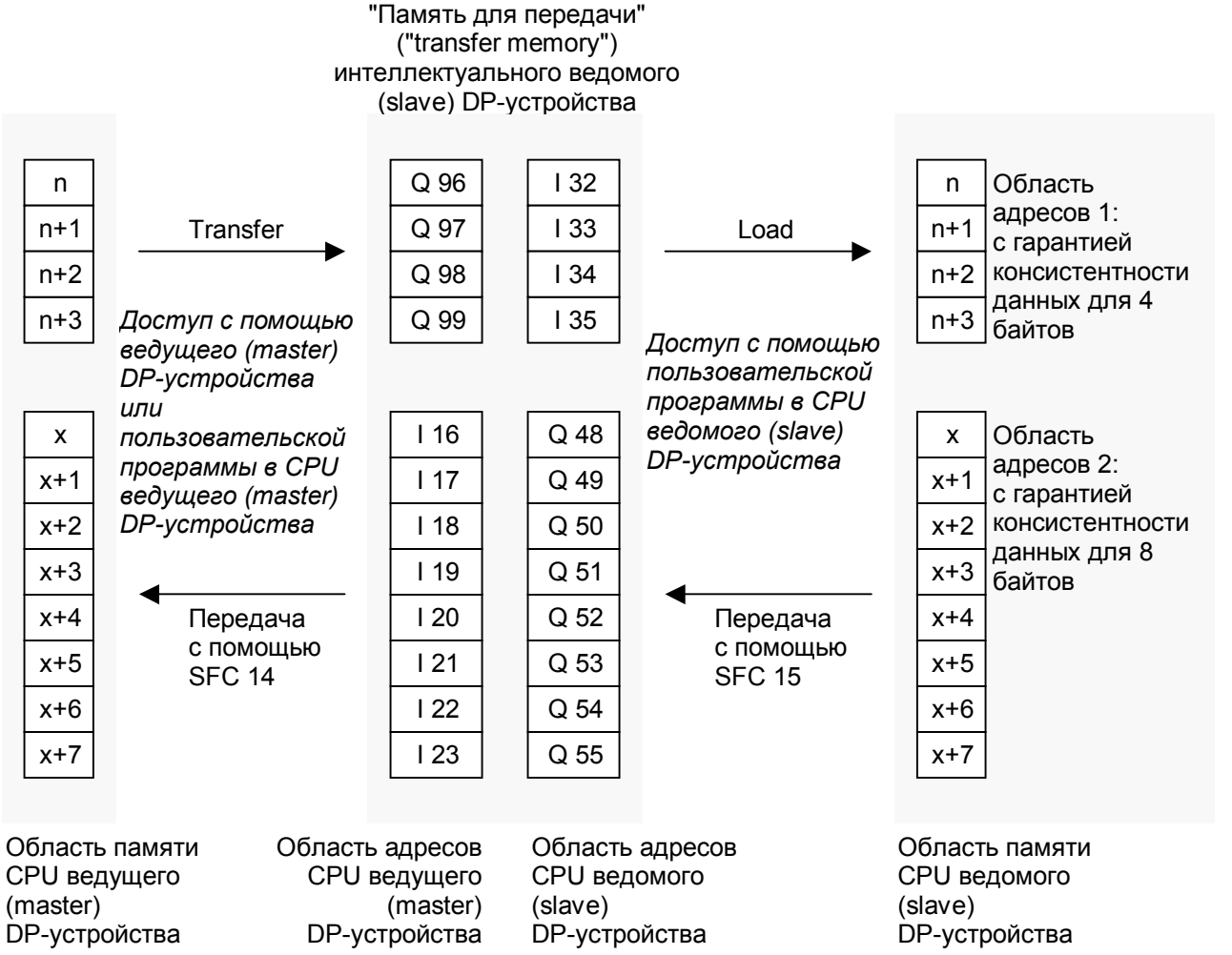


Рис. 20.8 Память для передачи данных в интеллектуальных ведомых DP-устройствах.

CPU ведущего (master) DP-устройства "видит" адреса области 1 как адреса выходного модуля, данные из которых могут быть считаны с помощью операций пересылки (или с помощью операций с отдельными битами, если адреса располагаются в области отображения процесса). Для CPU ведомого (slave) DP-устройства область адресов 1 является как бы входным модулем, данные из которых могут быть считаны посредством оператора Load или с помощью операции проверки (опроса),

если адреса располагаются в области отображения процесса. Адреса области 2 с гарантией консистентности данных объемом 8 байтов могут быть записаны CPU ведомого (slave) DP-устройства только с помощью функции SFC 15 и могут быть считаны CPU ведущего (master) DP-устройства только с помощью функции SFC 14.

Диагностический адрес (Diagnostics Address)

Каждое ведущее (master) DP-устройство и ведомое (slave) DP-устройство занимает один дополнительный байт "диагностического адреса". С помощью диагностического адреса Вы можете считывать данные диагностики.

Для ведомых (slave) DP-устройств стандарта DP V0 для считывания данных диагностики используется системная функция SFC 13 DPNRM_DG. При применении ведомых (slave) DP-устройств стандарта DP S7 для считывания записи DS 1, которая содержит данные диагностики, используется системная функция SFC 59 RD_REC. Диагностические данные, считываемые с помощью функции SFC 13 DPNRM_DG, имеют структуру, определенную стандартом (см. рис. 20.9).

Общая структура диагностических данных ведомых (slave) DP-устройств стандарта DP V0

Байт	
0 ... 2	Состояние станции 1, 2 и 3
3	Номер ведущей (master) станции
4 ... 5	ID производителя
6 ... n	Другие диагностические данные, определяемые типом ведомых (slave) DP-устройств

Общая структура диагностических данных ведомых (slave) DP-устройств стандарта DP S7

Байт	
0 ... 3	Запись данных DS 0 (стартовая информация в блоке OB 82)
4 ... n	Другие диагностические данные, определяемые типом ведомых (slave) DP-устройств

Рис. 20.9 Структура стандартных диагностических данных (Standard Diagnostics Data) и записи диагностических данных DS 1 (Diagnostics Data Record DS 1).

Записи данных DS 1, считываемые с помощью функции SFC 59 RD_REC, содержат 4 байта с диагностической информацией модуля, которую также можно найти, например, в стартовой информации организационного блока обработки диагностических прерываний OB 82 (это соответствует записи диагностических данных DS 0 [Diagnostics Data Record DS 0]). Структура остальной части диагностических данных определяется модулем.

Модули или устройства, которые не имеют "своих собственных" пользовательских данных, также могут иметь диагностический адрес, если они способны генерировать данные диагностики, как например, ведущее (master) DP-устройство или резервный (согласно конфигурации) источник питания.

Диагностический адрес занимает один байт в области периферийных входов (peripheral inputs). STEP 7 назначает диагностический адрес, по умолчанию начинающийся со старшего адреса I/O-области CPU. При просмотре адресов в таблице конфигурации оборудования Hardware Configuration диагностический адрес помечается звездочкой.

20.4.2 Конфигурирование распределенной периферии (I/O)

Общая процедура

Конфигурирование распределенной периферии (I/O) похоже на конфигурирование центральных модулей. Только вместо компоновки модулей в монтажной стойке Вы должны выполнить назначение DP-станций (PROFIBUS-узлов) системе ведущего (master) DP-устройства. При конфигурировании рекомендуется придерживаться следующего порядка действий:

1) Создайте новый или откройте существующий проект, используя SIMATIC Manager.

2) В проекте с помощью SIMATIC Manager создайте подсеть PROFIBUS и, если нужно, измените шинный профиль (bus profile).

3) Используя SIMATIC Manager, создайте в проекте ведущую (master) станцию, которая должна будет содержать ведущее (master) DP-устройство, например, станцию S7-400.

   Если Ваша система содержит интеллектуальные ведомые (slave) DP-устройства, Вам также необходимо создать здесь же соответствующие ведомые станции, например, станции S7-300.

   Теперь Вы должны запустить утилиту конфигурирования оборудования Hardware Configuration посредством открытия ведущей (master) станции.

4) С помощью утилиты конфигурирования оборудования Hardware Configuration поместите ведущее (master) DP-устройство в ведущую (master) станцию. Это может быть, например, CPU со встроенным DP-интерфейсом. Выполните назначение ранее созданной подсети PROFIBUS для DP-интерфейса; теперь Вы располагаете системой ведущего (master) DP-устройства. В дальнейшем также Вы можете сконфигурировать остальные модули. Сохраните и скомпилируйте станцию.

5) Если Вы создали S7-станцию для интеллектуальных ведомых (slave) DP-устройств, откройте ее, используя утилиту конфигурирования оборудования Hardware Configuration, и методом перетаскивания смонтируйте в ней модуль с требуемым DP-интерфейсом, например, S7-300 CPU со встроенным DP-интерфейсом или интеллектуальный базовый модуль BM 147/CPU станции распределенного ввода/вывода ET 200X. Если Вы установили DP-интерфейс как "ведомое (slave) DP-устройство", теперь назначьте для DP-интерфейса ранее созданную подсеть PROFIBUS, затем сконфигурируйте интерфейс пользовательских данных с точки зрения ведомого DP-устройства (память для передачи данных "transfer memory").

В дальнейшем Вы также можете сконфигурировать остальные модули. Сохраните и скомпилируйте станцию.

Теперь таким же образом выполните действия в отношении остальных станций, предназначенных для интеллектуальных ведомых (slave) DP-устройств.

6) Откройте ведущую (master) станцию с системой ведущего DP-устройства и, используя манипулятор "мышь", "перетащите" PROFIBUS-узлы (компактные и модульные ведомые [slave] DP-устройства) из каталога оборудования (hardware catalog) в систему ведущего DP-устройства. Назначьте адреса для узлов и, если необходимо, установите начальные адреса и диагностические адреса модулей.

7) Если Вы создали интеллектуальные ведомые (slave) DP-устройства, то используя манипулятор "мышь", "перетащите" соответствующие "иконки" (из каталога оборудования [hardware catalog] из разделов "PROFIBUS DP" и "Configured Stations" [сконфигурированные станции]) в систему ведущего DP-устройства.

   Активируйте (открывайте) оборудование щелчком по соответствующей "иконке" и назначайте для него уже созданное ведомое DP-устройство ("Connect" - "связать, соединить"), назначайте адрес узла (node address) и конфигурируйте интерфейс пользовательских данных с точки зрения ведущего (master) DP-устройства (или с точки зрения центрального CPU ведущего (master) DP-устройства).

   Теперь таким же образом выполните действия в отношении каждого из оставшихся интеллектуальных ведомых (slave) DP-устройств.

8) Сохраните и скомпилируйте все станции. Теперь Вы располагаете сконфигурированной системой ведущего (master) DP-устройства, и можете выполнить конфигурирование для центральных модулей или для дополнительных ведомых (slave) DP-устройств.

Вы можете также представить сконфигурированную таким образом систему ведущего DP-устройства графически, используя утилиту конфигурирования сетей Network Configuration. Откройте утилиту конфигурирования сетей Network Configuration одним из способов, например, двойным щелчком манипулятора "мышь" на значке подсети. Выберите следующие опции: *View -> DP Slaves (Вид -> ведомые DP-устройства)* для отображения ведомых (slave) DP-устройств. С помощью утилиты конфигурирования сетей Network Configuration Вы можете также создать систему ведущего (master) DP-устройства (или, более точно, назначить узлы для подсети PROFIBUS). При использовании утилиты конфигурирования сетей Network Configuration, после открытия станций можно их параметризировать. Здесь же Вы должны сначала установить интеллектуальные ведомые (slave) DP-устройства, прежде чем вводить их в систему ведущего DP-устройства.

Конфигурирование ведущего (master) DP-устройства

Сначала Вы должны создать проект и S7-станцию, используя SIMATIC Manager. Затем Вы должны открыть S7-станцию и создать монтажную стойку (см. раздел 2.3 "Конфигурирование станций").

Теперь Вы можете, используя манипулятор "мышь", "перетащить" модуль ведущего DP-устройства из каталога оборудования (hardware catalog) в таблицу конфигурации (configuration table) монтажной стойки. У Вас уже может быть выбран CPU с DP-интерфейсом. Строкой ниже отображается ведущее (master) DP-устройство с подключением к системе ведущего DP-устройства в окне станции.

При "монтировании" модуля ведущего DP-устройства Вы выбираете в окне подсеть PROFIBUS, для которой должна быть назначена система ведущего DP-устройства и адрес узла, назначаемый ведущему DP-устройству. В этом окне Вы можете также создать новую подсеть PROFIBUS.

Если система ведущего DP-устройства недоступна (например, она находится вне области видимости или закрыта объектом), создайте систему ведущего DP-устройства, выбрав ведущее (master) DP-устройство в окне конфигурации с последующим выбором опций: *Insert -> DP Master System (Вставка -> Система ведущего DP-устройства)*. Вы можете изменять адрес узла и соединение с подсетью PROFIBUS, выбрав модуль, а затем выбирая опции: *Edit -> Object Properties (Правка -> Свойства объекта)*, используя вкладку "General" ("Общие") и кнопку "Properties" ("Свойства"), после чего Вы получаете доступ к свойствам объекта.

CP 342-5DP как ведущее DP-устройство

Если ведущим DP-устройством является коммуникационный процессор CP 342-5DP, тогда поместите его в таблицу конфигурации станции; выберите его как объект, затем используйте опции меню: *Edit -> Object Properties (Правка -> Свойства объекта)*. В окне свойств объекта на вкладке "Mode" ("Режим") установите "DP Master" ("Ведущее DP-устройство").

На вкладке "Addresses" ("Адреса") указывается адрес, который занимает CP в области данных пользователя в адресном пространстве CPU. С точки зрения CPU ведущего (master) устройства коммуникационный процессор CP 342-5DP является "аналоговым модулем" ("analog module"), характеризующимся начальным адресом модуля и с 16 байтами пользовательских данных.

К коммуникационному процессору CP 342-5DP как к ведущему (master) DP-устройству могут быть подключены только стандартные ведомые DP-устройства ("DP standard slave") или ведомые S7 DP-устройства ("DP S7 slave"), ведущие себя как стандартные ведомые DP-устройства. Вы можете выбрать необходимые ведомые (slave) DP-устройства в каталоге оборудования (hardware catalog) в разделе "PROFIBUS-DP" / "CP 342-5DP as DP Master" (CP 342-5DP как ведущее DP-устройство). Для этого Вы можете выделить требуемый тип ведомого (slave) DP-устройства и "перетащить" его с помощью манипулятора "мышь" в систему ведущего (master) DP-устройства.

Память для передачи данных (transfer memory) у CP 342-5DP как у ведущего DP-устройства имеет максимальную длину 240 байтов. Данные пересылаются с помощью загружаемых блоков FC 1 DP_SEND и FC 2 DP_RECV (включенных в стандартную библиотеку *Standard Library* в разделе *Communication Blocks [коммуникационные блоки]*).

Консистентность данных обеспечивается в области памяти для передачи данных в целом.

Вы можете считывать диагностические данные подключенных ведомых (slave) DP-устройств с помощью FC 3 DP_DIAG (например, список станций, диагностические данные отдельной станции).

Блок FC 4 DP_CTRL позволяет переслать задания управления для CP 342-5DP (например, команды SYNC/FREEZE, CLEAR, команды установки рабочего режима CP 342-5DP).

Если Вы выберете CPU или CP 342-5DP, то с помощью опций меню: *View -> Address Overview (Вид -> Обзор адресов)* Вы можете увидеть список назначенных адресов, входов и/или выходов. Вы можете также увидеть интервалы неиспользованных адресов.

Конфигурирование компактных ведомых (slave) DP-устройств

Компактные ведомые DP-устройства (compact DP slave) должны находиться в каталоге оборудования (hardware catalog) в разделе "PROFIBUS-DP" в соответствующем подразделе, например, в ET 200B. Выделите требуемое ведомое (slave) DP-устройство и "перетащите" его с помощью манипулятора "мышь" на значок системы ведущего (master) DP-устройства.

Вы увидите окно свойств станции; здесь можно задать адрес узла и любой диагностический адрес. Значок ведомого (slave) DP-устройства появляется в верхней части окна свойств станции, а в нижней части окна располагается таблица конфигурации для этой станции.

С помощью двойного щелчка кнопкой "мыши" на значке в верхней части окна станции можно открыть диалоговое окно с одной или несколькими вкладками, с помощью которого Вы можете установить требуемые свойства станции. В нижней части окна при этом отображаются адреса входов/выходов. С помощью двойного щелчка кнопкой "мыши" на строке с адресом вызывается окно, в котором Вы можете изменять значения требуемых адресов.

В нижней части окна по выбору отображается таблица конфигурации для выбранного ведомого DP-устройства или для системы ведущего (master) DP-устройства (при этом переключение режима отображения выполняется с помощью навигационных клавиш клавиатуры [со стрелками]).

Конфигурирование модульных ведомых (slave) DP-устройств

Модульные ведомые DP-устройства (modular DP slave) должны находиться в каталоге оборудования (hardware catalog) в разделе "PROFIBUS-DP" в соответствующем подразделе, например, в ET 200M.

Щелкните кнопкой манипулятора "мышь" на выбранном интерфейсном модуле (базовом модуле [basic module]) и "перетащите" его с помощью манипулятора "мышь" на значок системы ведущего (master) DP-устройства. Вы увидите окно свойств станции; здесь можно задать адрес узла и диагностический адрес. Значок ведомого (slave) DP-устройства появляется в верхней части окна свойств станции, а в нижней части окна располагается таблица конфигурации для этой станции.

Теперь поместите в таблицу конфигурации модули, которые Вы выберете в каталоге оборудования (hardware catalog), *под выбранным интерфейсным модулем*. С помощью двойного щелчка кнопкой "мыши"

на соответствующей строке можно открыть диалоговое окно свойств модуля, после чего Вы можете параметризировать модуль.

Необходимо отметить, что адресное пространство каждого ведомого устройства из системы ведущего (master) DP-устройства и адресное пространство интерфейсного модуля ведомого (slave) DP-устройства ограничены. Например, предельное значение для CPU 315-2DP как ведущего DP-устройства составляет 122 байта для входов и 122 байта для выходов на каждое ведомое (slave) DP-устройство (восемь 8-канальных модулей в ET 200M превысят этот предел: 8 × 16 = 128 байтов), а предельное значение для ET 200X как ведомого DP-устройства составляет 104 байта для входов и 104 байта для выходов.

Конфигурирование CPU со встроенным DP-интерфейсом, используемого в качестве интеллектуального ведомого (slave) DP-устройства

При использовании соответствующих CPU Вы можете параметризировать станцию или как ведущую (master) DP-станцию или как ведомую (slave) DP-станцию. Сначала станцию необходимо создать, а затем подключить как ведомую (slave) DP-станцию к системе ведущего (master) DP-устройства. В этом случае используется точно такая же процедура, как для "обычной" станции: используя SIMATIC Manager, вставьте в проект S7-станцию, затем Вы должны открыть S7-станцию и "перетащить" в окно Hardware Configuration (конфигурация оборудования) монтажную стойку, используя манипулятор "мышь", затем поместите в стойку требуемые модули. При конфигурировании ведомой (slave) DP-станции достаточно в стойке "установить" CPU - остальные модули Вы сможете добавить позже.

При вставке CPU будет отображен список свойств PROFIBUS-интерфейса. В нем Вы должны назначить тип подсети (subnetwork) для DP-интерфейса, а также назначить адрес. Если подсеть PROFIBUS еще не существует в проекте, то Вы можете создать новую подсеть с помощью кнопки "New" ("Новая"). Это та самая подсеть, к которой в дальнейшем будет подключено интеллектуальное ведомое (slave) DP-устройство.

Список свойств PROFIBUS-интерфейса Вы можете открыть одним из способов: сначала выбрать DP-интерфейс, затем - опции меню: *Edit -> Object Properties (Правка -> Свойства объекта)* или сначала дважды щелкнуть кнопкой манипулятора "мышь" на значке PROFIBUS-интерфейса. В окне свойств объекта на вкладке "Operating Mode" ("Рабочий режим") установите "DP Slave" ("Ведомое DP-устройство"). Теперь Вы можете сконфигурировать интерфейс пользовательских данных на вкладке "Configuration" ("Конфигурация") с точки зрения ведомого (slave) DP-устройства (см. рис. 20.10). Информацию по интерфейсу пользовательских данных Вы можете найти в разделе 20.4.1 "Адресация распределенной периферии (I/O)" в параграфе "Память для передачи данных в интеллектуальных ведомых (slave) DP-устройствах".

Размер и структура памяти для передачи данных (transfer memory) определяются типом CPU. Например, для CPU 315-2DP Вы можете разделить всю область памяти для передачи данных (transfer memory) на 32 адресных области, к которым обеспечивается отдельный доступ. Такие адресные области могут иметь размер до 32 битов. Область памяти для передачи данных в целом может иметь до 122 входных и до 122 выходных адресов. Эти адреса находятся в адресном пространстве CPU ведомого (slave) устройства и они не должны перекрываться с адресами

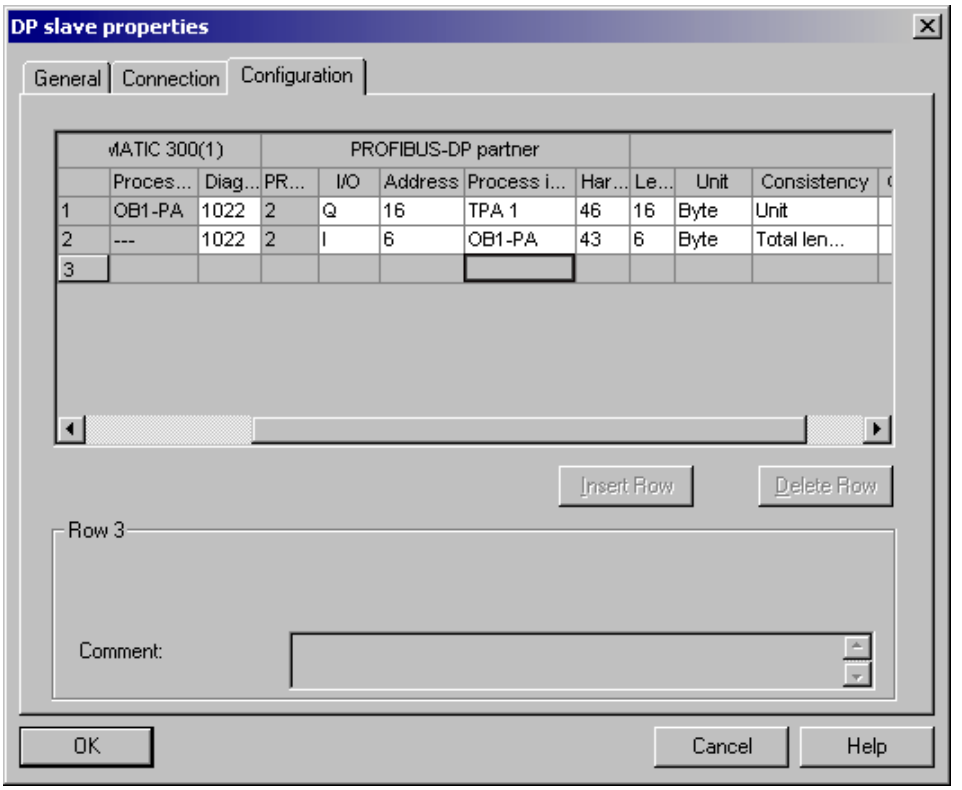


Рис. 20.10 Конфигурирование памяти для передачи данных в интеллектуальных ведомых (slave) DP-устройствах со встроенным DP-интерфейсом

центральных или периферийных модулей в ведомой (slave) DP-станции. Младший адрес в адресном пространстве является начальным адресом модуля ("module starting address").

Пользовательская программа в CPU ведомого (slave) устройства получает диагностическую информацию от ведущего (master) DP-устройства с использованием диагностических адресов, заданных на данной вкладке.

Завершая конфигурирование интеллектуального ведомого (slave) DP-устройства, сохраните и скомпилируйте станцию с помощью опций меню: *Station -> Save and Compile (Станция -> Сохранить и скомпилировать).* Подключение интеллектуального ведомого (slave) DP-устройства к системе ведущего (master) DP-устройства описано ниже.

Конфигурирование BM 147/CPU, используемого в качестве интеллектуального ведомого (slave) DP-устройства

При необходимости создания станции ET 200X как интеллектуальной ведомой (slave) DP-станции необходимо выполнить следующие действия: используя SIMATIC Manager, вставьте в проект SIMATIC 300 станцию, затем откройте объект *Hardware (оборудование).*

В Hardware Configuration (конфигурация оборудования), используя манипулятор "мышь", "перетащите" объект *BM 147/CPU* из каталога оборудования из разделов "PROFIBUS-DP" / "ET200X" в свободное окно или выберите объект двойным щелчком. Выберите адрес узла и подсеть

PROFIBUS в появившемся окне свойств для DP-интерфейса (или же Вы можете создать новую подсеть и назначить ее для DP-интерфейса). Теперь у Вас есть таблица конфигурации, соответствующая станции SIMATIC 300. Отображаемый CPU представляет здесь интеллектуальный модуль BM 147 для станции ET200X. Этот CPU не имеет MPI-интерфейса (BM 147/CPU программируется посредством MPI-интерфейса ведущей [master] станции).

Двойным щелчком на строке CPU Вы можете открыть окно свойств CPU; двойным щелчком на символе интерфейса Вы можете открыть окно свойств DP-интерфейса. Задайте область адресов для интерфейса пользовательских данных с точки зрения ведомого (slave) DP-устройства. Для BM 147 начальный адрес имеет фиксированное значение 128; максимальный размер области пользовательских данных составляет 32 байта для входов и 32 байта для выходов. Вы можете разделить эту область памяти на 8 подобластей с различной консистентностью данных. Для диагностического адреса устанавливается значение 127. Пользовательская программа получает диагностическую информацию от ведущего (master) DP-устройства с использованием данного диагностического адреса.

Дальнейшее конфигурирование станции ET 200X выполняется таким же образом, как и конфигурирование S7 300-станции с фиксированной адресацией слотов. Только требуемые модули выбираются в каталоге оборудования из раздела "BM 147/CPU".

Завершая конфигурирование интеллектуального ведомого (slave) DP-устройства, сохраните и скомпилируйте станцию с помощью опций меню: *Station -> Save and Compile (Станция -> Сохранить и скомпилировать).* Подключение интеллектуального ведомого (slave) DP-устройства к системе ведущего (master) DP-устройства описано ниже.

Конфигурирование IM 151/CPU, используемого в качестве интеллектуального ведомого (slave) DP-устройства

При необходимости создания станции ET 200S как интеллектуальной ведомой (slave) DP-станции необходимо выполнить следующие действия: используя SIMATIC Manager, вставьте в проект SIMATIC 300 станцию, затем откройте объект *Hardware (оборудование).*

В Hardware Configuration (конфигурация оборудования), используя манипулятор "мышь", "перетащите" объект *IM 151/CPU* из каталога оборудования из разделов "PROFIBUS-DP" / "ET200S" в свободное окно или выберите объект двойным щелчком. Выберите адрес узла и подсеть PROFIBUS в появившемся окне свойств для DP-интерфейса (или же Вы можете создать новую подсеть и назначить ее для DP-интерфейса). Теперь Вы можете видеть таблицу конфигурации, такую же как для станции SIMATIC 300. Отображаемый CPU представляет здесь интеллектуальный интерфейсный модуль IM 151 для станции ET200S. Этот CPU не имеет MPI-адреса в таблице адресов, так как не имеет MPI-интерфейса (IM 151/CPU программируется посредством MPI-интерфейса ведущей [master] станции).

Двойным щелчком на строке CPU Вы можете открыть окно свойств CPU; двойным щелчком на символе интерфейса Вы можете открыть окно свойств DP-интерфейса. Задайте здесь область адресов для интерфейса пользовательских данных с точки зрения ведомого (slave) DP-устройства.

Для IM 151/CPU максимальный размер области пользовательских данных составляет 64 байта. Вы можете разделить эту область памяти на 8 подобластей с различной консистентностью данных. Пользовательская программа получает диагностическую информацию от ведущего (master) DP-устройства с использованием диагностического адреса.

Дальнейшее конфигурирование станции ET 200S выполняется таким же образом, как и конфигурирование S7 300-станции с фиксированной адресацией слотов. Только требуемые модули выбираются в каталоге оборудования из раздела "IM 151/CPU".

Завершая конфигурирование интеллектуального ведомого (slave) DP-устройства, сохраните и скомпилируйте станцию с помощью опций меню: *Station -> Save and Compile (Станция -> Сохранить и скомпилировать).* Подключение интеллектуального ведомого (slave) DP-устройства к системе ведущего (master) DP-устройства описано ниже.

Конфигурирование станции S7-300 с CP 342-5DP, используемой в качестве интеллектуального ведомого (slave) DP-устройства

Если есть такая необходимость, вставьте в проект SIMATIC 300 станцию, затем откройте объект *Hardware (оборудование)* и сконфигурируйте как обычно ("normal") станцию S7-300. Помимо других свойств в таблице конфигурации сконфигурируйте коммуникационный процессор CP 342-5DP.

После того, как Вы вставите в проект SIMATIC 300 станцию, появится окно свойств для DP-интерфейса; в этом окне необходимо для DP-интерфейса назначить подсеть, к которой необходимо будет в дальнейшем подключить интеллектуальное ведомое (slave) DP-устройство, также здесь Вы должны назначить адрес узла (node address).

Для того чтобы открыть окно свойств, выберите CP 342-5DP, а затем - опции меню: *Edit -> Object Properties (Правка -> Свойства объекта)*, или дважды щелкнув кнопкой мыши на значке CP 342-5DP, после чего Вы получите доступ к свойствам объекта. На вкладке "Mode" ("Режим") выберите опцию "DP Slave" ("Ведомое DP-устройство").

На вкладке "Addresses" ("Адреса") отображается интерфейс для области данных пользователя с точки зрения CPU ведомого (slave) устройства (начальный адрес модуля и с 16 байтов пользовательских данных).

Память для передачи данных (transfer memory) для CP 342-5DP с точки зрения ведомого (slave) DP-устройства имеет максимальную длину 86 байтов, при этом Вы можете разделить эту область памяти на различные по размеру подобласти адресов после подключения к системе ведущего (master) DP-устройства.

Завершая конфигурирование интеллектуального ведомого (slave) DP-устройства, сохраните и скомпилируйте станцию с помощью опций меню: *Station -> Save and Compile (Станция -> Сохранить и скомпилировать).* Подключение интеллектуального ведомого (slave) DP-устройства к системе ведущего (master) DP-устройства описано ниже.

Подключение интеллектуального ведомого (slave) DP-устройства к системе ведущего (master) DP-устройства

Для подключение интеллектуального ведомого (slave) DP-устройства к

системе ведущего (master) DP-устройства у Вас должен быть создан проект, а также сконфигурированы ведущая (master) DP-станция и интеллектуальное ведомое (slave) DP-устройство (в каждом случае по крайней мере с DP-интерфейсом). И ведущее (master) DP-устройство, и ведомое (slave) DP-устройство должны быть сконфигурированы для одной подсети PROFIBUS.

Откройте ведущую (master) станцию. Вы должны убедиться, что система ведущего (master) DP-устройства (прямоугольник, выделенный пунктиром) существует; если это не так, создайте систему, используя опции меню: *Insert -> DP Master System (Вставка -> Система ведущего DP-устройства).* В каталоге оборудования в разделах "PROFIBUS-DP" и "Configured Stations" ("Сконфигурированные станции") Вы можете найти объекты, представляющие собой интеллектуальные ведомые (slave) устройства:

"CPU31x-2DP" представляет станции S7-300 со встроенным ведомым (slave) DP-устройством;

"X-BM147/CPU" представляет станции ET 200X с базовым модулем BM147/CPU;

"ET 200S/CPU" представляет интеллектуальное ведомое (slave) DP-устройство (станцию) ET 200S DP;

"S7-300 CP342-5DP" представляет станции S7-300 с коммуникационным процессором CP342-5 как с ведомым (slave) интерфейсным DP-модулем.

Выберите требуемый тип ведомого (slave) DP-устройства и "перетащите" его с помощью манипулятора "мышь" на систему ведущего (master) DP-устройства.

CPU, ET 200X или ET 200S как ведомые (slave) DP-устройства

При "перетаскивании" ведомого (slave) DP-устройства посредством манипулятора "мышь" на систему ведущего (master) DP-устройства или при двойном щелчке на значке ведомого (slave) DP-устройства открывается список свойств объекта. Заранее сконфигурированные ведомые (slave) устройства для подсети PROFIBUS Вы можете найти на вкладке "Connection" ("Соединения"). Выберите требуемый тип ведомого (slave) DP-устройства и щелкните кнопкой манипулятора "мышь" на кнопке "Connect" ("Соединить"). Это действие приведет к активации требуемого соединения (см. в нижней части того же диалогового окна).

Задайте диагностический адрес ведомого (slave) DP-устройства с точки зрения системы ведущего (master) DP-устройства на вкладке "General" ("Общие").

На вкладке "Configuration" ("Конфигурация") установите адреса интерфейса пользовательских данных ведомого (slave) DP-устройства с точки зрения системы ведущего (master) DP-устройства. Адреса выходов ведущего (master) DP-устройства являются адресами входов ведомого (slave) DP-устройства и наоборот. Более подробную информацию по интерфейсу пользовательских данных Вы можете найти в разделе 20.4.1 "Адресация распределенной периферии I/O" в параграфе "Память для передачи данных в интеллектуальных ведомых DP-устройствах (Transfer memory)".

CP 342-5DP как ведомые (slave) DP-устройства

При "перетаскивании" ведомого (slave) DP-устройства посредством манипулятора "мышь" на систему ведущего (master) DP-устройства или при двойном щелчке на значке ведомого (slave) DP-устройства открывается список свойств объекта. Ранее сконфигурированные ведомые (slave) устройства для подсети PROFIBUS Вы можете найти на вкладке "Connection" ("Соединения"). Выберите требуемый тип ведомого (slave) DP-устройства и щелкните кнопкой манипулятора "мышь" на кнопке "Connect" ("Соединить"). Это действие приведет к активации требуемого соединения (см. в нижней части того же диалогового окна).

Если Вы выбрали ведомое (slave) DP-устройство, таблица конфигурации, соответствующая этому устройству, будет показана в нижней части окна станции. Теперь Вы можете структурировать "память для передачи" ("transfer memory"). Для этого выберите "Universal submodule" ("Универсальный подмодуль") в каталоге оборудования (hardware catalog) из раздела для используемого коммуникационного процессора (CP), затем, используя манипулятор "мышь", "перетащите" объект из каталога оборудования в свободную строку таблицы конфигурации или выберите строку в таблице и дважды щелкните на объекте "Universal submodule" ("Универсальный подмодуль") в каталоге оборудования. Для каждой отдельной (консистентной) адресной области в памяти для передачи ("transfer memory") Вы должны размещать по одному универсальному подмодулю; максимальное их число может быть равно 32.

Для того чтобы открыть окно, в котором Вы определяете свойства адресной области, Вы можете выделить универсальный подмодуль, а затем выбрать опции меню: *Edit -> Object Properties (Правка -> Свойства объекта)*, или откройте свойства двойным щелчком на выделенной строке таблицы. Определите здесь тип модуля: входной, выходной, или модуль входов/выходов. Определите здесь также начальный адрес и длину области. Определенные таким образом адреса размещаются в адресном пространстве CPU ведущего (master) устройства. Область может иметь размер до 64 байт; максимальный общий размер памяти для передачи ("transfer memory") может достигать 86 байт.

Если CP 342-5DP является ведущим (master) DP-устройством, то операция структурирования памяти для передачи ("transfer memory") может быть пропущена, так как коммуникационный процессор CP 342-5DP пересылает данные из памяти для передачи ("transfer memory") единым массивом за один прием.

При разбиении "памяти для передачи" ("transfer memory") Вы организуете адресное пространство целиком без пропусков, начиная с 0-го байта. Вы можете получить доступ к целиком определенной памяти для передачи в CPU ведомого (slave) устройства с помощью загружаемых блоков FC 1 DP_SEND и FC 2 DP_RECV (включенных в стандартную библиотеку *Standard Library* в разделе *Communication Blocks [коммуникационные блоки]*).

Консистентность данных обеспечивается в области памяти для передачи данных в целом.

Задайте диагностический адрес ведомого (slave) DP-устройства с точки зрения системы ведущей (master) станции на вкладке "General" ("Общие"). Вы можете считывать диагностические данные с помощью FC 3 DP_DIAG (в ведущей [master] станции).

Более подробную информацию по интерфейсу пользовательских данных Вы можете найти в разделе 20.4.1 "Адресация распределенной периферии I/O" в параграфе "Память для передачи данных в интеллектуальных ведомых DP-устройствах (Transfer memory)".

GSE-файлы

Пользователь имеет возможность "отложенной инсталляции" ("post install") ведомых (slave) DP-устройств, которые не включены в каталог модулей (module catalog). Для этой цели используются файлы особого типа, специально создаваемые для этих ведомых (slave) DP-устройств (GSE-файлы, базы данных для устройств). Для инсталляции устройства необходимо выбрать опции меню: *Options -> Install GSE (Опции -> Инсталляция GSE-файла)* при выполнении конфигурирования оборудования. При этом в появившемся окне необходимо указать каталог, в котором располагается GSE-файл. Здесь же Вы можете определить пиктограмму, которую STEP 7 будет использовать для графического представления DP-устройства. Система STEP 7 считывает GSE-файл и отображает DP-устройство в каталоге оборудования (hardware catalog) в разделе "Other Field Devices" ("Другие полевые приборы").

Вы можете скопировать в текущий проект GSE-файлы, которые ранее были установлены в других S7-проектах, с помощью опций меню: *Options -> Import Station GSE (Опции -> Импортирование станции GSE)*.

Система STEP 7 сохраняет GSE-файлы в каталоге …\Step7\S7data\gsd. GSE-файлы, которые удаляются при выполняемых в дальнейшем операциях инсталляции или импортирования, сохраняются в подкаталоге …\gsd\bkpx. Из этого каталога GSE-файлы могут быть восстановлены при использовании опций меню: *Options -> Install New GSE (Опции -> Инсталляция нового GSE-файла)*.

Конфигурирование PROFIBUS-PA

Для конфигурирования системы ведущего (master) устройства PROFIBUS-PA и для параметризации полевых PA-приборов необходимо использовать поставляемое по особому заказу (опционное) программное обеспечение SIMATIC PDM. При конфигурировании оборудования (Hardware Configuration) Вы можете выполнить подключение к системе ведущего (master) DP-устройства посредством DP/PA-соединителя (DP/PA-Link).

Для этого из каталога оборудования (Hardware Catalog) необходимо выбрать интерфейсный модуль IM 151, используя манипулятор "мышь", и "перетащить" объект IM 151 из каталога оборудования из разделов "PROFIBUS-DP" и "DP/PA-Link" на систему ведущего (master) DP-устройства. При использовании ведомого (slave) DP-устройства аналогичным образом создается система ведущего (master) PA-устройства в своей собственной подсети PROFIBUS (45,45 кбит/с); эта система отображается пунктирным прямоугольником на схеме.

DP/PA-интерфейс обеспечивает передачу данных без всякого их изменения или преобразования между двумя шинными системами, и по этой причине этот интерфейс не параметризируется.

Полевые PA-приборы получают адреса от ведущего (master) PD-устройства. Они могут быть встроены в конфигурацию оборудования STEP 7 (Hardware Configuration) посредством GSE-файла. После этого Вы сможете найти соответствующие PA-устройства в каталоге оборудования (hardware catalog) в разделах "PROFIBUS-DP" и "Other Field Devices" ("Другие полевые приборы").

Конфигурирование DP/AS-интерфейсного соединителя (DP/AS-i-Link)

Конфигурирование DP/AS-интерфейсного соединителя (DP/AS-i-Link) выполняется как конфигурирование ведомого (slave) DP-устройства. Вы можете найти соответствующие модули, такие, например, как DP/AS-i-Link 20, в каталоге оборудования (hardware catalog) в разделах "PROFIBUS-DP" и "DP/AS-i", а затем "перетащить" посредством манипулятора "мышь" на систему ведущего (master) DP-устройства. В появившемся окне выполните сначала установки конфигурации (от 16 до 20 байт), затем задайте адрес узла.

Для соединителя DP/AS-i-Link 20 Вы можете выполнить установки конфигурации в 16 байтах для входов/выходов и еще в 4 байтах для команд управления. В последнем случае в нижней части окна конфигурирования оборудования Hardware Configuration предлагаются в 16 байтах пользовательские данные и в 4 байтах - команды с адресами, например, 512.

Вы можете выбрать ведомое (slave) DP-устройство, а затем выбрать опции меню: *Edit -> Object Properties (Правка -> Свойства объекта)*, или Вы можете двойным щелчком выбрать ведомое (slave) DP-устройство для того, чтобы открыть окно, в котором Вы можете изменить предложенные утилитой конфигурирования Hardware Configuration значения адресов, а также задать адресное поле для отображения подпроцесса (если Вы используете подходящий CPU).

Выберите ведомое (slave) DP-устройство, а затем выберите опции меню: *Edit -> Object Properties (Правка -> Свойства объекта)*, или дважды щелкните на ведомом (slave) DP-устройстве для того, чтобы открыть окно свойств объекта. На вкладке "Parameterize" ("Параметризация") установите параметры ведомых (slave) DP-устройств AS-i в 4-х битах для каждого устройства.

Система ведущего (master) AS-i-устройства с ведомыми (slave) AS-i-устройствами утилитой конфигурирования Hardware Configuration не отображается.

Конфигурирование групп SYNC/FREEZE

Команда управления SYNC вызывает для объединенных в группу ведомых (slave) DP-устройств одновременное (синхронное) изменение состояний выходов устройств. Команда управления FREEZE вызывает для объединенных в группу ведомых (slave) DP-устройств одновременное (синхронное) "замораживание" текущих состояний входов устройств, для того чтобы ведущее (master) DP-устройство могло их циклически считывать. Команды управления UNSYNC и UNFREEZE отменяют действие команд управления SYNC и FREEZE соответственно.

При этом необходимо отслеживать корректное выполнение данных функций как с точки зрения ведомых (slave) DP-устройств, так и ведущего (master) DP-устройства. С помощью окна свойств для каждого ведомого (slave) DP-устройства Вы можете контролировать, какие именно функции установлены для него. Для этого Вы можете выбрать ведомое (slave) DP-устройство, а затем выберите опции меню: *Edit -> Object Properties (Правка -> Свойства объекта)*, далее откройте вкладку "General" ("Общие") и смотрите установленные свойства SYNC и FREEZE "SYNC/FREEZE Capabilities".

Для системы каждого ведущего (master) DP-устройства пользователь может сформировать до 8 групп, объединенных функциями SYNC и FREEZE, которые активизируются или командой SYNC, или командой FREEZE, или одновременно двумя этими командами. При этом Вы можете назначить любое ведомое (slave) DP-устройство в группу. Что касается CP 342-5DP специальной версии, то одно ведомое (slave) DP-устройство может входить одновременно в несколько групп (максимально до 8 групп).

Вызывая системную функцию SFC 11 DPSYC_FR, Вы можете организовать формирование из пользовательской программы команды для соответствующей группы (см. раздел 20.4.3 "Системные функции для распределенной периферии I/O"). При этом ведущее (master) DP-устройство посылает соответствующую команду одновременно всем ведомым (slave) DP-устройствам соответствующей группы.

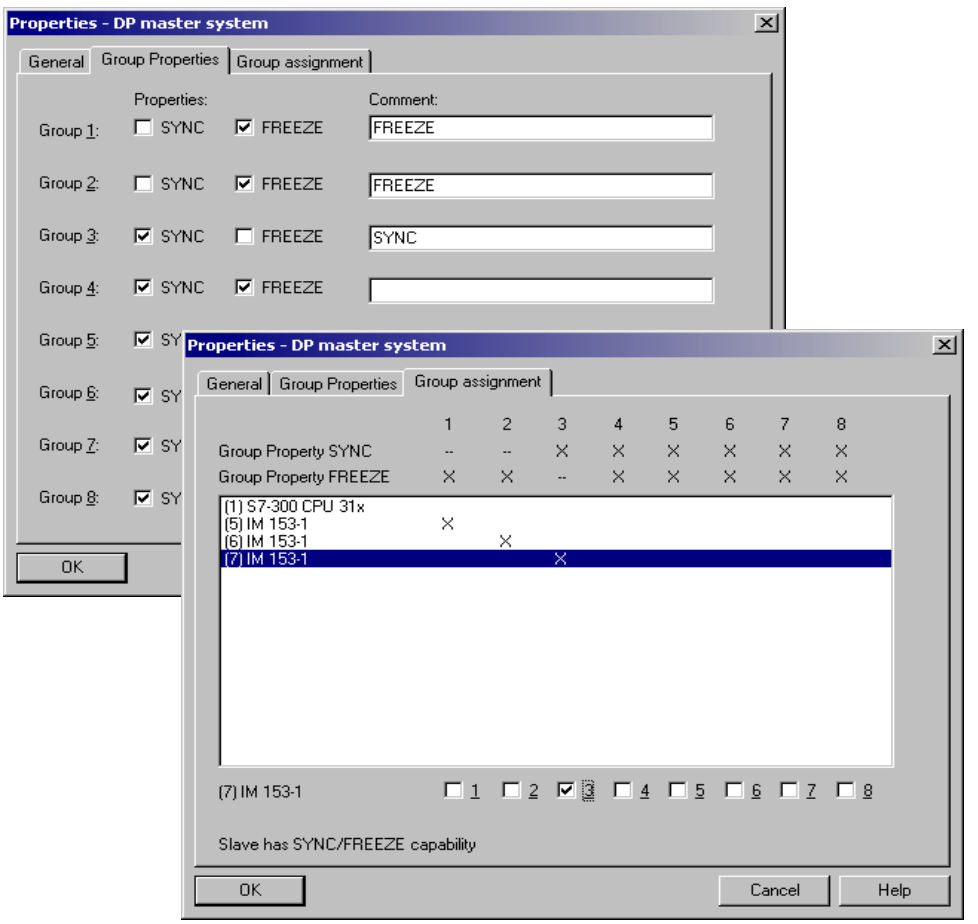


Рис. 20.11 Конфигурирование SYNC- и FREEZE-групп

Конфигурирование SYNC/FREEZE-группы должно выполняться после конфигурирования системы ведущего (master) DP-устройства (все ведомые (slave) DP-устройства должны присутствовать в системе). Выберите систему ведущего (master) DP-устройства (она выделена как прямоугольник с пунктирным контуром), а затем выберите опции меню: *Edit -> Object Properties (Правка -> Свойства объекта)*. Далее в появившемся окне на вкладке "Group Properties" ("Свойства группы") установите команды для групп, а затем на вкладке "Group Assignment" ("Назначения в группы") назначьте в отдельные группы соответствующие ведомые (slave) DP-устройства.

Здесь Вы последовательно выбираете из списка каждое ведомое (slave) DP-устройство с его номером узла (node number) и выбираете для каждого случая группу, которой данное устройство должно принадлежать. Если какое-либо ведомое (slave) DP-устройство не может выполнять определенную команду, например, FREEZE, группа, которая содержит данную команду, не сможет быть выбрана, например, все группы с командой FREEZE. Конфигурирование SYNC/FREEZE групп должно быть завершено нажатием кнопки "OK".

Необходимо отметить, что при конфигурировании одинаковых по длине (постоянных) шинных циклов (bus cycles) для групп 7 и 8 требуются специальные значения.

Конфигурирование постоянных шинных циклов (bus cycle time)

Обычно ведущее (master) DP-устройство управляет назначенными ему ведомыми (slave) DP-устройствами циклически непрерывно (без пауз). Для S7-коммуникаций, устанавливаемых, например, когда посредством программатора (PG), выполняется модификация функций посредством подсети PROFIBUS, это может приводить к изменениям во временных интервалах. Если, например, состояния выходов должны модифицироваться с использованием распределенной периферии I/O через равные интервалы, Вы можете назначить постоянные шинные циклы (bus cycle time) для специальных (соответствующим образом оснащенных) ведущих (master) DP-устройств. Используемое для этих целей ведущее (master) DP-устройство в подсети PROFIBUS должно относиться только к ведущим (master) DP-устройствам 1 класса (Class 1 master). Свойства постоянности шинных циклов (bus cycle time) могут быть установлены для "шинных профилей" "DP" и "User-Defined" ("пользовательский").

Вы можете открыть окно свойств подсети PROFIBUS следующим способом: выберите подсеть PROFIBUS, а затем выберите опции меню: *Edit -> Object Properties (Правка -> Свойства объекта)* в окне конфигурирования подсети. Далее в появившемся окне на вкладке "Network Settings" ("Установки подсети") щелкните на кнопке "Options" ("Опции"). Теперь на вкладке "Constant Bus Cycle Time" ("Постоянные шинные циклы") щелкните на кнопке "Options" ("Опции") и щелкните на управляющем элементе "checkbox" "Activate constant bus cycle time / Recalculate constant bus cycle time" ("Активировать постоянные шинные циклы / Пересчитать постоянные шинные циклы"). Вы можете изменить предложенные по умолчанию значения постоянных шинных циклов, но Вы не можете задать для цикла значение, меньшее указанной минимальной величины. С помощью кнопки "Details" ("Подробности") Вы можете увидеть отдельные компоненты постоянных шинных циклов.

Необходимо отметить, что постоянные шинные циклы увеличиваются с ростом числа программаторов, непосредственно подключенных к подсети PROFIBUS, а также с ростом числа интеллектуальных ведомых (slave) DP-устройств, входящих в систему ведущего (master) DP-устройства.

Также, если Вы сконфигурировали SYNC/FREEZE-группы в дополнение к установлению свойства постоянства циклов ("эквидистантности") необходимо отметить, что:

- Для ведомых (slave) DP-устройств, входящих в группу 7, ведущее (master) DP-устройство автоматически инициирует команду SYNC/FREEZE в каждом шинном цикле. Инициация команды в пользовательской программе блокируется.
- Группа 8 используется для организации "эквидистантных" сигналов, при этом блокируется ее использование для ведомых (slave) DP-устройств. Вы не сможете сконфигурировать свойства постоянства циклов ("эквидистантности") сигналов, если Вы имеете уже сконфигурированные ведомые (slave) DP-устройства для группы 8.

Конфигурирование прямого обмена данными (lateral communication - "дополнительные" коммуникации)

Ведущее (master) DP-устройство в системе ведущего (master) DP-устройства имеет исключительные функции управления назначенными ему ведомыми (slave) DP-устройствами. Для специальных (соответствующим образом оснащенных) станций (ведущих [master] или ведомых [slave] устройств, обозначаемых как "приемник" ["receiver"]) возможно отслеживать в подсети PROFIBUS, какие именно входные данные отдельное ведомое (slave) DP-устройство ("передатчик" ["sender"]) посылает "своему" ведущему (master) устройству. Такой прямой обмен данными называется также "lateral communication" ("дополнительные коммуникации"). В принципе все ведомые (slave) DP-устройства (соответствующим образом оснащенные) могут функционировать как "передатчик" ("sender") в прямом обмене данными.

После того как все станции будут подключены в подсети PROFIBUS Вы можете сконфигурировать прямой обмен данными с помощью утилиты конфигурирования оборудования Hardware Configuration в окне свойств "Properties" ведомого (slave) DP-устройства ("receiver" - "приемника"). Откройте станцию-"приемник" и выберите DP-интерфейс, а затем выберите опции меню: *Edit -> Object Properties (Правка -> Свойства объекта).* Вкладка "Configuration" ("Конфигурация") содержит интерфейс передачи между ведомым (slave) DP-устройством и ведущим (master) DP-устройством. Здесь в столбце "Mode" ("Режим") установите рабочий режим DX (Direct Data Exchange - Прямой обмен данными). Далее выберите устройство-"sender" ("передатчик"), чьи сигналы должны отслеживаться в "PROFIBUS DP partner" ("партнер по PROFIBUS") в столбце "Address" ("Адрес").

Кроме того, Вы можете использовать прямой обмен данными между двумя системами ведущих (master) DP-устройств в одной подсети PROFIBUS. Например, ведущее устройство в системе ведущего DP-устройства 1 может отслеживать данные ведомого (slave) DP-устройства в системе ведущего (master) DP-устройства 2 таким же путем.

20.4.3 Системные функции для распределенной периферии (I/O)

Вы можете использовать следующие системные функции для распределенной периферии (I/O):

- SFC 7 DP_PRAL
  Системная функция для инициации прерывания процесса;
- SFC 11 DPSYN_FR
  Системная функция для посылки SYNC/FREEZE-команд;
- SFC 12 D_ACT_DP
  Системная функция для активации/деактивации ведомого (slave) DP-устройства;
- SFC 13 DPNRM_DG
  Системная функция для чтения диагностических данных от стандартного ведомого (slave) DP-устройства;
- SFC 14 DPRD_DAT
  Системная функция для чтения пользовательских данных из ведомого (slave) DP-устройства;
- SFC 15 DPWR_DAT
  Системная функция для записи пользовательских данных в ведомое (slave) DP-устройство.

Далее в таблице 20.7 представлены параметры вышеуказанных системных функций:

Таблица 20.7 Параметры для системных функций для распределенной периферии (I/O)

SFC	Параметр	Объявление	Тип данных	Содержание, описание
7	REQ	INPUT	BOOL	Запрос на инициализацию по REQ ="1"
	IOID	INPUT	BYTE	B#16#54 = input ID (ID входа) B#16#55 = output ID (ID выхода)
	LADDR	INPUT	WORD	Начальный адрес адресной области в памяти для передачи (transfer memory)
	AL_INFO	INPUT	DWORD	Interrupt ID (ID прерывания - передача стартовой информации OB обработки прерывания)
	RET_VAL	OUTPUT	INT	Информация об ошибках
	BUSY	OUTPUT	BOOL	Если BUSY = "1", это означает, что пока нет подтверждения от ведущего (master) DP-устройства
11	REQ	INPUT	BOOL	Запрос на передачу по REQ ="1"
	LADDR	INPUT	WORD	Сконфигурированный диагностический адрес ведущего (master) DP-устройства
	GROUP	INPUT	BYTE	Группа ведомых (slave) DP-устройств (утилита конфигурации)
	MODE	INPUT	BYTE	Команда (см. текст)
	RET_VAL	OUTPUT	INT	Информация об ошибках
	BUSY	OUTPUT	BOOL	Если BUSY = "1", это означает, что задание все еще выполняется.

Таблица 20.7 Параметры для системных функций для распределенной периферии (I/O) (*Продолжение*)

SFC	Параметр	Объявление	Тип данных	Содержание, описание
12	REQ	INPUT	BOOL	Запрос на активацию/деактивацию по REQ ="1"
	MODE	INPUT	BYTE	Режим работы (Function mode): 0 - проверка состояния ведомого (slave) DP-устройства (устройство активировано или деактивировано) 1 - активация ведомого DP-устройства 2 - деактивация ведомого DP-устройства 3 - отмена активации/деактивации ведомого DP-устройства
	LADDR	INPUT	WORD	Диагностический или начальный адрес модуля ведомого (slave) DP-устройства
	RET_VAL	OUTPUT	INT	Результат проверки или информация об ошибках
	BUSY	OUTPUT	BOOL	Если BUSY = "1", это означает, что задание все еще выполняется.
13	REQ	INPUT	BOOL	Запрос на чтение по REQ ="1"
	LADDR	INPUT	WORD	Сконфигурированный диагностический адрес ведомого (slave) DP-устройства
	RET_VAL	OUTPUT	INT	Информация об ошибках
	RECORD	OUTPUT	ANY	Область назначения для считывания диагностических данных
	BUSY	OUTPUT	BOOL	Если BUSY = "1", это означает, что процесс считывания еще не закончен.
14	LADDR	INPUT	WORD	Сконфигурированный начальный адрес (области входов I)
	RET_VAL	OUTPUT	INT	Информация об ошибках
	RECORD	OUTPUT	ANY	Область назначения для считывания пользовательских данных
15	LADDR	INPUT	WORD	Сконфигурированный начальный адрес (области выходов Q)
	RECORD	INPUT	ANY	Область источника пользовательских данных для записи
	RET_VAL	OUTPUT	INT	Информация об ошибках

SFC 7 DP_PRAL
Инициация прерывания процесса

С помощью системной функции SFC 7 DP_PRAL Вы можете инициировать прерывание процесса в ведущем (master) DP-устройстве, связанном с интеллектуальным ведомым (slave) DP-устройством, из пользовательской программы этого ведомого DP-устройства.

В параметре AL_INFO передается идентификатор ID прерывания, определенный пользователем, то есть передается в стартовую информацию организационного блока OB обработки прерывания, вызываемого в ведущем (master) DP-устройстве (переменная OBxx_POINT_ADDR). Запрос на прерывание инициируется при REQ = "1"; параметры RET_VAL и BUSY отображают состояние выполнения задания. Задание завершено, когда завершено выполнение OB обработки прерывания.

"Память для передачи" ("transfer memory") между ведущим (master) DP-устройством и ведомым (slave) DP-устройством может быть разделена на несколько отдельных адресных областей, которые с точки зрения CPU ведущего (master) устройства рассматриваются как отдельные модули. Самый младший адрес отдельного адресного пространства является начальным (базовым) адресом модуля ("module starting address"). Вы можете инициировать прерывание процесса в ведущем (master) DP-устройстве для каждой из этих адресных областей ("виртуальных" модулей).

Адресная область определяется в функции SFC 7 с помощью параметров IOID и LADDR с точки зрения CPU ведомого (slave) устройства (идентификатор I/O (ID) и начальный адрес ведомого модуля). При этом стартовая информация OB обработки прерывания будет содержать адрес "модуля", для которого инициируется прерывание, с точки зрения CPU ведущего (master) устройства.

SFC 11 DPSYN_FR
Посылка SYNC/FREEZE-команд

С помощью системной функции SFC 11 DPSYN_FR Вы можете посылать команды SYNC, UNSYNC, FREEZE и UNFREEZE в SYNC/FREEZE-группы, которые Вы сконфигурировали при помощи утилиты конфигурирования оборудования Hardware Configuration. Операция посылки команды (SEND) инициализируется при значении параметра REQ = "1" и завершается при значении параметра BUSY = "0".

В параметре GROUP знак каждой группы занимает один бит (при этом бит 0 соответствует группе 1, бит 7 соответствует группе 8). Команды в параметре MODE также организованы побитно:

- UNFREEZE, если бит 2 = "1";
- FREEZE, если бит 3 = "1";
- UNSYNC, если бит 4 = "1";
- SYNC, если бит 5 = "1".

Таким образом, режимы SYNC и FREEZE для ведомых (slave) DP-устройств сначала выключаются (согласно очередности битов). Входы ведомых (slave) DP-устройств сканируются последовательно ведущим (master) DP-устройством, а выходы ведомых (slave) DP-устройств модифицируются; ведомые (slave) DP-устройства немедленно передают принятые выходные сигналы на выходные оконечные устройства ("терминалы").

Если необходимо "заморозить" ("freeze") входные сигналы нескольких ведомых (slave) DP-устройств в определенное время, Вы можете послать команду FREEZE для соответствующей группы. При этом ведущее (master) DP-устройство будет последовательно считывать входные сигналы, имеющие те состояния, которые были на входах в момент прихода команды FREEZE (то есть в момент их "замораживания"). Эти входные сигналы сохраняют свое значение до того момента, когда пользователь пошлет новую команду FREEZE, в соответствии с которой ведомые (slave) DP-устройства вновь считают и будут удерживать считанные значения сигналов (новые текущие значения) или до того момента, когда пользователь вновь переключит ведомые (slave) DP-устройства в "нормальный" режим командой UNFREEZE.

Если необходимо выдавать выходные сигналы нескольких ведомых (slave) DP-устройств синхронно в определенное время, то сначала Вы должны послать команду SYNC для соответствующей группы. При этом адресованные ведомые (slave) DP-устройства зафиксируют выходные сигналы на выходных оконечных устройствах ("терминалах"). Теперь Вы можете переслать требуемые состояния сигналов для ведомых (slave) DP-устройств. Вслед за передачей этих сигналов Вы вновь должны послать команду SYNC, что приведет к тому, что ведомые DP-устройства одновременно передадут принятые выходные сигналы на выходные оконечные устройства. Эти выходные сигналы будут сохранены ведомыми (slave) DP-устройствами до того момента, когда пользователь пошлет новую команду SYNC, или до того момента, когда пользователь вновь переключит ведомые (slave) DP-устройства в "нормальный" режим командой UNSYNC.

SFC 12 D_ACT_DP
Активация и деактивация ведомого (slave) DP-устройства

С помощью системной функции SFC 12 D_ACT_DP Вы можете деактивировать сконфигурированное (и существующее) ведомое (slave) DP-устройство так, что это устройство будет более недоступно для ведущего (master) DP-устройства. Выходные оконечные устройства ("терминалы") такого деактивированного ведомого (slave) DP-устройства после этого будут иметь "нулевые" состояния сигналов (=0) или "значения подстановки" (или "значения замены" - "substitute value").

Деактивированное ведомое (slave) DP-устройство может быть демонтировано из шины, что не повлечет за собой выдачи сообщения об ошибке; при этом не будет ни сообщения об отказе, ни сообщения об отсутствии модуля. Вызовы организационных блоков обработки асинхронных ошибок OB 85 (ошибки выполнения программы при размещении пользовательских данных деактивированного ведомого [slave] DP-устройства в области автоматически обновляемого образа процесса) и OB 86 (отказ станции) блокируются. После деактивации ведомого (slave) DP-устройства пользователь в дальнейшем времени не должен пытаться получить доступ к этому ведомому устройству, иначе возникнут ошибки ввода/вывода ("I/O access errors").

С помощью системной функции SFC 12 D_ACT_DP Вы можете вновь активировать ранее деактивированное ведомое (slave) DP-устройство. Ведомое (slave) DP-устройство конфигурируется и параметризируется ведущим (master) DP-устройством таким же путем, как при восстановлении ("restore") станции. При активировании ранее деактивированного ведомого (slave) DP-устройства организационные блоки обработки ошибок OB 85 и OB 86 не запускаются. Если параметр BUSY имеет состояние сигнала "0" после активации ведомого (slave) DP-устройства, то это устройство может быть доступно из пользовательской программы.

SFC 13 DPNRM_DG
Считывание диагностических данных

С помощью системной функции SFC 13 DPNRM_DG Вы можете считывать диагностические данные из ведомого (slave) DP-устройства. Процедура считывания инициируется, когда параметр запроса REQ = "1", а завершается при возврате параметра BUSY = "0".

Значение функции RET_VAL при этом содержит число считанных байтов. В зависимости от ведомого (slave) DP-устройства диагностические данные могут содержать от 6 до 240 байтов. Если объем диагностических данных превышает 240 байтов, то первые 240 байтов пересылаются при считывании и при этом устанавливается соответствующий бит переполнения в диагностических данных.

В параметре RECORD указывается область, в которой сохраняются считанные данные. В качестве фактических параметров допускаются переменные типов ARRAY и STRUCT или указатели ANY типа BYTE (например, P#DBzDBXy.x BYTE nnn).

SFC 14 DPRD_DAT
Считывание пользовательских данных

С помощью системной функции SFC 14 DPRD_DAT Вы можете считывать пользовательские данные из ведомого (slave) DP-устройства с гарантией консистентности для объемов данных, составляющих 3 байта или больше 4-х байтов. При задании параметров ведомого (slave) DP-устройства Вы должны определить объем консистентных данных.

В параметре LADDR содержится начальный адрес модуля ведомого (slave) DP-устройства (область входов).

В параметре RECORD указывается область, в которой сохраняются считанные данные. В качестве фактических параметров допускаются переменные типов ARRAY и STRUCT или указатели ANY типа BYTE (например, P#DBzDBXy.x BYTE nnn).

SFC 15 DPWR_DAT
Запись пользовательских данных

С помощью системной функции SFC 15 DPWR_DAT Вы можете записывать пользовательские данные в ведомое (slave) DP-устройство с гарантией консистентности для объемов данных, составляющих 3 байта или больше 4-х байтов. При задании параметров ведомого (slave) DP-устройства Вы должны определить объем консистентных данных.

В параметре LADDR содержится начальный адрес модуля ведомого (slave) DP-устройства (область входов).

В параметре RECORD указывается исходная область, в которой хранятся считываемые данные. В качестве фактических параметров допускаются переменные типов ARRAY и STRUCT или указатели ANY типа BYTE (например, P#DBzDBXy.x BYTE nnn).

20.5 Коммуникации посредством глобальных данных

20.5.1 Основы

Коммуникации посредством глобальных данных (GD-коммуникации) - это особая система связи (коммуникационная служба), встроенная в операционную систему CPU, которая используется для обмена

небольшими объемами некритичных ко времени данных с использованием MPI-шины. К данным, которые могут переноситься с помощью глобальных данных, относятся:

- данные входов и выходов (отображения процесса);
- данные меркеров;
- данные блоков DB;
- данные таймеров и счетчиков как данные, которые необходимо переслать.

Для использования GD-коммуникаций необходимо обеспечение ряда требований: все CPU должны быть включены в сеть посредством MPI-интерфейса или подключены к K-шине как в монтажной стойке S7-400, все CPU должны существовать в одном проекте STEP 7, чтобы можно было сконфигурировать GD-коммуникации.

Для коммуникаций посредством глобальных данных не обязательно использовать операционную систему: существуют системные функции для GD-коммуникаций в S7-400.

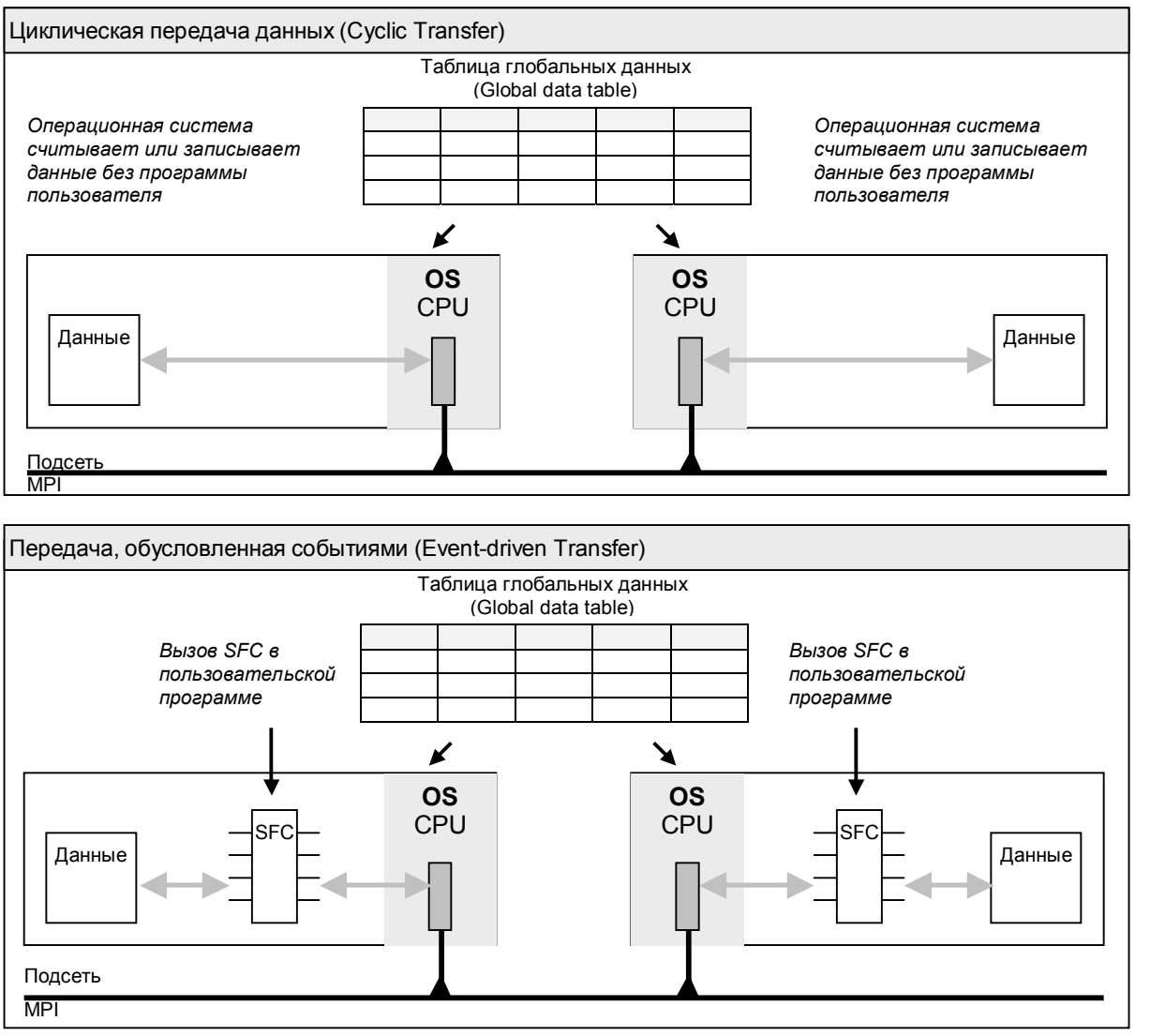


Рис. 20.12 Коммуникации посредством глобальных данных

Необходимо отметить, что CPU-приемник не подтверждает факт приема глобальных данных. Следовательно, CPU-передатчик не получает никакого ответного сообщения, из которого следовало бы, что "приемник" получил данные, и если получил, то какие именно. Тем не менее, Вы имеете возможность мониторинга состояния связи между двумя CPU, так же как и состояния всех GD-групп CPU.

Посылка и прием глобальных данных зависят от параметра, известного как "частота сканирования" ("scan rate"). Этот параметр определяется количеством циклов сканирования пользовательской программы, после выполнения которого CPU осуществляет передачу или прием данных. Каждый раз процессы передачи данных и их приема между "передатчиком" и "приемником" происходят синхронно в определенной точке цикла, то есть в некоторый момент по окончании сканирования циклически выполняемой программы и перед новым циклом ее выполнения (как, например, при обновлении образа процесса).

Обмен данными происходит в форме пересылки пакетов данных (GD packet) между CPU, сгруппированными в GD-группы (GD circle).

GD-группа (GD circle)

CPU, которые обмениваются общими (shared) GD-пакетами, формируют GD-группы (GD circles). GD-группы могут быть следующих видов:

- Одностороннее соединение CPU, который посылает GD-пакеты данных в адрес нескольких других CPU, которые принимают эти пакеты данных.
- Двустороннее соединение между двумя CPU, при котором каждый из этих двух CPU может посылать GD-пакет данных другому.
- Двустороннее соединение между тремя CPU, при котором каждый из этих трех CPU может послать один GD-пакет данных двум другим CPU (только для S7-400 CPU).

В одной GD-группе могут обмениваться данными друг с другом до 15 единиц CPU. Один CPU может при этом принадлежать к нескольким GD-группам.

В таблице 20.8 Вы можете получить информацию об исходных данных для GD-коммуникаций для отдельных типов CPU.

GD-пакет (GD packet)

GD-пакет (GD packet) включает в себя заголовок (header) пакета и один или несколько элементов глобальных данных (GD-элементов):

- Заголовок пакета (8 байтов).
- ID 1-го GD-элемента (2 байта).
- Пользовательские данные 1-го GD-элемента (x байтов).
- ID 2-го GD-элемента (2 байта).
- Пользовательские данные 2-го GD-элемента (x байтов).
- и т.д.

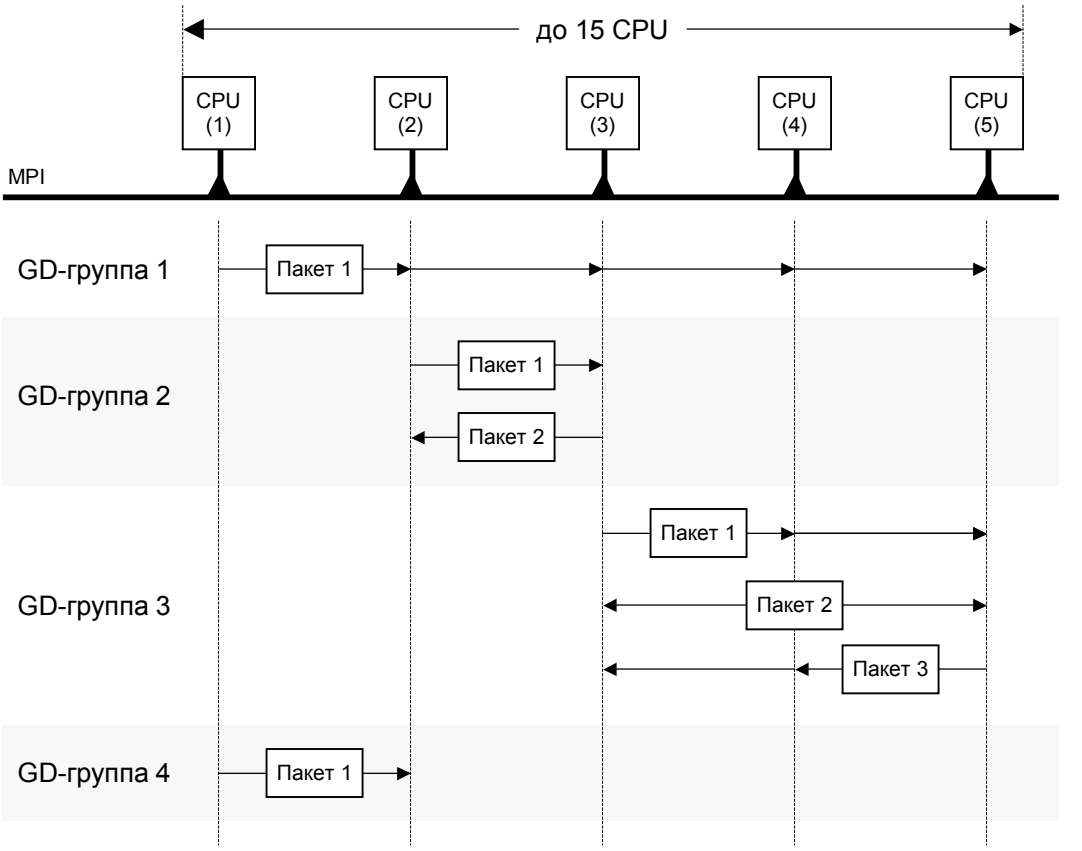


Рис. 20.13 Пример организации GD-групп

Каждый GD-элемент состоит из 2 байтов описания и данных, передаваемых по сети (actual net data). 3 байта требуются для передачи байта данных (байта меркеров), 4 байта требуются для передачи слова данных (слова меркеров), 6 байт требуются для передачи двойного слова данных (двойного слова меркеров). Булева переменная занимает 1 байт данных, передаваемых по сети (net data); следовательно, она занимает такой же объем как и переменная длиной один байт. Значения таймеров и счетчиков, имеющие размер 2 байта каждый, в GD-пакете занимают по 4 байта.

GD-элемент может также включать в себя даные из адресной области. Например, запись MB 0:15 означает область байтов от MB 0 до MB 15, а DB20.DBW14:8 представляет область данных, размещенных в блоке DB20, которая начинается со слова данных DBW14 и содержит 8 слов данных.

Максимальный размер GD-пакета составляет 32 байта для S7-300 и 64 байта для S7-400. Максимальное число байтов данных, передаваемых по сети посредством GD-пакета, составляет до 22 байтов для S7-300 и до 54 байтов для S7-400.

Консистентность данных

Консистентность данных обеспечивается для одного GD-элемента. В таблице 20.8 приведены области, определенные для отдельных CPU для

случая, когда GD-элемент соответствует переменной данного CPU.

Если GD-элемент больше, чем объем данных, для которого гарантируется консистентность, то консистентные данные соответствующего размера располагаются, начиная с первого байта.

Таблица 20.8 Ресурсы CPU для связи посредством глобальных данных

GD-ресурсы максимальное число:	CPU 312 CPU 313 CPU 314	CPU 315 CPU 316	CPU 318	CPU 412 CPU 413 CPU 414	CPU 416 CPU 417
GD-группа на 1 CPU	4	4	8	8	16
принимаемых GD-пакетов на 1 CPU	4	4	16	16	32
принимаемых GD-пакетов на 1 GD-группу	1	1	2	2	2
посылаемых GD-пакетов на 1 CPU	4	4	8	8	16
посылаемых GD-пакетов на 1 GD-группу	1	1	1	1	1
максимальный размер GD-пакета	32 байта	32 байта	64 байта	64 байта	64 байта
максимальный размер консистентных данных	8 байта	8 байта	32 байта	16 байта	32 байта

20.5.2 Конфигурирование GD-коммуникаций

Требования

Для того, чтобы использовать возможности GD-коммуникаций у Вас должен быть создан проект, в котором должна быть доступна подсеть MPI и должны быть сконфигурированы S7-станции. В каждой станции должен быть по крайней мере один CPU. Используя кнопку "Properties" ("Свойства") в MPI-интерфейсе на вкладке "General" ("Общие") окна свойств CPU (можно открыть, дважды щелкнув на строке CPU в окне конфигурации оборудования или в строке, содержащей субмодуль MPI-интерфейса), Вы можете установить MPI-адрес и выбрать подсеть MPI для подключаемого CPU.

Таблица глобальных данных (Global data table)

Конфигурирование GD-коммуникаций производится путем заполнения таблицы. Вы можете вызвать пустую таблицу посредством выбора значка подсети MPI в Simatic Manager или при конфигурировании сети (Network Configuration) с помощью опций меню: *Options -> Define Global Data (Опции -> Определить глобальные данные).* Выберите в таблице столбец, а затем опции меню: *Edit -> CPU (Правка -> CPU).* В левой половине открывшегося окна для выбора проекта выберите станцию, затем в правой половине окна выберите CPU. Выбор CPU для таблицы глобальных данных подтверждается нажатием клавиши "OK".

Выполните те же действия в отношении остальных CPU, участвующих в обмене посредством глобальных данных. Таблица глобальных данных может содержать до 15 столбцов CPU.

Для конфигурирования обмена данными между CPU посредством GD-коммуникаций выберите первую строку под CPU-передатчиком и задайте адрес, значение которого должно передаваться (завершить действие клавишей Enter).

С помощью опций меню: *Edit -> Sender (Правка -> Передающий CPU)* Вы определяете данное значение, как значение, которое должно быть передано, обозначаемое символом-префиксом ">" и выделяемое тенью. В той же строке под CPU-приемником Вы определяете адрес, в который должно быть принято данное значение (свойство "Receiver" ["Приемник"] устанавливается по умолчанию). Функции таймера и счетчика Вы можете использовать только как "передатчики"; "приемником" данных и от функции таймера, и от функции счетчика должен служить адрес области, имеющей размер слова данных.

Строка может содержать несколько "приемников", но при этом может иметь только один "передатчик" (см. таблицу 20.9). После заполнения такой таблицы Вы должны ее скомпилировать. Для этого выберите опции меню: *GD Table -> Compile (GD-таблица -> Компилировать).*

Таблица 20.9 Пример GD-таблицы (GD Table) с указанием состояния (Status) и частоты сканирования (Scan Rates)

GD-иден-тификатор	Станция 417 \ CPU 417 (3)	Станция 417 \ CPU 414 (4)	Станция 416 \ CPU 416 (5)	Станция 315 ведомая (slave) CPU 315 (7)	Станция 314CP CPU 314 (10)
GST	MD100	MD100	MD100	DB10.DBD200	DB10.DBD200
GDS 1.1	DB9.DBD0		MD92	DB10.DBD204	DB10.DBD204
SR 1.1	44	0	44	8	8
GD 1.1.1	>DB9.DBW10		MW90	DB10.DBW208	DB10.DBW208
GDS 2.1	MD96	MD96			
SR 2.1	44	23	0	0	0
GD 2.1.1	>Z10:10	DB3.DBW20:10			
GDS 3.1			MD96		
SR 3.1	0	0	44	8	8
GD 3.1.1			>MW98	DB10.DBW220	DB10.DBW210

После компиляции (фаза 1) созданных системных данных достаточно для обеспечения обмена посредством глобальных данных. Если Вы также конфигурируете состояние GD-коммуникаций (GD-status) и частоту сканирования (scan rates) Вы снова должны выполнить

компилирование GD-таблицы.

Идентификатор глобальных данных (GD ID)

После выполнения без ошибок процесса компилирования GD-таблицы STEP 7 заполняет столбец "GD ID". Этот столбец показывает пользователю структуру передаваемых данных в GD-группах, GD-пакетах и GD-элементах. Например, GD ID "GD 2.1.3" соответствует GD-группе 2, GD-пакету 1 и GD-элементу 3. В GD-таблице в столбце каждого CPU Вы можете также найти назначенные ресурсы (число GD-групп) для соответствующего CPU.

Состояние GD-коммуникаций (GD status)

После выполнения процесса компилирования GD-таблицы Вы можете задать адреса для формирования состояния (статуса) GD-коммуникаций. Это выполняется с помощью опций меню: *View -> GD Status (Вид -> состояние GD-коммуникаций).* Параметр общего состояния (статуса) (GST) определяет все коммуникационные соединения в таблице. Параметр состояния (статуса) (GDS) показывает состояние коммуникационных соединений для передачи GD-пакетов. В каждом случае параметр использует двойное слово данных.

Частота сканирования (Scan rate)

Осуществление функций связи посредством глобальных данных требует значительной доли времени на выполнение соответствующих операций в общем объеме работы операционной системы CPU, а также требует определенных затрат времени на передачу по MPI-шине. Для того, чтобы свести такие затраты времени ("коммуникационная нагрузка") к минимуму, можно соответствующим образом настраивать параметр "Scan rate" ("Частота сканирования"). Этот параметр определяет количество циклов сканирования пользовательской программы, после выполнения которых CPU осуществляет передачу или прием данных в форме GD-пакета (GD-packet).

Так как с учетом применения параметра "Scan rate" ("Частота сканирования") обновление данных происходит не в каждом цикле сканирования программы, пользователю рекомендуется избегать использования GD-коммуникаций для пересылки данных, критичных ко времени.

После первого выполнения процесса компилирования GD-таблицы (если не произошло ошибок) Вы можете самостоятельно определить параметры "Scan rate" ("SR" - "частота сканирования") для каждого GD-пакета и для каждого CPU. Это выполняется с помощью опций меню: *View -> Scan Rates (Вид -> Частота сканирования)*. Стандартно параметр "Scan rate" ("Частота сканирования") устанавливается таким, что при "пустом" (без пользовательской программы) CPU посылка и прием GD-пакета осуществляется приблизительно каждые 10 миллисекунд. Если после этого загрузить в CPU пользовательскую программу, интервал между соседними посылками или приемами GD-пакета возрастает.

Пользователь может задавать параметр "Scan rate" ("Частота сканирования") в диапазоне от 1 до 255.

Необходимо отметить, что при уменьшении значения "Scan rate" ("Частота

сканирования") "коммуникационная нагрузка" CPU возрастает.

Для того, чтобы поддерживать коммуникационную нагрузку CPU в приемлемых рамках, необходимо устанавливать такие значения параметра "Scan rate" ("Частота сканирования") для CPU-передатчика, чтобы произведение этого значения на величину времени сканирования для S7-300 было больше 60 мс, а для S7-400 было больше 10 мс. Соответственно для CPU-приемника необходимо устанавливать такие частоты сканирования, чтобы произведение их значений на величину времени сканирования для S7-300 было меньше 60 мс, а для S7-400 было меньше 10 мс. Выполнение указанных рекомендаций позволяет избежать потерь отдельных GD-пакетов.

Установив нулевое ("0") значение параметра "Scan rate" ("Частота сканирования"), Вы можете выключть обмен данными посредством соответствующего GD-пакета, если необходимо использовать только обусловленный событиями обмен данными с использованием SFC-функций.

После конфигурирования параметров состояния (статуса) GD и параметров "Scan rate" ("Частота сканирования") Вы должны снова (во второй раз) скомпилировать GD-таблицу. Затем STEP 7 введет скомпилированные данные в объект *System data* (системные данные). Функции связи посредством глобальных данных становятся доступными после того, как GD-таблица пересылается в подключенные CPU с помощью опций меню: *PLC -> Download (PLC -> Загрузить)*.

GD-коммуникации становятся доступными также после того, как будет переслан объект *System data* (системные данные), который содержит все установки для оборудования и установки для параметров.

20.5.3 Системные функции для GD-коммуникаций

В системах S7-400 Вы можете из программы пользователя управлять обменом данными посредством GD-коммуникаций. В дополнение к циклической передаче глобальных данных или вместо циклической передачи глобальных данных Вы можете организовать передачу или прием GD-пакета посредством следующих системных функций:

- SFC 60 GD_SND
  функция пересылки GD-пакета,
- SFC 61 GD_RCV
  функция приема GD-пакета.

Список параметров для этих системных функций представлен в таблице 20.10. Необходимым условием для использования этих системных функций является наличие сконфигурированной таблицы глобальных данных. После компилирования этой таблицы STEP 7 в столбце "GD Identifier" (Идентификатор GD-коммуникаций) показывает пользователю структуру передаваемых данных - номера GD-групп и GD-пакетов, которые необходимо учитывать при назначении параметров функций.

Функция пересылки GD-пакета SFC 60 GD_SND вводит GD-пакет в системную память CPU и инициализирует его пересылку; функция приема

GD-пакета SFC 61 GD_RCV позволяет выбрать GD-пакет из системной памяти CPU. Если параметр "Scan rate" ("Частота сканирования") имеет ненулевое (больше, чем "0") значение в GD-таблице, то циклическая передача данного GD-пакета данных также имеет место.

Если необходимо обеспечивать консистентность данных для GD-пакета в целом при пересылке с использованием системных функций SFC 60 и SFC 61, то пользователь должен заблокировать (disable) или отложить на время (delay) выполнение прерываний с высоким приоритетом, как и обработку асинхронных ошибок, как на "передающей" (Send) так и на "принимающей" (Receive) стороне, во время выполнения функций SFC 60 и SFC 61.

Системные функции SFC 60 и SFC 61 не обязательно вызывать парами; и "смешанные" операции также возможны. Например, Вы можете использовать функцию SFC 60 GD_SND для обеспечения пересылки GD-пакетов, обусловленной событиями с последующим циклическим их считыванием (Receive).

Таблица 20.10 Параметры для SFC для GD-коммуникаций.

Параметр	Используется в функции SFC		Объявление	Тип	Содержание, описание
CIRCLE_ID	60	61	INPUT	BYTE	Номер GD-группы
BLOCK_ID	60	61	INPUT	BYTE	Номер GD-пакета, который должен быть послан или принят
RET_VAL	60	61	OUTPUT	INT	Информация об ошибках

20.6 SFC-коммуникации

20.6.1 Внутристанционные (Station-Internal) SFC-коммуникации

Основы

При использовании "внутристанционных" (Station-Internal) SFC-коммуникаций Вы можете организовать обмен данными между программируемыми модулями внутри SIMATIC-станции. В этом случае в качестве коммуникационных функций используются SFC-функции операционной системы CPU. Если это необходимо, данные SFC-функции могут устанавливать коммуникационные соединения самостоятельно. Поэтому такие "внутристанционные" соединения не конфигурируются в таблице соединений ("Communication via non-configured connections", basic communication - "Функции связи посредством несконфигурированных соединений", базовые функции связи).

Внутристанционные (Station-Internal) SFC-коммуникации могут использоваться параллельно с циклическим, обусловленным событиями, обменом данными посредством PROFIBUS-DP между ведущим (master) CPU и ведомым (slave) CPU (см. рис. 20.14).

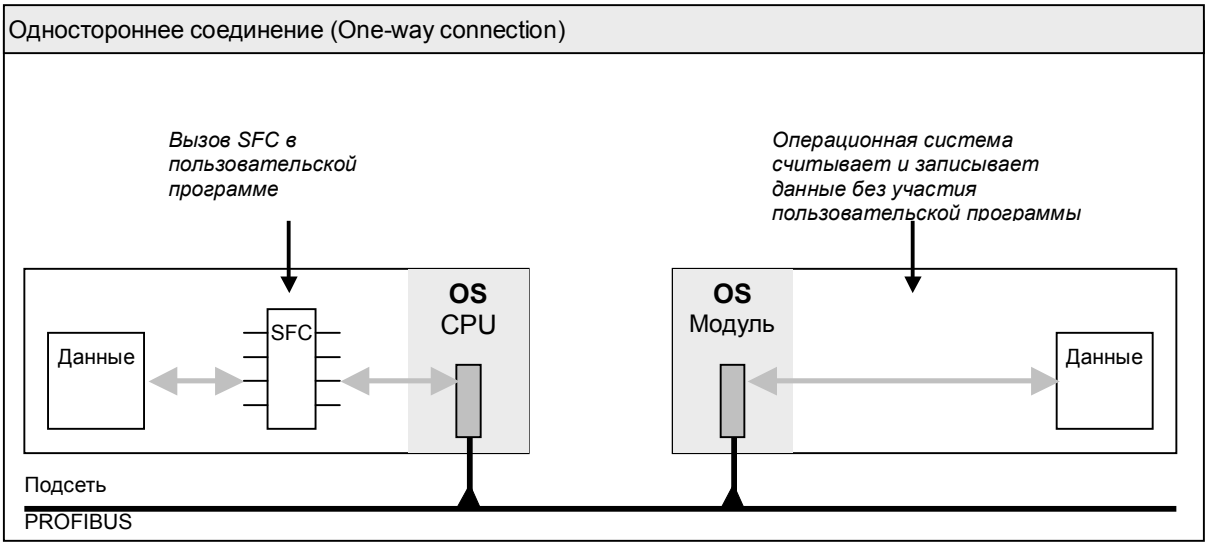


Рис. 20.14 Внутристанционные (Station-Internal) SFC-коммуникации

Адресация узлов, соединений

Идентификация узла определяется его I/O-адресом: в параметре LADDR Вы задаете начальный адрес модуля, а в параметре IOID определяете, будет ли данный адрес в области входов (input area) или в области выходов (output area).

Рассматриваемые системные функции устанавливают требуемые коммуникационные соединения динамически. Они разрывают вновь соединение после окончания выполнения коммуникационного задания (в соответствии с программой). Если соединение не может быть установлено из-за отсутствия требуемых ресурсов в устройстве, передающем данные, или в устройстве, принимающем данные, поступает сигнал в виде сообщения: "Temporary lack of resources" ("Временная нехватка ресурсов"). В этом случае попытка выполнить передачу данных должна быть предпринята некоторое время спустя. При этом в каждом направлении может быть установлено только одно соединение между двумя коммуникационными партнерами.

Вы можете использовать одну системную функцию для разных коммуникационных соединений, изменяя параметры блока во время выполнения программы. Любая SFC не может вызвать прерывание самой себя. Отдельные части программы, в которых вызывается одна из рассматриваемых функций, могут быть изменены только в режиме STOP; после этого выполняется полный перезапуск.

Пользовательские данные, консистентность данных

SFC-функции для обмена данными между программируемыми модулями внутри SIMATIC-станции могут использоваться для передачи до 76 байтов пользовательских данных. Независимо от направления передачи операционная система CPU разбивает пользовательские данные на блоки, которые обладают свойством внутренней консистентности ("consistent within themselves"). В S7-300 эти блоки имеют длину 8 байтов;

в CPU 412/413 эти блоки имеют длину 16 байтов; в CPU 414/416 эти блоки имеют длину 32 байта. При обмене данными между двумя CPU размер блока, характерный для "пассивного" CPU, имеет решающее значение с точки зрения консистентности передаваемых данных.

Конфигурирование "внутристанционных" (Station-Internal) SFC-коммуникаций

Внутристанционные (Station-Internal) SFC-коммуникации являются особым видом связи, для которой не требуется конфигурирования, так как соединения для передачи данных устанавливаются динамически. Вы можете либо использовать существующую подсеть PROFIBUS, либо создать новую в SIMATIC Manager (для этого выберите объект *Proect,* после чего выберите опции меню: *Insert -> Subnetwork -> PROFIBUS [Вставка -> Подсеть -> PROFIBUS]*) или при помощи утилиты конфигурирования сети Network Configuration (см. раздел 2.4 "Конфигурирование сети").

Пример: допустим у Вас есть сконфигурированная распределенная периферия (I/O) с CPU 315-2DP в качестве ведущего (master) устройства и Вы используете другой CPU 315-2DP в качестве интеллектуального ведомого (slave) устройства. В данной ситуации Вы можете использовать внутристанционные (Station-Internal) SFC-коммуникации для обоих контроллеров как для чтения, так и для записи данных.

20.6.2 Системные функции для обмена данными внутри станции

Для обмена данными между двумя CPU внутри одной станции используются следующие системные функции:

- SFC 72 I_GET
  функция для считывания данных (Read data),
- SFC 73 I_PUT
  функция для записи данных (Write data),
- SFC 74 I_ABORT
  функция для разрыва соединения (Disconnect).

Список параметров для данных системных функций представлен в таблице 20.11.

SFC 72 I_GET
Считывание данных (Read data)

Выполнение данной функции начинается при следующих значениях параметров REQ и BUSY: REQ = "1" и BUSY = "0" (при первом вызове). Пока функция выполняется, параметр BUSY установлен (BUSY = "1"). При этом любые изменения параметра REQ не влияют на процесс выполнения задания. Как только выполнение функции завершается, параметр BUSY сбрасывается (BUSY = "0"). Если при этом параметр REQ все еще установлен (REQ = "1"), то сразу же вновь начинает выполняться данная функция.

Если запущена процедура считывания данных, операционная система в CPU партнера собирает и посылает запрашиваемые данные. При вызове SFC принятые данные пересылаются в область назначения. Параметр RET_VAL при этом содержит число переданных байтов.

Если параметр CONT сброшен (CONT = "0"), то коммуникационное соединение разорвано, если параметр CONT установлен (CONT = "1"), то коммуникационное соединение создано. Данные также могут быть считаны, когда коммуникационный партнер находится в режиме STOP.

Параметры RD и VAR_ADDR описывают область, из которой пересылаемые данные должны быть считаны, или в которую принимаемые данные должны быть записаны. Фактическими параметрами функции могут быть адреса, переменные или области данных, адресованные указателем ANY. Передаваемые и принимаемые данные при этом не проверяются на корректность (на соответствие типу данных).

Таблица 20.11 Параметры для SFC для GD-коммуникаций.

Параметр	Используется в функции SFC			Объяв-ление	Тип	Содержание, описание
REQ	72	73	74	INPUT	BOOL	При REQ="1" функция запускается на выполнение
CONT	72	73	-	INPUT	BOOL	При CONT="1" соединение остается неразорванным после завершения выполнения функции
IOID	72	73	74	INPUT	BYTE	B#16#54 = Input area (область входов) B#16#55 = Output area (область выходов)
LADDR	72	73	74	INPUT	WORD	Начальный адрес модуля
VAR_ADDR	72	73	-	INPUT	ANY	Область данных в CPU партнера
SD	-	73	-	INPUT	ANY	Область данных в "своем" CPU, которая содержит передаваемые данные
RET_VAL	72	73	74	OUTPUT	INT	Информация об ошибках
BUSY	72	73	74	OUTPUT	BOOL	При BUSY ="1" функция все еще выполняется
RD	72	-	-	OUTPUT	ANY	Область данных в "своем" CPU, которая получает принятые данные

SFC 73 I_PUT
Запись данных (Write data)

Выполнение данной функции начинается при следующих значениях параметров REQ и BUSY: REQ = "1" и BUSY = "0" (при первом вызове). Пока функция выполняется, параметр BUSY установлен (BUSY = "1"). При этом любые изменения параметра REQ не влияют на процесс выполнения задания. Как только выполнение функции завершается, параметр BUSY сбрасывается (BUSY = "0"). Если при этом параметр REQ

все еще установлен (REQ = "1"), то сразу же вновь начинает выполняться данная функция.

Если запущена процедура записывания данных, операционная система при первом вызове передает все данные из области CPU-источника во внутренний буфер и посылает их коммуникационному партнеру. CPU-приемник записывает принятые данные в область VAR_ADDR. После этого параметр BUSY сбрасывается (BUSY = "0"). Данные также могут быть записаны, когда коммуникационный партнер находится в режиме STOP.

Параметры SD и VAR_ADDR описывают область, из которой пересылаемые данные должны быть считаны, или в которую принимаемые данные должны быть записаны. Фактическими параметрами функции могут быть адреса, переменные или области данных, адресованные указателем ANY. Передаваемые и принимаемые данные при этом не проверяются на корректность (на соответствие типу данных).

SFC 74 I_ABORT
Разрыв соединения (Disconnect)

Выполнение данной функции инициируется при значении параметра REQ, равном "1", - при этом происходит разрывание связи с определенным коммуникационным партнером. С помощью системной функции SFC 74 I_ABORT Вы можете разорвать только те связи, которые установлены в рассматриваемой станции с помощью системных функций SFC 72 I_GET и SFC 73 I_PUT. Пока функция выполняется, параметр BUSY установлен (BUSY = "1"). При этом любые изменения параметра REQ не влияют на процесс выполнения функции. Как только выполнение функции завершается, параметр BUSY сбрасывается (BUSY = "0"). Если при этом параметр REQ все еще установлен (REQ = "1"), то сразу же вновь начинает выполняться данная функция.

20.6.3 Внестанционные (Station-External) SFC-коммуникации

Основы

При использовании "внестанционных" (Station-External) SFC-коммуникаций Вы можете организовать обусловленный событиями обмен данными между разными SIMATIC-станциями. Станции должны быть связаны друг с другом посредством MPI-подсети. В этом случае в качестве коммуникационных функций используются SFC-функции операционной системы CPU. Если это необходимо, данные SFC-функции могут устанавливать коммуникационные соединения самостоятельно. Поэтому такие "внестанционные" соединения не конфигурируются в таблице соединений ("Communication via non-configured connections", basic communication - "Функции связи посредством несконфигурированных соединений", базовые функции связи).

Внестанционные (Station-External) SFC-коммуникации позволяют осуществлять обусловленный событиями обмен данными параллельно, например, с циклическим обменом посредством глобальных данных (см. рис. 20.15).

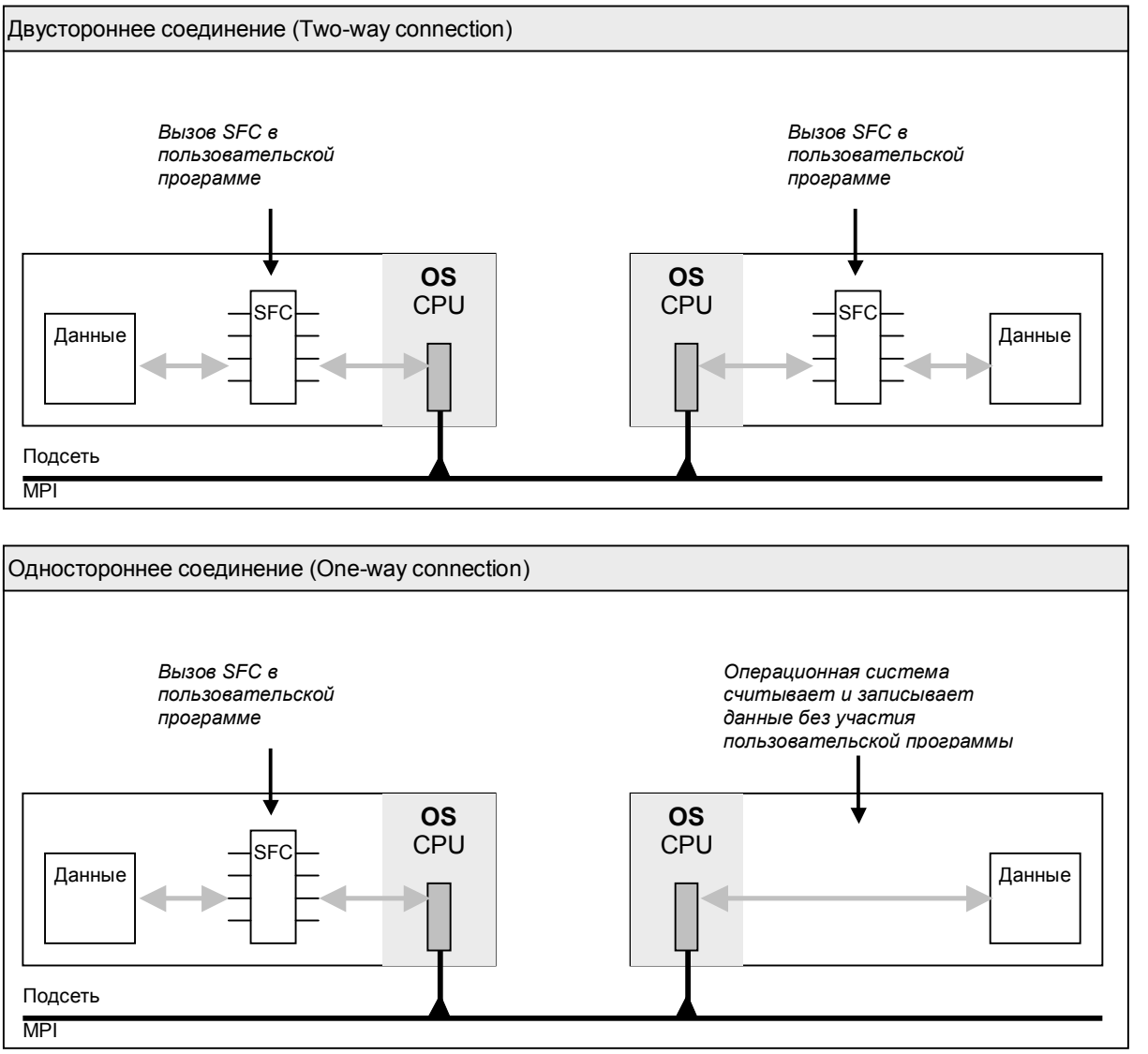


Рис. 20.15 Внестанционные (Station-External) SFC-коммуникации

Адресация узлов, соединений

Рассматриваемые системные функции обеспечивают связь между узлами, адреса которых принадлежат одной MPI-подсети. Идентификация узла определяется его MPI-адресом (параметр DEST_ID).

SFC-функции для обмена данными между разными SIMATIC-станциями одной MPI-подсети устанавливают требуемые коммуникационные связи динамически и, если требуется, они могут разорвать связь после окончания выполнения коммуникационного задания. Если соединение не может быть установлено из-за отсутствия требуемых ресурсов в устройстве, передающем данные, или в устройстве, принимающем данные, то поступает сигнал в виде сообщения: "Temporary lack of resources" ("Временная нехватка ресурсов"). В этом случае попытка выполнить передачу данных должна быть предпринята некоторое время

спустя. При этом в каждом направлении может быть установлено только одно соединение между двумя коммуникационными партнерами.

При переходе системы из режима RUN в режим STOP все активные соединения (все системные функции, кроме X_RECV) разрываются.

Вы можете использовать любую из этих системных функций для разных коммуникационных соединений, изменяя параметры блока во время выполнения программы. Любая SFC не может вызвать прерывание самой себя. Отдельные части программы, в которых вызывается одна из рассматриваемых функций, могут быть изменены только в режиме STOP; после этого выполняется полный перезапуск.

Пользовательские данные, консистентность данных

SFC-функции для обмена данными между разными SIMATIC-станциями могут использоваться для передачи до 76 (максимум) байтов пользовательских данных. Независимо от направления передачи операционная система CPU разбивает пользовательские данные на блоки, которые обладают свойством внутренней консистентности ("consistent within themselves"). В S7-300 эти блоки имеют длину 8 байтов; в CPU 412/413 эти блоки имеют длину 16 байтов; в CPU 414/416 эти блоки имеют длину 32 байта.

При обмене данными между двумя CPU с помощью функций X_GET или X_PUT размер блока, характерный для "пассивного" CPU, имеет решающее значение с точки зрения консистентности передаваемых данных. При обмене данными с помощью SEND/RECEIVE обеспечивается консистентность всех данных.

Конфигурирование "внестанционных" (Station-External) SFC-коммуникаций

Внестанционные (Station-External) SFC-коммуникации являются особым видом связи, для которой не требуется конфигурирования, так как соединения для передачи данных устанавливаются динамически. Вы можете либо использовать существующую подсеть MPI, либо создать новую.

Пример: допустим у Вас есть секционированная монтажная стойка S7-400, в которой в каждой секции установлен один CPU 416; кроме того к одной из секций S7-400 с помощью MPI-кабеля подключена станция S7-300 с CPU 314. Все три CPU сконфигурированы с помощью утилиты конфигурирования оборудования Hardware Configuration, например, как устройства, связанные подсетью MPI. В данной ситуации Вы можете использовать внестанционные (Station-External) SFC-коммуникации для всех трех контроллеров для обмена данными.

20.6.4 Системные функции для обмена данными между станциями ("внестанционные" SFC)

Для обмена данными между устройствами-партнерами в разных станциях одной подсети используются следующие системные функции:

- SFC 65 X_SEND
  функция для пересылки данных (Send data),
- SFC 66 X_RCV
  функция для приема данных (Receive data),
- SFC 67 X_GET
  функция для считывания данных (Read data),
- SFC 68 X_PUT
  функция для записи данных (Write data),
- SFC 69 X_ABORT
  функция для разрыва соединения (Disconnect).

Список параметров для данных системных функций представлен в таблице 20.12.

Таблица 20.12 Параметры для SFC для обмена данными между станциями.

Параметр	Используется в функции SFC					Объяв-ление	Тип	Содержание, описание
REQ	65	-	67	68	69	INPUT	BOOL	При REQ ="1" функция запускается на выполнение
CONT	65	-	67	68	-	INPUT	BOOL	При CONT ="1" соединение остается неразорванным после завершения выполнения функции
DEST_ID	65	-	67	68	69	INPUT	WORD	Идентификация коммуникационного партнера (MPI-адрес)
REQ_ID	65	-	-	-	-	INPUT	DWORD	Маркировка данных для идентификации задания
VAR_ADDR	-	-	67	68	-	INPUT	ANY	Область данных в CPU партнера
SD	65	-	-	68	-	INPUT	ANY	Область данных в "своем" CPU, которая содержит пересылаемые данные
EN_DT	-	66	-	-	-	INPUT	BOOL	При EN_DT ="1": принять получаемые данные в область назначения
RET_VAL	65	66	67	68	69	OUTPUT	INT	Информация об ошибках или число обработанных функцией данных
BUSY	65	-	67	68	69	OUTPUT	BOOL	При BUSY ="1" функция все еще выполняется
REQ_ID	-	66	-	-	-	OUTPUT	DWORD	Маркировка данных для идентификации задания
NDA	-	66	-	-	-	OUTPUT	BOOL	При NDA ="1" принятые данные поступили в область назначения; При NDA ="0" нет данных во внутренней очереди.
RD	-	66	67	-	-	OUTPUT	ANY	Область данных в "своем" CPU, которая получает принятые данные.

SFC 65 X_SEND
Пересылка данных (Send data)

Выполнение данной функции начинается при следующих значениях параметров REQ и BUSY: REQ = "1" и BUSY = "0" (при первом вызове). Пока функция выполняется, параметр BUSY находится в установленном состоянии (BUSY = "1"). При этом любые изменения параметра REQ не влияют на процесс выполнения задания. Как только выполнение функции завершается, параметр BUSY сбрасывается (BUSY = "0"). Если при этом параметр REQ все еще установлен (REQ = "1"), то сразу же вновь начинает выполняться данная функция.

При первом вызове операционная система пересылает все данные из области источника во внутренний буфер, а затем пересылает эти данные CPU партнера.

Пока выполняется функция пересылки данных, параметр BUSY находится в установленном состоянии (BUSY = "1"). Если устройство-партнер сигнализирует о получении данных, параметр BUSY сбрасывается в "0" и выполнение функции пересылки данных завершается.

Если параметр CONT сброшен (CONT = "0"), то коммуникационное соединение разорвано, и ресурсы соответствующего CPU доступны для установления других коммуникационных связей. Если параметр CONT установлен (CONT = "1"), то коммуникационное соединение создано. Параметр REQ_ID позволяет назначить посылаемым данным идентификатор ID, который в дальнейшем может быть проверен с помощью функции SFC X_RCV.

Параметры SD описывает область, из которой пересылаемые данные должны быть считаны. Фактическими параметрами функции могут быть адреса, переменные или области данных, адресованные указателем ANY. Передаваемые и принимаемые данные при этом не проверяются на корректность (на соответствие типу данных).

SFC 66 X_RCV
Прием данных (Receive data)

При выполнении функции принимаемые данные помещаются во внутренний буфер. Несколько пакетов данных могут быть помещены в очередь в хронологическом порядке - в порядке их поступления.

Для проверки факта принятия данных используется параметр EN_DT со значением "0". Если EN_DT = "0", параметр NDA = "1", а в параметре RET_VAL содержится число принятых байтов данных. При этом параметр REQ_ID содержит такое же значение, как и соответствующий параметр функции SFC 65 X_SEND. Если параметр EN_DT = "1", то рассматриваемая функция SFC 66 X_RCV передает самый первый пакет данных (поступивший первым) в область назначения; при этом параметр NDA = "1", а в параметре RET_VAL содержится число переданных байтов данных. Если параметр EN_DT = "1", а в очереди при этом отсутствуют данные, то параметр NDA = "0".

При полном перезапуске все пакеты данных в очереди отбрасываются. В случае разрыва соединения или при перезапуске сохраняется только первый пакет данных в очереди (поступивший первым), при условии, что он уже был запрошен посредством параметра EN_DT = "0", в противоположном случае он также будет отброшен, как и остальные

данные.

Параметр RD описывает область, в которую принимаемые данные должны быть записаны. Фактическими параметрами функции могут быть адреса, переменные или области данных, адресованные указателем ANY.

Передаваемые и принимаемые данные при этом не проверяются на корректность (на соответствие типу данных). Если принимаемые данные не имеют значения для Вас, то в качестве параметра RD функции SFC 66 X_RCV допустимо использовать "пустой" указатель ANY (NIL указатель).

SFC 67 X_GET
Считывание данных (Read data)

Выполнение данной функции начинается при следующих значениях параметров REQ и BUSY: REQ = "1" и BUSY = "0" (при первом вызове). Пока функция выполняется, параметр BUSY остается установленным (BUSY = "1"). При этом любые изменения параметра REQ не влияют на процесс выполнения задания.

Как только выполнение функции завершается, параметр BUSY сбрасывается (BUSY = "0"). Если параметр REQ все еще установлен (REQ = "1"), то сразу же вновь начинает выполняться данная функция.

Если запущена процедура считывания данных, операционная система в CPU партнера собирает и посылает данные, запрашиваемые в параметре VAR_ADDR. При вызове SFC принятые данные пересылаются в область назначения, определенную в параметре RD. Параметр RET_VAL при этом содержит число переданных байтов.

Если параметр CONT сброшен (CONT = "0"), то коммуникационное соединение разорвано, если параметр CONT установлен (CONT = "1"), то коммуникационное соединение установлено. Данные также могут быть считаны, когда коммуникационный партнер находится в режиме STOP.

Параметры RD и VAR_ADDR описывают область, из которой пересылаемые данные должны быть считаны, или в которую принимаемые данные должны быть записаны. Фактическими параметрами функции могут быть адреса, переменные или области данных, адресованные указателем ANY. Передаваемые и принимаемые данные при этом не проверяются на корректность (на соответствие типу данных).

SFC 68 X_PUT
Запись данных (Write data)

Выполнение данной функции начинается при следующих значениях параметров REQ и BUSY: REQ = "1" и BUSY = "0" (при первом вызове). Пока функция выполняется, параметр BUSY установлен (BUSY = "1"). При этом любые изменения параметра REQ не влияют на процесс выполнения задания.

Как только выполнение функции завершается, параметр BUSY сбрасывается (BUSY = "0"). Если параметр REQ все еще установлен (REQ = "1"), то сразу же вновь начинает выполняться данная функция.

Если запущена процедура записывания данных, операционная система при первом вызове передает все данные из области источника, определенной в параметре SD, во внутренний буфер при первом вызове, а затем посылает их коммуникационному партнеру. Операционная система CPU партнера записывает принятые данные в область, определенную в параметре VAR_ADDR. После этого параметр BUSY сбрасывается (BUSY = "0").

Данные также могут быть записаны, когда коммуникационный партнер находится в режиме STOP.

Параметры RD и VAR_ADDR описывают область, из которой пересылаемые данные должны быть считаны, или в которую принимаемые данные должны быть записаны. Фактическими параметрами функции могут быть адреса, переменные или области данных, адресованные указателем ANY. Передаваемые и принимаемые данные при этом не проверяются на корректность (на соответствие типу данных).

SFC 69 X_ABORT
Рассоединение связи (Disconnect)

Выполнение данной функции инициируется при значении параметра REQ, равном "1" - при этом происходит разрывание связи с определенным коммуникационным партнером. С помощью системной функции SFC 69 X_ABORT Вы можете разорвать только те связи, которые установлены в рассматриваемой станции с помощью системных функций SFC X_SEND, X_GET и X_PUT. Пока функция выполняется, параметр BUSY установлен (BUSY = "1"). При этом любые изменения параметра REQ не влияют на процесс выполнения функции. Как только выполнение функции завершается, параметр BUSY сбрасывается (BUSY = "0"). Если параметр REQ все еще установлен (REQ = "1"), то сразу же вновь начинает выполняться данная функция.

20.7 SFB-коммуникации

20.7.1 Основы

При использовании SFB-коммуникаций (коммуникаций посредством системных функциональных блоков) Вы можете организовать обмен данными большого объема между разными SIMATIC-станциями. При этом станции должны быть связаны друг с другом посредством подсети; это может быть подсеть MPI, PROFIBUS или Ethernet. В данном случае коммуникационные соединения будут иметь статический характер. Поэтому эти соединения должны конфигурироваться в таблице соединений ("Communication via configured connections", extended communication - "Функции связи посредством сконфигурированных соединений", дополнительные функции связи).

Коммуникационные функции являются системными функциональными блоками, встроенными в операционную систему S7-400 CPU. Связанные экземплярные блоки при этом размещаются в памяти пользователя.

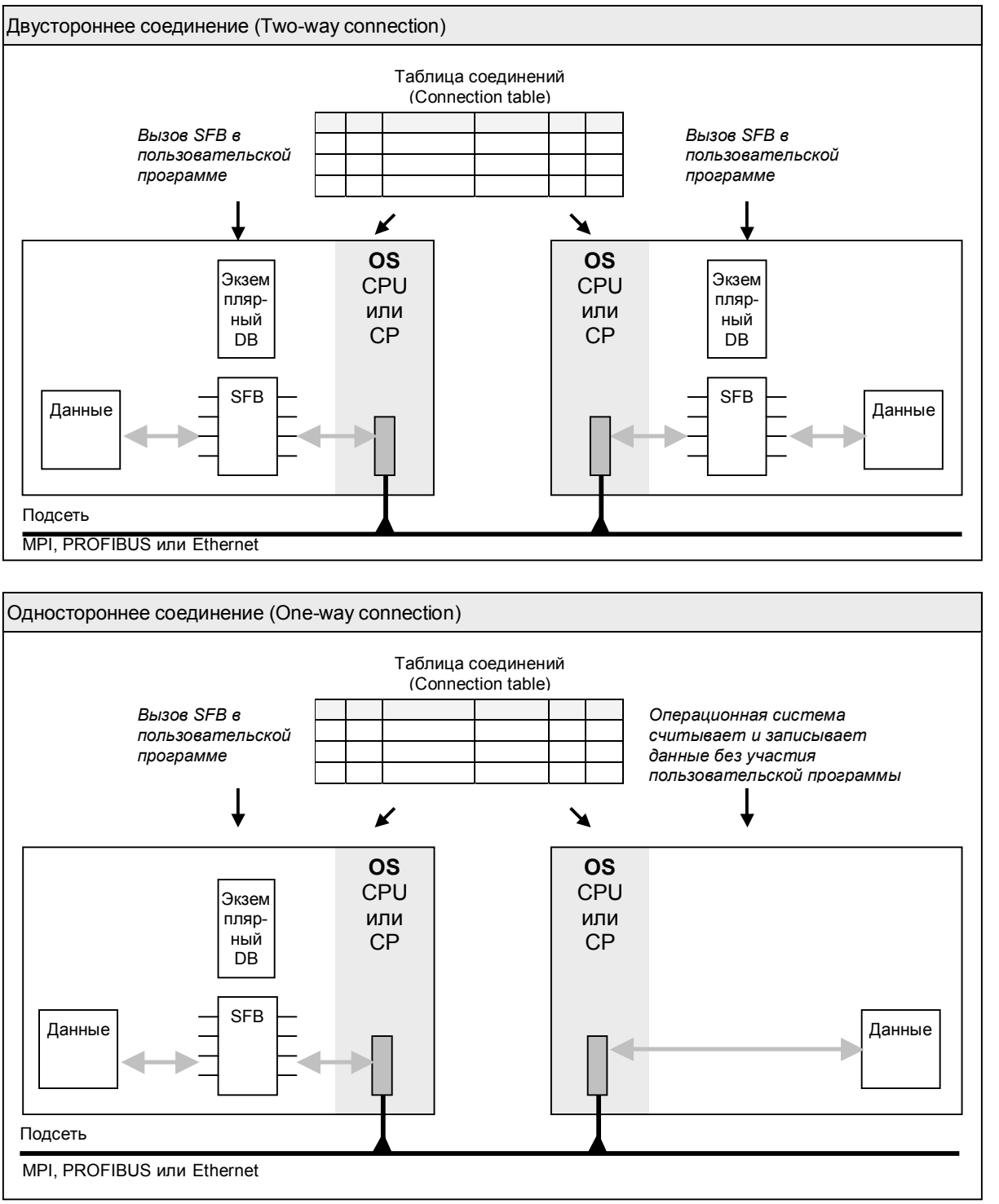


Рис. 20.16 SFB-коммуникации

Если Вам необходимо использовать SFB-коммуникации, то скопируйте описание интерфейса SFB из стандартной библиотеки *Standard Library* из раздела *System Function Blocks (Системные функциональные блоки)* в каталог *Blocks (Блоки)*, создайте экземплярные блоки данных для каждого

вызова SFB со связанным экземплярным блоком данных. В случае инкрементного программирования Вы можете также выбирать SFB из каталога программных элементов (program element catalog) и получить автоматически созданный экземплярный блок данных.

Конфигурирование SFB-коммуникаций

Предпосылкой для осуществления коммуникаций посредством системных функциональных блоков является наличие сконфигурированной таблицы соединений (Connection table), в которой должны быть определены коммуникационные соединения.

Коммуникационные соединения специфицируются идентификаторами (ID соединения) для каждого коммуникационного партнера. STEP 7 назначает ID соединения при компилировании таблицы соединений (Connection table). При этом различаются "local ID" ("локальные ID") для инициализации SFB в локальном (или "собственном" - "own") модуле и "remote ID" ("удаленные ID") для инициализации SFB в модуле коммуникационного партнера.

Одни и те же логические соединения могут быть использованы для различных запросов функций передачи/приема (Send/Receive). Чтобы различать подобные случаи, Вы должны дополнительно использовать идентификатор задания (job ID) вместе с ID соединения (connection ID), чтобы определить связи между передающим (Send) блоком и принимающим (Receive) блоком.

Инициализация

Для установления связи с коммуникационным партнером при перезапуске необходимо инициализировать SFB-коммуникации. Инициализация имеет место в CPU, имеющем атрибут "Active connection buildup = Yes" ("Установление активного соединения = Да") в таблице соединений. Вы должны в цикле вызывать используемые коммуникационные SFB в OB перезапуска и инициализировать доступные параметры, как показано ниже:

- REQ = FALSE (ЛОЖЬ)
- ID = ID локального соединения из таблицы соединений (connection table) (тип данных: WORD W#16#xxxx)
- PI_NAME = переменной с содержанием 'P_PROGRAM' в кодировке ASCII (например, ARRAY[1..9] OF CHAR).

Функциональные блоки должны продолжать вызываться в программном цикле, пока параметр DONE не получит значение "1". Параметры ERROR и STATUS предоставляют информацию, касающуюся ошибок, которые были обнаружены, и о состоянии (статусе) задания. Вам не нужно переключать области данных при перезапуске (это касается параметров ADDR_x, RD_x и SD_x).

В программном цикле должны "абсолютно" вызываться функциональные блоки SFB, и при этом управление обменом данными может осуществляться с помощью параметров REQ и EN_R.

20.7.2 Двусторонний обмен данными (Two-way Data Exchange)

При использовании двустороннего обмена данными (Two-way Data Exchange) Вам необходимо организовать вызов одного передающего (SEND) блока и одного принимающего (Receive) блока на соответствующих концах коммуникационного соединения. Оба блока должны иметь идентификаторы соединения (connection ID), которые находятся в таблице соединений в одной и той же строке. Вы можете также использовать несколько пар блоков, которые должны при этом отличаться идентификаторами задания (job ID).

Для двустороннего обмена данными (Two-way Data Exchange) используются следующие SFB:

- SFB 8 USEND
  функциональный блок для некоординированной передачи пакета данных, размер которого определяется CPU (Uncoordinated sending of data packet),
- SFB 9 URCV
  функциональный блок для некоординированного приема пакета данных, размер которого определяется CPU (Uncoordinated receiving of data packet),
- SFB 12 BSEND
  функциональный блок для передачи блока данных размером до 64 кбайтов (Block-oriented sending of data),
- SFB 13 BRCV
  функциональный блок для приема блока данных размером до 64 кбайтов (Block-oriented sending of data).

Список параметров для этих SFB представлен в таблице 20.13.

SFB 8 USEND и SFB 9 URCV
Некоординированная передача и прием пакета данных
(Uncoordinated sending/receiving of data packet)

В рассматриваемых системных функциональных блоках параметры SD_x и RD_x используются для определения переменной или области, предназначенной для передачи данных. Область пересылаемых данных SD_x должна соответствовать области принимаемых данных RD_x. Эти параметры должны задавать область данных без каких-либо пропусков (пробелов), начиная с 1.

Не требуется определять значения для неиспользуемых параметров (так как и в FB; не все параметры SFB требуют присвоения значений).

При первом вызове SFB 9 создается "приемный почтовый ящик" (Receive mailbox); при всех последующих вызовах прием данных производится в данный "почтовый ящик".

Положительный фронт сигнала, поступающего в параметр REQ (request - "запрос") запускает процедуру обмена данными, положительный же фронт сигнала, поступающего в параметр R (reset - "сброс") прекращает процедуру обмена. Значение "1" в параметре EN_R (enable receive - прием разрешен) сигнализирует о том, что коммуникационный партнер готов принять данные.

Таблица 20.13 Параметры для SFB для передачи и приема данных.

Параметр	Используется в SFB				Объяв-ление	Тип	Содержание, описание
REQ	8	-	12	-	INPUT	BOOL	При REQ ="1" функция запускается на выполнение
EN_R	-	9	-	13	INPUT	BOOL	Параметр EN_R ="1" - это сигнал о том, что партнер готов к приему данных
R	-	-	12	-	INPUT	BOOL	При положительном фронте сигнала в параметре R обмен данных прерывается
ID	8	9	12	13	INPUT	WORD	ID соединения (connection ID)
R_ID	8	9	12	13	INPUT	DWORD	ID задания (job ID)
DONE	8	-	12	-	OUTPUT	BOOL	Задание завершено
NDR	-	9	-	13	OUTPUT	BOOL	Считаны новые данные
ERROR	8	9	12	13	OUTPUT	BOOL	Обнаружена ошибка
STATUS	8	9	12	13	OUTPUT	WORD	Состояние (статус) задания
SD_1	8	-	12	-	IN_OUT	ANY	1-я область данных для пересылки
SD_2	8	-	-	-	IN_OUT	ANY	2-я область данных для пересылки
SD_3	8	-	-	-	IN_OUT	ANY	3-я область данных для пересылки
SD_4	8	-	-	-	IN_OUT	ANY	4-я область данных для пересылки
RD_1	-	9	-	13	IN_OUT	ANY	1-я область для приема данных
RD_2	-	9	-	-	IN_OUT	ANY	2-я область для приема данных
RD_3	-	9	-	-	IN_OUT	ANY	3-я область для приема данных
RD_4	-	9	-	-	IN_OUT	ANY	4-я область для приема данных
LEN	-	-	12	13	IN_OUT	WORD	Длина блока данных в байтах

В параметре ID инициализируются идентификаторы соединения (connection ID), которые вводятся системой STEP 7 в таблицу соединений (connection table) и для локального модуля, и для модуля партнера (эти два идентификатора соединения могут различаться). Параметр R_ID позволяет задать определяющий и уникальный идентификатор задания (job ID), который должен быть идентичен для передаваемого (Send) и получаемого (Receive) блоков данных. Такой подход позволяет для нескольких пар передаваемых (Send) и получаемых (Receive) блоков использовать единое логическое соединение (так как каждая пара имеет уникальный идентификатор).

При первом вызове блок передает фактические значения параметров ID и R_ID в соответствующий экземплярный блок. При первом вызове устанавливается коммуникационная связь (для данного экземпляра) до момента следующего полного перезапуска.

Значениями параметра DONE и параметра NDR, равными "1", блок сигнализирует о том, что задание выполнено без ошибок. Если в процессе выполнения задания обнаруживается ошибка, то для индикации этого используется параметр ERROR. Если значение параметра STATUS отлично от нуля и при этом ERROR = "0", то это соответствует сигналу предупреждения, а если при этом ERROR = "1", то это соответствует сигналу об ошибке (о сбое). Пользователь должен обеспечить проверку параметров DONE, NDR, STATUS и ERROR после *каждого* вызова блока.

SFB 12 BSEND и SFB 13 BRCV
Передача и прием блока данных
(Block-oriented sending/receiving of data)

В рассматриваемых системных функциональных блоках параметры SD_x и RD_x используются для указания на первый байт области данных (при этом длина области не проверяется); число байтов пересылаемых или принимаемых данных должна быть задана область в параметре LEN.

Указанные функции позволяют передавать до 64 кбайтов данных; при этом данные передаются в блоках (иногда называемых фреймами [frames]). Такая передача данных сама по себе является асинхронной по отношению к процессу сканирования пользовательской программы.

Положительный фронт сигнала, поступающего в параметр REQ (request - "запрос") запускает процедуру обмена данными, тогда как положительный фронт сигнала, поступающего в параметр R (reset - "сброс") прекращает процедуру обмена. Значение "1" в параметре EN_R (enable receive - прием разрешен) сигнализирует о том, что коммуникационный партнер готов принять данные. В параметре ID инициализируются идентификаторы соединения (connection ID), которые вводятся системой STEP 7 в таблицу соединений (connection table) и для локального модуля, и для модуля партнера (эти два идентификатора соединения могут различаться).

Параметр R_ID позволяет задать определяющий и уникальный идентификатор задания (job ID), который должен быть идентичен для передаваемого (Send) и получаемого (Receive) блоков данных. Такой подход позволяет для нескольких пар передаваемых (Send) и получаемых (Receive) блоков использовать единое логическое соединение (так как каждая пара имеет уникальный идентификатор).

При первом вызове блок передает фактические значения параметров ID и R_ID в соответствующий экземплярный блок. При первом вызове устанавливается коммуникационная связь (для данного экземпляра) до момента следующего полного перезапуска.

Значениями параметра DONE и параметра NDR, равными "1", блок сигнализирует о том, что задание выполнено без ошибок. Если в процессе выполнения задания обнаруживается ошибка, то для индикации этого используется параметр ERROR. Если значение параметра STATUS отлично от нуля и при этом ERROR = "0", то это соответствует сигналу предупреждения, а если при этом ERROR = "1", то это соответствует сигналу об ошибке (о сбое). Пользователь должен обеспечить проверку параметров DONE, NDR, STATUS и ERROR после *каждого* вызова блока.

20.7.3 Односторонний обмен данными (One-way Data Exchange)

При одностороннем обмене данными (One-way Data Exchange) Вам необходимо организовать вызов одного коммуникационного блока SFB только в одном из CPU. Операционная система CPU коммуникационного партнера самостоятельно выполнит необходимые коммуникационные функции.

Для одностороннего обмена данными (One-way Data Exchange) используются следующие SFB:

- SFB 14 GET
  функциональный блок для считывания данных, максимальный размер которых определяется CPU,
- SFB 15 PUT
  функциональный блок для записывания данных, максимальный размер которых определяется CPU.

Список параметров для функциональных блоков SFB представлен в таблице 20.14.

Операционная система CPU коммуникационного партнера собирает данные, считанные с помощью системного функционального блока SFB 14; операционная система CPU коммуникационного партнера распределяет данные, записываемые с помощью системного функционального блока SFB 15. В данных случаях пользователю не приходится создавать программы для CPU коммуникационного партнера для пересылки или приема данных.

Положительный фронт сигнала, поступающего в параметр REQ (request - "запрос") запускает процедуру обмена данными.

В параметре ID инициализируется идентификатор соединения (connection ID), который вводится системой STEP 7 в таблицу соединений (connection table).

Значениями параметра DONE и параметра NDR, равными "1", блок сигнализирует о том, что задание выполнено без ошибок. Если в процессе выполнения задания обнаруживается ошибка (любая), то при этом ERROR = "1".

Если значение параметра STATUS отлично от нуля и при этом ERROR = "0", то это соответствует сигналу предупреждения, а если при этом ERROR = "1", то это соответствует сигналу об ошибке (о сбое). Пользователь должен обеспечить проверку параметров DONE, NDR, STATUS и ERROR после *каждого* вызова блока.

С помощью параметра ADDR_n определяется переменная или область в CPU коммуникационного партнера, из которой необходимо считать или в которую необходимо записать данные. Области, определенные в ADDR_n, должны совпадать с областями, определенными в параметрах SD_x или RD_x. Эти параметры должны задавать область данных без каких-либо пропусков (пробелов), начиная с 1.

Не требуется определять значения для неиспользуемых параметров (так как и в FB: не все параметры SFB требуют присвоения значений).

Таблица 20.14 Параметры для SFB для чтения и записи данных.

Параметр	Используется в SFB		Объяв-ление	Тип	Содержание, описание
REQ	14	15	INPUT	BOOL	При REQ ="1" функция запускается на выполнение
ID	14	15	INPUT	WORD	ID соединения (connection ID)
NDR	14	-	OUTPUT	BOOL	Считаны новые данные
DONE	-	15	OUTPUT	BOOL	Задание завершено
ERROR	14	15	OUTPUT	BOOL	Обнаружена ошибка
STATUS	14	15	OUTPUT	WORD	Состояние (статус) задания
ADDR_1	14	15	IN_OUT	ANY	1-я область данных в CPU коммуникационного партнера
ADDR_2	14	15	IN_OUT	ANY	2-я область данных в CPU коммуникационного партнера
ADDR_3	14	15	IN_OUT	ANY	3-я область данных в CPU коммуникационного партнера
ADDR_4	14	15	IN_OUT	ANY	4-я область данных в CPU коммуникационного партнера
RD_1	14	-	IN_OUT	ANY	1-я область для приема данных
RD_2	14	-	IN_OUT	ANY	2-я область для приема данных
RD_3	14	-	IN_OUT	ANY	3-я область для приема данных
RD_4	14	-	IN_OUT	ANY	4-я область для приема данных
SD_1	-	15	IN_OUT	ANY	1-я область данных для пересылки
SD_2	-	15	IN_OUT	ANY	2-я область данных для пересылки
SD_3	-	15	IN_OUT	ANY	3-я область данных для пересылки
SD_4	-	15	IN_OUT	ANY	4-я область данных для пересылки

20.7.4 Передача данных на принтер (Print Data)

Функциональный блок **SFB 16 PRINT** позволяет организовать передачу данных на печатающее устройство вместе со спецификацией формата этих данных с использованием коммуникационного процессора CP 441.

Список параметров для функционального блока SFB 14 GET представлен в таблице 20.15.

Положительный фронт сигнала, поступающего в параметр REQ (request - "запрос") запускает процедуру обмена данными с принтером в соответствии с параметрами ID и PRN_NR. В параметре ID инициализируется идентификатор соединения (connection ID), в параметре PRN_NR инициализируется номер принтера (prnter number).

Значением параметра DONE, равным "1", блок сигнализирует о том, что задание выполнено без ошибок. Если в процессе выполнения задания

обнаруживается ошибка (любая), то при этом ERROR = "1".

Если значение параметра STATUS отлично от нуля и при этом ERROR = "0", то это соответствует сигналу предупреждения, а если при этом ERROR = "1", то это соответствует сигналу об ошибке (о сбое). Пользователь должен обеспечить проверку параметров DONE, STATUS и ERROR после *каждого* вызова блока.

Символы для печати необходимо ввести в формате STRING в параметр FORMAT. Вы можете задать до 4 описаний форматов для переменных, определенных в параметрах SD_1…SD_4. Эти параметры должны задаваться без каких-либо пропусков (пробелов), начиная с 1. Не требуется определять значения для неиспользуемых параметров.

На один запрос задания для принтера Вы можете передавать до 420 байтов данных (суммарно для формата и всех переменных).

Таблица 20.15 Параметры для SFB 16 PRINT

Параметр	Объяв-ление	Тип	Содержание, описание
REQ	INPUT	BOOL	При REQ ="1" функция запускается на выполнение
ID	INPUT	WORD	ID соединения (connection ID)
DONE	OUTPUT	BOOL	Задание завершено
ERROR	OUTPUT	BOOL	Обнаружена ошибка
STATUS	OUTPUT	WORD	Состояние (статус) задания
PRN_NR	IN_OUT	BYTE	Номер принтера (prnter number)
FORMAT	IN_OUT	STRING	Описание формата данных
SD_1	IN_OUT	ANY	1-я переменная
SD_2	IN_OUT	ANY	2-я переменная
SD_3	IN_OUT	ANY	3-я переменная
SD_4	IN_OUT	ANY	4-я переменная

20.7.5 Функции управления (Control Functions)

Для управления коммуникационным партнером используются следующие SFB:

- SFB 19 START
  функциональный блок для инициации "полного" (complete) перезапуска контроллера коммуникационного партнера,
- SFB 20 STOP
  функциональный блок для выполнения переключения контроллера коммуникационного партнера в режим STOP,
- SFB 21 RESUME
  функциональный блок для инициации "теплого" (warm) перезапуска контроллера коммуникационного партнера.

Данные функциональные блоки предназначены для одностороннего обмена данными; пользователю не приходится создавать программы для контроллера коммуникационного партнера, чтобы обрабатывать рассматриваемые функциональные блоки.

Список параметров для функциональных блоков SFB представлен в таблице 20.16.

Таблица 20.16 Параметры для SFB для управления контроллером коммуникационного партнера.

Параметр	Используется в SFB			Объяв-ление	Тип	Содержание, описание
REQ	19	20	21	INPUT	BOOL	При REQ ="1" задание запускается на выполнение
ID	19	20	21	INPUT	WORD	ID соединения (connection ID)
DONE	19	20	21	OUTPUT	BOOL	Задание завершено
ERROR	19	20	21	OUTPUT	BOOL	Обнаружена ошибка
STATUS	19	20	21	OUTPUT	WORD	Состояние (статус) задания
PI_NAME	19	20	21	IN_OUT	ANY	Имя программы (P_PROGRAM)
ARG	19	-	21	IN_OUT	ANY	Не имеет значения
IO_STATE	19	20	21	IN_OUT	BYTE	Не имеет значения

Положительный фронт сигнала, поступающего в параметр REQ (request - "запрос") запускает процедуру обмена данными. В параметре ID инициализируется идентификатор соединения (connection ID), который вводится системой STEP 7 в таблицу соединений (connection table).

Значением параметра DONE, равным "1", блок сигнализирует о том, что задание выполнено без ошибок. Если в процессе выполнения задания обнаруживается ошибка (любая), то при этом ERROR = "1".

Если значение параметра STATUS отлично от нуля и при этом ERROR = "0", то это соответствует сигналу предупреждения, а если при этом ERROR = "1", то это соответствует сигналу об ошибке (о сбое). Пользователь должен обеспечить проверку параметров DONE, STATUS и ERROR после *каждого* вызова блока.

С помощью параметра PI_NAME определяется массив символов, содержащий имя программы "P_PROGRAM" (ARRAY [1...9] OF CHAR).

Параметры ARG и IO_STATE в настоящее время не определены и не имеют значения.

Функциональный блок **SFB 19 START** позволяет выполнить "полный" (complete) перезапуск CPU коммуникационного партнера. Необходимые условия для выполнения "полного" перезапуска: CPU коммуникационного партнера должен находиться в режиме STOP, а переключатель режимов должен находиться в положении RUN или RUN_P.

Функциональный блок **SFB 20 STOP** позволяет перевести CPU коммуникационного партнера в режим STOP. Необходимым условием для корректного выполнения задания: CPU партнера не должен находиться в режиме STOP в момент прихода запроса на выполнение задания.

Функциональный блок **SFB 21 RESUME** позволяет выполнить "теплый" (warm) перезапуск CPU коммуникационного партнера. Необходимые условия для выполнения "теплого" перезапуска: CPU коммуникационного партнера должен находиться в режиме STOP, переключатель режимов должен находиться в положении RUN или RUN_P и в момент прихода запроса на выполнение задания должно допускаться выполнение "теплого" перезапуска.

20.7.6 Функции мониторинга (Monitoring Functions)

Для осуществления мониторинга используются следующие системные функциональные блоки:

- SFB 22 STATUS
  функциональный блок для проверки состояния коммуникационного партнера ("status" - статус),
- SFB 23 USTATUS
  функциональный блок для приема от коммуникационного партнера информации о его состоянии ("status" - статус),
- SFC 62 CONTROL
  функциональный блок для проверки состояния ("status" - статус) экземпляра SFB.

Список параметров для системных функциональных блоков SFB 22 и SFB 23 представлен в таблице 20.17, список параметров для системной функции SFC 62 представлен в таблице 20.18

Таблица 20.17 Параметры для функциональных блоков SFB 22 и SFB 23

Параметр	Используется в SFB		Объяв-ление	Тип	Содержание, описание
REQ	22	-	INPUT	BOOL	При REQ = "1" задание запускается на выполнение
EN_R	-	23	INPUT	BOOL	EN_R = "1" означает готовность к приему данных
ID	22	23	INPUT	WORD	ID соединения (connection ID)
NDR	22	23	OUTPUT	BOOL	Считаны новые данные
ERROR	22	23	OUTPUT	BOOL	Обнаружена ошибка
STATUS	22	23	OUTPUT	WORD	Состояние (статус) задания
PHYS	22	23	IN_OUT	ANY	Физическое состояние (Phisical status)
LOG	22	23	IN_OUT	ANY	Логическое состояние (Logical status)
LOCAL	22	23	IN_OUT	ANY	Состояние (статус) коммуникационного партнера S7 CPU

Таблица 20.18 Параметры для системной функции SFC 62 CONTROL

Параметр	Объяв-ление	Тип	Содержание, описание
EN_R	INPUT	BOOL	EN_R = "1" означает готовность к приему данных
I_DB	INPUT	BLOCK_DB	Экземплярный блок данных
OFFSET	INPUT	WORD	Номер локального экземпляра
RET_VAL	OUTPUT	INT	Информация об ошибке
ERROR	OUTPUT	BOOL	Обнаружена ошибка
STATUS	OUTPUT	WORD	Слово состояния
I_TIP	OUTPUT	BYTE	Идентификатор типа блока
I_STATE	OUTPUT	BYTE	Идентификатор текущего состояния (status)
I_CONN	OUTPUT	BOOL	Состояние соединения (если I_CONN = "1", то соединение установлено)
I_STATUS	OUTPUT	WORD	Параметр состояния STATUS для экземпляра SFB

Общая информация, касающаяся рассматриваемых системных блоков:

Если в процессе выполнения задания обнаруживается ошибка (любая), то при этом ERROR = "1". Если значение параметра STATUS отлично от нуля и при этом ERROR = "0", то это соответствует сигналу предупреждения, а если при этом ERROR = "1", то это соответствует сигналу об ошибке.

SFB 22 STATUS
Проверка состояния коммуникационного партнера

Функциональный блок SFB 22 STATUS предназначен для считывания состояния CPU коммуникационного партнера и отображения его в параметрах PHYS (физическое состояние), LOG (логическое состояние) и LOCAL (рабочее состояние, если коммуникационным партнером является S7 CPU).

Положительный фронт сигнала, поступающего в параметр REQ (request - "запрос") инициирует запрос на выполнение процедуры. В параметре ID инициализируется идентификатор соединения (connection ID), который вводится системой STEP 7 в таблицу соединений (connection table).

Значением параметра NDR, равным "1", блок сигнализирует о том, что задание выполнено без ошибок. Пользователь должен обеспечить проверку параметров NDR, STATUS и ERROR после *каждого* вызова блока.

SFB 23 USTATUS
Прием от коммуникационного партнера информации о его состоянии

Функциональный блок SFB 23 USTATUS предназначен для приема от коммуникационного партнера информации о его состоянии, которую он посылает в связи с произошедшими изменениями и без дополнительного

запроса от локального модуля. Состояние устройства отображается в параметрах PHYS, LOG и LOCAL.

Если параметр EN_R (enable receive - "прием разрешен") содержит единичное значение (EN_R = "1"), это означает готовность к приему данных. В параметре ID инициализируется идентификатор соединения (connection ID), который вводится системой STEP 7 в таблицу соединений (connection table).

Значением параметра NDR, равным "1", блок сигнализирует о том, что задание выполнено без ошибок. Пользователь должен обеспечить проверку параметров NDR, STATUS и ERROR после *каждого* вызова блока.

SFC 62 CONTROL
Проверка состояния экземпляра SFB

Системная функция SFC 62 CONTROL позволяет определить состояние экземпляра SFB и связанного (associated) соединения в локальном контроллере. В параметре I_DB необходимо задать экземплярный блок данных. Если SFB вызывается как локальный экземпляр, определите номер локального экземпляра в параметре OFFSET (0 - если нет локального экземпляра, 1 - для первого локального экземпляра, 2 - для второго локального экземпляра, и т.д.).

Если параметр EN_R (enable receive - "прием разрешен") содержит единичное значение (EN_R = "1"), это означает готовность к приему данных. В параметре ID инициализируется идентификатор соединения (connection ID), который вводится системой STEP 7 в таблицу соединений (connection table).

Значением параметра NDR, равным "1", блок сигнализирует о том, что задание выполнено без ошибок. Пользователь должен обеспечить проверку параметров NDR, STATUS и ERROR после *каждого* вызова блока.

Параметры I_TIP, I_STATE, I_CONN и I_STATUS содержат информацию, касающуюся состояния локального экземпляра SFB.

21 Обработка прерываний

Обработка прерываний всегда обусловлена событиями. Когда операционная система получает сигнал о том, что произошло некоторое событие, она прекращает сканирование основной программы и вызывает определенную программу, соответствующую этому конкретному событию. После выполнения этой программы операционная система продолжает сканирование основной программы с точки прерывания. Такие прерывания обработки программы могут иметь место после каждой операции (оператора).

Упомянутыми событиями могут быть как собственно "прерывания" (interrupts) так и ошибки. Порядок, в котором обрабатываются фактически одновременно происходящие прерывания, регулируется планом приоритетов. Каждое событие имеет свой приоритет обработки. Отдельные прерывания могут быть объединены в классы приоритетов (priority class - "приоритетный" класс).

Каждая программа, связанная с событием-прерыванием, должна быть записана в организационном блоке, из которого могут быть вызваны также и другие блоки. Событие с более высоким приоритетом прерывает выполняемую программу в организационном блоке с более низким приоритетом. Пользователь может вызывать прерывание процесса выполнения программы событиями с высоким приоритетом, используя системные функции.

21.1 Общие замечания

SIMATIC S7 предоставляет пользователю следующие типы прерываний:

- аппаратные прерывания (hardware interrupts) -
  прерывания, вызываемые модулем (посредством входного сигнала от процесса, или посредством сигнала, сгенерированного собственно в модуле);
- таймерные прерывания (watchdog interrupts) -
  прерывания, генерируемые операционной системой с заданным периодом (интервалом);
- временные (по времени суток) прерывания (time-of-day interrupts) -
  прерывания, генерируемые операционной системой в заданное время суток (однократные или периодические);
- прерывания с задержкой обработки (time-delay interrupts) -
  прерывания, генерируемые по истечении заданного периода времени (при этом в системной функции можно определять момент начала отсчета времени заданного периода);
- прерывания мультипроцессорного режима (multiprocessor interrupts) -
  прерывания, генерируемые другими CPU в подсети, при мультипроцессорном режиме.

Другие прерывания вызываются такими событиями, как синхронные ошибки, которые могут происходить при сканировании программы, и асинхронные ошибки, которыми могут являться, например, диагностические прерывания. Обработка этих событий обсуждается в главе 23 "Обработка ошибок".

Приоритеты

Событие с более высоким приоритетом прерывает выполняемую программу обработки некоторого события с более низким приоритетом. Основная программа (main program) имеет самый низкий приоритет (она относится к приоритетному классу 1), асинхронные ошибки имеют самый высокий приоритет (относятся к приоритетному классу 26), не учитывая программу запуска (start-up routine). Все остальные события относятся к приоритетным классам, занимающим промежуточное положение в ряду приоритетов. В системах S7-300 приоритеты фиксированы; в системах S7-400 Вы можете самостоятельно изменять приоритеты при параметризации CPU.

Обзор всех приоритетных классов, а также организационные блоки, по умолчанию назначенные для каждого из них, рассматриваются в разделе 3.1.2 "Классы приоритетов".

Блокировка прерываний

Выполнение организационных блоков, запускаемых для обработки отдельных событий при сканировании программы, может быть заблокировано (disable) или, наоборот, разрешено (enable) с помощью системных функций SFC 39 DIS_IRT и SFC 40 EN_IRT соответственно, а может также быть заблокировано на время (т.е., отложено), а затем вновь разблокировано с помощью системных функций SFC 41 DIS_AIRT и SFC 42 EN_AIRT соответственно (см. раздел 21.7 "Обработка прерываний").

Текущие состояния сигналов

Нужно учитывать, что при выполнении программы обработки прерываний Вы должны иметь дело с текущими состояниями сигналов от I/O модулей (а не с состояниями сигналов входов, которые обновляются в начале выполнения основной программы), и должны записывать состояния выбранных сигналов непосредственно в I/O (не дожидаясь, пока таблица выходов образа процесса будет обновлена по окончании обработки основной программы).

В случае использования нескольких входов и выходов I/O модулей при выполнении программы, обрабатывающей прерывание, для доступа к I/O модулям достаточно использовать операции загрузки (load) и передачи (transfer). В таком случае рекомендуется установить строгое разграничение между главной программой и программой обработки прерывания в отношении I/O сигналов.

Если необходимо обрабатывать множество входных и выходных сигналов в программе обработки прерывания, то решением для систем S7-400 может быть использование нескольких образов подпроцессов (subprocess image). При назначении адресов Вы можете назначить каждому модулю отдельный образ подпроцесса. С помощью системных функций SFC 26

UPDAT_PI и SFC 27 UPDAT_PO Вы можете управлять отображениями подпроцессов из пользовательской программы (см. также раздел 20.2.1 "Обновление образа процесса").

В S7-400 CPU новых образцов Вы можете назначать образы входов и выходов подпроцесса для каждого организационного блока (для каждого приоритетного класса) и таким путем организовать автоматическое обновление образа процесса при каждом прерывании.

Стартовая информация, временные локальные данные

В таблице 21.1 представлен обзор стартовой информации для запускаемых событиями организационных блоков. В системах S7-300 временные локальные данные имеют фиксированную длину - 256 байтов. В системах S7-400 Вы можете сами определять размер области этих локальных данных для каждого приоритетного класса при параметризации CPU (параметр "локальные данные" [local data] блока), при этом в каждом случае нельзя превышать некоторый максимальный размер, определяемый типом CPU.

Необходимо отметить, что для временных локальных данных для каждого используемого приоритетного класса должно отводиться определенное минимальное число байтов, а именно 20 байтов, для размещения стартовой информации. Для неиспользуемых приоритетных классов Вы должны определить нулевое значение.

Таблица 21.1 Стартовая информация в организационных блоках для обработки прерываний

Байт	Прерывания мультипро-цессорного режима	Аппаратные прерывания	Таймерные прерывания (watchdog interrupts)	Прерывания с задержкой обработки (time-delay interrupts)	Прерывания по времени суток (time-of-day interrupts)
	OB 60	OB 40..OB 47	OB 30..OB 38	OB 20..OB 23	OB 10..OB 17
0	Класс события	Класс события	Класс события	Класс события	Класс события
1	"Стартовое" событие	"Стартовое" событие	"Стартовое" событие	"Стартовое" событие	"Стартовое" событие
2	Приоритетный класс	Приоритетный класс	Приоритетный класс	Приоритетный класс	Приоритетный класс
3	Номер OB	Номер OB	Номер OB	Номер OB	Номер OB
4	-	-	-	-	-
5	-	Идентификатор адреса	-	-	-
6..7	Идентификатор задания (INT)	Начальный адрес модуля (WORD)	"Фазовое" смещение [мс] (WORD)	Идентификатор задания (WORD)	Интервал (WORD)
8..9	-	Информация аппаратного прерывания (DWORD)	-	Оставшееся время задержки (TIME)	-
10..11	-		Время цикла [мс] (INT)		-
12..19	Время события (DT)	Время события (DT)	Время события (DT)	Время события (DT)	Время события (DT)

21.2 Аппаратные прерывания (Hardware Interrupts)

Аппаратные прерывания (hardware interrupts) используются для немедленного обнаружения из пользовательской программы событий, происходящих в управляемом процессе, что дает возможность выполнить программу обработки прерывания в качестве реакции на конкретное событие. STEP 7 предоставляет организационные блоки с OB 40 по OB 47 для обслуживания аппаратных прерываний. Тем не менее, то, какие из данных восьми организационных блоков доступны, зависит от типа CPU.

Обработка аппаратных прерываний может быть запрограммирована при конфигурировании оборудования (в данных конфигурирования оборудования). С помощью системных функций SFC 55 WR_PARM, SFC 56 WR_DPARM и SFC 57 PARM_MOD Вы можете задавать (изменять) параметры модулей даже в рабочем режиме RUN, используя возможности аппаратных прерываний.

21.2.1 Генерация аппаратных прерываний

Аппаратные прерывания генерируются в модулях, в которых заложена такая возможность. Это может быть, например, дискретный входной модуль, который обнаруживает сигнал, поступающий от процесса, или функциональный модуль, который генерирует аппаратное прерывание, причиной которого являются процессы, происходящие в модуле.

В установках по умолчанию аппаратные прерывания заблокированы. Для разрешения генерации аппаратного прерывания используется параметр, являющийся статическим; Вы можете также определить, будет ли генерироваться прерывание при приходящем событии, при уходящем событии или и при том, и при другом варианте (динамический параметр). Динамические параметры - это такие параметры, которые Вы можете изменять во время выполнения программы, используя системные функции SFC.

В интеллектуальных ведомых (slave) DP-устройствах, в которых предусмотрена такая возможность, Вы можете инициировать прерывание процесса в CPU ведущего (master) DP-устройства с помощью SFC 7 DP_PRAL.

Аппаратное прерывание подтверждается в модуле, когда заканчивается выполнение соответствующего организационного блока с программой обслуживания прерывания.

Прерывания в S7-300

Если во время выполнения организационного блока с программой обработки аппаратного прерывания происходит событие, которое инициирует генерацию такого же аппаратного прерывания, то данное аппаратное прерывание будет потеряно, если событие, вызывающее это аппаратное прерывание, не будет больше иметь место после того, как придет подтверждение завершения обработки предыдущего прерывания.

Не имеет значения, в каком модуле происходит данное событие - в

модуле, генерировавшем аппаратное прерывание, которое в текущий момент было обработано, или в другом модуле.

Во время обработки аппаратного прерывания может возникнуть диагностическое прерывание. Если в том же канале в промежутке времени между моментом, когда было создано первое аппаратное прерывание, и моментом, когда оно было квитировано, возникает другое аппаратное прерывание, то это второе прерывание будет потеряно, что подтвердит диагностическое прерывание, которое будет сгенерировано диагностическими средствами системы.

Прерывания в S7-400

Если во время выполнения организационного блока с программой обработки аппаратного прерывания в том же канале в том же модуле происходит событие, которое инициирует генерацию такого же аппаратного прерывания, то это новое аппаратное прерывание будет потеряно. Если событие происходит в том же модуле, но в другом канале, или же в другом модуле, то операционная система перезапустит организационный блок для обработки второго прерывания, как только будет завершена обработка первого прерывания.

21.2.2 Обслуживание аппаратных прерываний

Запрос информации прерывания

Начальный адрес модуля, который инициировал аппаратное прерывание, находится в байтах 6 и 7 области стартовой информации организационного блока, обрабатывающего данное прерывание. Если этот адрес является входным, то байт 5 области стартовой информации содержит B#16#54, в противоположном случае он содержит B#16#55. Если модуль, указанный в запросе, является дискретным входным модулем, то байты с 8 по 11 содержать информацию о состоянии входов; для всех других типов модулей в этих байтах содержится информация о состоянии (статусе) прерывания в модуле.

Обработка прерывания в программе запуска (start-up program)

При выполнении программы запуска (start-up program), модули не инициируют аппаратных прерываний. Обработка прерываний начинается с переходом к рабочему режиму RUN. Любые аппаратные прерывания, возникающие во время такого перехода теряются.

Обработка ошибок

Если инициировано аппаратное прерывание, для которого в пользовательской программе нет организационного блока, операционная система вызывает OB 85 (организационный блок обработки ошибок, возникающих при выполнении программы).

Аппаратное прерывание квитируется (подтверждается). Если OB 85 не запрограммирован, CPU переходит в режим STOP.

Блокировка (disabling), задержка во времени (delaying) и разблокировка (enabling)

Вызов организационных блоков обработки аппаратных прерываний может быть заблокирован (disable) и разблокирован (enable) с помощью системных функций SFC 39 DIS_IRT и SFC 40 EN_IRT соответственно, а также задержан на время (delay) и разблокирован (enable) с помощью системных функций SFC 41 DIS_AIRT и SFC 42 EN_AIRT соответственно.

21.2.3 Конфигурирование аппаратных прерываний с помощью STEP 7

Обработка аппаратных прерываний может быть запрограммирована при конфигурировании оборудования (в данных конфигурирования оборудования). Выберите CPU, откройте объект с помощью опций: *Edit -> Object Properties (Правка -> Свойства объекта)* и в диалоговом окне выберите вкладку "Interrupts" ("Прерывания").

Для систем S7-300 для блока OB 40 установленный по умолчанию приоритет 16 не может быть изменен. Для систем S7-400 для всех возможных организационных блоков (в соответствии со спецификациями CPU) Вы можете устанавливать приоритет в диапазоне от 2 до 24; при этом при установлении для блока приоритета с нулевым значением (0) блок перестанет выполняться. Никогда не назначайте один и тот же приоритет дважды, так как при этом могут быть потеряны прерывания в случае, если больше, чем 12 прерываний с одинаковым приоритетом будут инициированы одновременно.

Вы должны также разблокировать (enable) запуск аппаратных прерываний при использовании соответствующих модулей. Для этого эти модули параметризируются точно также как CPU.

При сохранении результатов конфигурирования оборудования STEP 7 записывает скомпилированные данные в *System data (Системные данные)* в пользовательской папке *Blocks (Блоки)*, откуда Вы сможете выгрузить данные параметризации в CPU, пока CPU находится в состоянии режима STOP. Данные параметризации для CPU вступают в силу немедленно после загрузки; назначения параметров для модулей вступают в силу после последующего перезапуска.

21.3 Таймерные прерывания (watchdog Interrupts)

Таймерное прерывание (watchdog interrupt) - это такое прерывание, которое генерируется через периодический временной интервал и инициирует запуск организационного блока обработки данного прерывания.

Таймерные прерывания позволяют Вам организовывать периодический запуск отдельной программы, независимо от времени обработки циклической (основной) программы.

В STEP 7 предусмотрены организационные блоки с OB 30 по OB 38 специально для обслуживания таймерных прерываний. Тем не менее, то, какие из данных девяти организационных блоков доступны, зависит от типа CPU.

Обработка таймерных прерываний может быть запрограммирована при конфигурировании оборудования (в данных конфигурирования оборудования) при параметризации CPU.

21.3.1 Обработка таймерных прерываний (watchdog Interrupts)

Запуск таймерных прерываний в S7-300

В S7-300 для обработки таймерного прерывания служит организационный блок OB 35, имеющий приоритет 12. При параметризации CPU Вы можете задать значение интервала времени для таймерного прерывания в диапазоне от 1 миллисекунды до 1 минуты с шагом, равным 1 миллисекунде.

Запуск таймерных прерываний в S7-400

При параметризации CPU Вы можете задать параметры таймерного прерывания. Таймерное прерывание имеет три параметра: интервал, фазовое смещение и приоритет. Вы можете определить все три параметра. Значения, которые Вы можете задать для первых двух из указанных параметров, лежат в диапазоне времени от 1 миллисекунды до 1 минуты с шагом, равным 1 миллисекунде. Значения для приоритета выбираются из диапазона значений от 2 до 24, а также возможно нулевое значение (0), в зависимости от CPU. Если приоритет приравнивается нулю, то таймерное прерывание не будет активно.

В STEP 7 предусмотрены организационные блоки, указанные в таблице 21.2 с их предельными значениями для конфигурации.

Таблица 21.2 Значения параметров, принимаемые по умолчанию для таймерных прерываний

OB	Временной интервал (Time Interval)	Фазовое смещение (Phase Offset)	Приоритет (Priority)
30	5 с	0 мс	7
31	2 с	0 мс	8
32	1 с	0 мс	9
33	500 мс	0 мс	10
34	200 мс	0 мс	11
35	100 мс	0 мс	12
36	50 мс	0 мс	13
37	20 мс	0 мс	14
38	10 мс	0 мс	15

Параметр "Фазовое смещение" (Phase Offset)

Параметр "Фазовое смещение" (Phase Offset) может быть использован для обеспечения сдвига запуска подпрограмм, запускаемых таймерными прерываниями, вне зависимости от того факта, что для этих программ в качестве параметров задано множество одинаковых временных периодов. Использование фазового смещения позволяет уточнить процесс периодического запуска прерываний.

Параметры "время старта" (start time) и "фазовое смещение" (phase offset) определяют момент перехода от режима запуска (STARTUP) в режим выполнения (RUN) организационного блока. Момент вызова OB таймерного прерывания находится как сумма временного интервала (time interval) и фазового смещения (phase offset). На рис. 21.1 показан пример, иллюстрирующий вышесказанное.

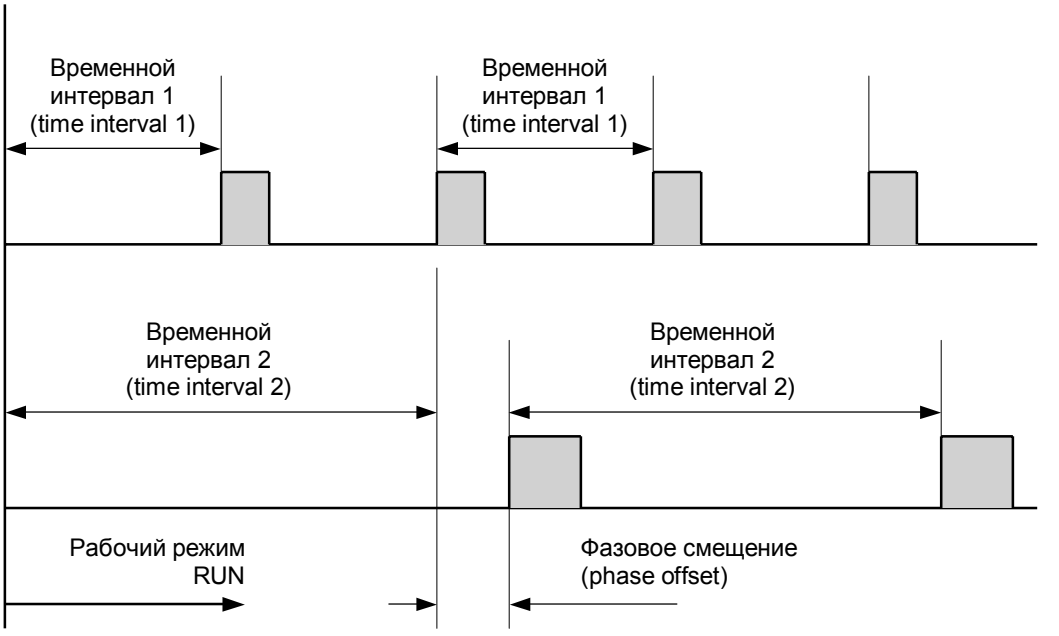


Рис. 21.1 Пример, иллюстрирующий применение параметра "фазовое смещение" (phase offset)

Как видно из рисунка, для временного интервала 1 параметр "фазовое смещение" не устанавливается (или равен 0); временной интервал 2 имеет в два раза большую величину, чем временной интервал 1. Так как временной интервал 2 имеет ненулевое фазовое смещение, то организационные блоки для временного интервала 2 и организационные блоки для временного интервала 1 не могут быть вызваны одновременно. Следовательно, OB с более низким приоритетом не будет принужден ожидать, пока выполнится OB с более высоким приоритетом, и будет обрабатываться в свой срок, точно выдерживая заданный временной интервал.

Обработка прерывания в программе запуска (start-up program)

Таймерные прерывания не могут быть обработаны в OB запуска (start-up).

Генерация прерываний с заданными временными интервалами не начнется, пока не будет включен рабочий режим (RUN).

Обработка ошибок

Если вновь генерируется то же самое таймерное прерывание, для которого в пользовательской программе уже выполняется соответствующий организационный блок, то операционная система вызывает OB 80 (организационный блок обработки временных ошибок [timing error]). Если OB 80 не запрограммирован, CPU переходит в режим STOP.

Операционная система сохраняет таймерное прерывание, которое не было обработано, чтобы обработать его позднее. При этом на каждый приоритетный класс может быть сохранено только одно необработанное таймерное прерывание независимо от того, как много накоплено необработанных таймерных прерываний.

Таймерное прерывание, которое поступило, но не было обработано из-за того, что выполнялась параметризация CPU, не может быть обработано, даже если доступен соответствующий организационный блок. В этом случае CPU переходит в режим STOP.

Блокировка (disabling), задержка во времени (delaying) и разблокировка (enabling)

Вызов организационных блоков обработки таймерных прерываний может быть заблокирован (disable) и разблокирован (enable) с помощью системных функций SFC 39 DIS_IRT и SFC 40 EN_IRT соответственно, а также задержан на время (delay) и разблокирован (enable) с помощью системных функций SFC 41 DIS_AIRT и SFC 42 EN_AIRT соответственно.

21.3.2 Конфигурирование таймерных прерываний (watchdog Interrupts) с помощью STEP 7

Обработка таймерных прерываний может быть запрограммирована при конфигурировании оборудования (в данных конфигурирования оборудования). Выберите CPU, откройте объект с помощью опций: *Edit -> Object Properties (Правка -> Свойства объекта)* и в диалоговом окне выберите вкладку "Cyclic Interrupts" ("Циклические прерывания").

Для контроллеров S7-300 установленный для блока обработки по умолчанию приоритет 12 не может быть изменен. Для систем S7-400 для всех возможных организационных блоков (в соответствии со спецификациями CPU) Вы можете устанавливать приоритет в диапазоне от 2 до 24; при этом при установлении для блока приоритета с нулевым значением (0) блок перестанет выполняться. Никогда не назначайте один и тот же приоритет дважды, так как при этом могут быть потеряны прерывания в случае, если больше, чем 12 прерываний с одинаковым приоритетом будут инициированы одновременно.

Интервалы для каждого организационного блока выбираются в разделе "Execution" ("Выполнение"), а моменты отложенного запуска (т.е., величины временной задержки) - выбираются в разделе "Phase Offset" ("Фазовое смещение").

При сохранении результатов конфигурирования оборудования STEP 7 записывает скомпилированные данные в *System data (Системные данные)* в пользовательской папке *Blocks (Блоки)*, откуда Вы сможете выгрузить данные параметризации в CPU, пока CPU находится в состоянии режима STOP. Эти данные вступают в силу немедленно после загрузки.

21.4 Прерывания по времени суток (time-of-day interrupts)

Прерывания по времени суток (time-of-day interrupts) - это такое прерывание, которое используется, когда нужно выполнять программу в заданное время суток однократно или периодически, например, ежедневно. В STEP 7 предусмотрены организационные блоки с OB 10 по OB 17 специально для обслуживания прерывания по времени суток (time-of-day interrupts). Тем не менее, то, какие из данных восьми организационных блоков доступны, зависит от типа CPU.

Обработка прерываний по времени суток может быть запрограммирована при конфигурировании оборудования (в данных конфигурирования оборудования) или же может быть выполнена посредством использования системных функций во время рабочего режима RUN. Предпосылкой правильного выполнения обработки прерываний по времени суток является корректная установка часов реального времени CPU (real-time clock).

21.4.1 Обработка прерываний по времени суток (time-of-day interrupts)

Общие замечания

Для запуска прерывания по времени суток Вы должны сначала установить время запуска (start time), а затем активировать прерывание. Вы можете инициировать прерывания по времени суток (time-of-day interrupts) отдельно посредством конфигурирования оборудования или посредством использования системных функций. Необходимо отметить, что при активации посредством конфигурирования оборудования прерывание запускается автоматически после параметризации CPU.

Вы можете запустить прерывание по времени суток (time-of-day interrupts) двумя способами:

- однократный запуск: соответствующий организационный блок вызывается один раз в заданное время;
- периодический запуск: в зависимости от значений параметров OB вызывается каждую минуту, ежечасно, ежедневно, еженедельно, ежемесячно или ежегодно.

После однократного запуска OB прерывания по времени суток данное прерывание отменяется. Вы также можете отменить прерывание по времени суток, используя системную функцию SFC 29 CAN_TINT.

Если Вы вновь хотите возобновить отмененное прерывание по времени суток, то Вы должны будете вновь установить параметр "время запуска" (start time), а затем вновь активировать прерывание.

Вы также можете запросить о состоянии прерывания по времени суток, используя системную функцию SFC 31 QRY_TINT.

Прерывания по времени суток во время запуска

Во время "холодного" (cold) перезапуска или полного (complete) перезапуска операционная система стирает все установки, сделанные посредством SFC. Установки, выполненные посредством конфигурирования оборудования сохраняются. При "теплом" (warm) перезапуске CPU продолжает обработку прерываний по времени суток в первом завершенном (complete) цикле сканирования основной программы.

Вы можете запросить о состоянии прерываний по времени суток, вызывая системную функцию SFC 31 QRY_TINT в OB запуска, и последовательно отменяя или вновь инициализируя и вновь активируя прерывания.

Прерывания по времени суток обрабатываются только в рабочем режиме RUN.

Прерывания по времени суток и ситуации сбоя (ошибки)

Если вызывается OB обработки прерывания по времени суток (time-of-day interrupts), но данный OB не запрограммирован, то операционная система вызывает OB 85 (организационный блок обработки ошибок, возникающих при выполнении программы). Если OB 85 не запрограммирован, CPU переходит в режим STOP.

Прерывание по времени суток, которое поступило, но не было обработано из-за того, что выполнялась параметризация CPU, не может быть обработано, даже если доступен соответствующий организационный блок. В этом случае CPU переходит в режим STOP.

Если Вы активировали прерывание по времени суток в режиме однократного вызова, и если при этом заданное стартовое время (start time) уже прошло на текущий момент (по часам реального времени системы), то операционная система вызовет OB 80 (организационный блок обработки временных ошибок [timing error]). Если OB 80 недоступен, то CPU перейдет в режим STOP.

Если Вы активировали прерывание по времени суток в режиме периодического вызова, и если при этом заданное стартовое время (start time) уже прошло на текущий момент (по часам реального времени системы), то операционная система сгенерирует прерывание по времени суток в следующем периоде в заданный момент времени (согласно заданным временным параметрам).

Если Вы перевели часы реального времени вперед с целью коррекции их показаний или для синхронизации так, что при этом проскочили заданное стартовое время (start time) прерывания по времени суток, то

операционная система вызовет OB 80 (организационный блок обработки временных ошибок [timing error]). После этого OB обработки прерывания по времени суток (time-of-day interrupts) будет вызван только один раз.

Если Вы перевели часы реального времени назад с целью коррекции их показаний или для синхронизации, то активированный OB обработки прерывания по времени суток (time-of-day interrupts) не будет в дальнейшем повторно выполняться в моменты времени, которые ранее уже были пройдены до перевода часов назад.

Если вызывается OB обработки прерывания по времени суток (time-of-day interrupts), для которого в пользовательской программе уже выполняется соответствующий организационный блок (режим периодического запуска), то операционная система вызывает OB 80 (организационный блок обработки временных ошибок [timing error]). Когда OB 80 и OB обработки прерывания по времени суток будут выполнены, OB обработки прерывания по времени суток будет перезапущен.

Блокировка (disabling), задержка во времени (delaying) и разблокировка (enabling)

Вызов организационных блоков обработки прерываний по времени суток может быть заблокирован (disable) и разблокирован (enable) с помощью системных функций SFC 39 DIS_IRT и SFC 40 EN_IRT соответственно, а также задержан на время (delay) и разблокирован (enable) с помощью системных функций SFC 41 DIS_AIRT и SFC 42 EN_AIRT соответственно.

21.4.2 Конфигурирование прерываний по времени суток (time-of-day interrupts) с помощью STEP 7

Обработка прерываний по времени суток может быть запрограммирована при конфигурировании оборудования (в данных конфигурирования оборудования). Выберите CPU, откройте объект с помощью опций: *Edit -> Object Properties (Правка -> Свойства объекта)* и в диалоговом окне выберите вкладку "Time-of-Day" ("Время суток").

Для контроллеров S7-300 для блока обработки установлен по умолчанию фиксированный приоритет 2. Для систем S7-400 для всех возможных организационных блоков (в соответствии со спецификациями CPU) Вы можете устанавливать значение приоритета из диапазона от 2 до 24 или равное 0; при этом при установлении для блока приоритета со значением 0 блок перестанет выполняться. Никогда не назначайте один и тот же приоритет дважды, так как при этом могут быть потеряны прерывания в случае, если больше, чем 12 прерываний с одинаковым приоритетом будут инициированы одновременно.

Опция "Active" ("Активировано") автоматически включает запуск прерывания по времени суток (time-of-day interrupts). Опция "Execution" ("Выполнение") включает отображение списка, который позволит Вам выбрать вариант выполнения прерывания по времени суток - в режиме однократного вызова или в режиме периодического вызова. Последний параметр определяет время выполнения прерывания по времени суток -

в нем задается "стартовое время" или время запуска прерывания (дата и время).

При сохранении результатов конфигурирования оборудования STEP 7 записывает скомпилированные данные в *System data* (Системные данные) в автономной пользовательской программе *Blocks* (Блоки), откуда Вы сможете выгрузить данные параметризации в CPU, пока CPU находится в состоянии режима STOP. Эти данные вступают в силу немедленно после загрузки.

21.4.3 Системные функции для прерываний по времени суток (time-of-day interrupts)

Для управления прерываниями по времени суток (time-of-day interrupts) предусмотрены следующие системные функции:

- SFC 28 SET_TINT
  функция для установки прерывания по времени суток (time-of-day interrupt);
- SFC 29 CAN_TINT
  функция для отмены прерывания по времени суток (time-of-day interrupt);
- SFC 30 ACT_TINT
  функция для активации прерывания по времени суток (time-of-day interrupt);
- SFC 31 QRY_TINT
  функция для запроса о состоянии прерывания по времени суток (time-of-day interrupt).

Параметры для вышеуказанных системных функций представлены в таблице 21.3.

SFC 28 SET_TINT
Установка прерывания по времени суток (time-of-day interrupt)

Вы можете определить стартовое время (время запуска) для прерывания по времени суток, вызвав системную функцию SFC 28 SET_TINT. Функция SFC 28 SET_TINT позволяет только задать параметр "стартовое время" ("start time"); для того чтобы запустить прерывание по времени суток, Вы должны будете еще и активировать это прерывание, вызвав системную функцию SFC 30 ACT_TINT. "Стартовое время" определяется в параметре SDT в формате DATE_AND_TIME, например, DT#1997-06-30-08:30. Операционная система игнорирует секунды и миллисекунды и устанавливает для них нулевые значения. При установке стартового времени для прерывания по времени суток старое значение, если оно было задано, переписывается новым значением.

При этом активность прерывания по времени суток аннулируется, что означает, что Вы после установки стартового времени снова должны будете активизировать прерывание с помощью функции SFC 30 ACT_TINT.

Таблица 21.3 Параметры для системных функций для управления прерываниями по времени суток (time-of-day interrupts)

OB	Параметр	Описание	Тип данных	Содержание, описание
28	OB_NR	INPUT	INT	Номер OB для вызова в заданное время (однократно/периодически)
	SDT	INPUT	DT	Дата и время запуска в формате DATE_AND_TIME
	PERIOD	INPUT	WORD	Период запуска прерывания (базовый период для времени запуска [start time]): W#16#0000 = однократный запуск W#16#0201 = запуск каждую минуту W#16#0401 = запуск каждый час W#16#1001 = запуск каждый день W#16#1201 = запуск каждую неделю W#16#1401 = запуск каждый месяц W#16#2001 = запуск на протяжении месяца W#16#1801 = запуск каждый год
	RET_VAL	OUTPUT	INT	Информация об ошибках
29	OB_NR	INPUT	INT	Номер OB, стартовое время которого необходимо отменить.
	RET_VAL	OUTPUT	INT	Информация об ошибках
30	OB_NR	INPUT	INT	Номер OB, который необходимо активировать.
	RET_VAL	OUTPUT	INT	Информация об ошибках
31	OB_NR	INPUT	INT	Номер OB, о состоянии которого необходимо сделать запрос.
	RET_VAL	OUTPUT	INT	Информация об ошибках
	STATUS	OUTPUT	WORD	Состояние прерывания по времени суток (time-of-day interrupt)

SFC 30 ACT_TINT
Активация прерывания по времени суток (time-of-day interrupt)

Вы можете активировать прерывание по времени суток, вызвав системную функцию SFC 30 ACT_TINT. Если прерывание активировано, то считается, что параметр "стартовое время" ("start time") установлен для прерывания по времени суток. Если Вы активировали прерывание по времени суток в режиме однократного вызова, и если при этом заданное стартовое время (start time) уже прошло на текущий момент, то SFC 30 выдаст сообщение об ошибке. Если Вы активировали прерывание в режиме периодического вызова, и если при этом заданное стартовое время уже прошло, то операционная система вызовет для обработки соответствующий OB в следующем подходящем периоде в заданный момент времени. Если прерывание по времени суток в режиме однократного вызова на текущий момент обслужено, то это прерывание для дальнейшего применения более недоступно. Если требуется опять его использовать (с другим значением стартового времени) Вы должны его переустановить и затем вновь активировать.

SFC 29 CAN_TINT
Отмена прерывания по времени суток (time-of-day interrupt)

Вы можете отменить стартовое время (время запуска) и таким образом

деактивировать прерывание по времени суток, вызвав системную функцию SFC 29 CAN_TINT. При этом соответствующий OB прерывания по времени суток не будет далее вызываться. Если требуется опять использовать такое же прерывание по времени суток, Вы должны переустановить стартовое время, а затем вновь активировать прерывание.

SFC 31 QRY_TINT
Запрос о состоянии прерывания по времени суток (time-of-day interrupt)

Вы можете запросить информацию о состоянии прерывания по времени суток, вызвав системную функцию SFC 31 QRY_TINT. Функция SFC 31 QRY_TINT возвращает требуемую информацию в параметре STATUS.

Если соответствующий бит имеет сигнал со значением "1", то это означает:

0 - прерывание по времени суток заблокировано операционной системой;

1 - новое прерывание по времени суток отвергнуто;

2 - прерывание по времени суток не активировано и не завершено;

3 - (- зарезервирован -);

4 - загружен OB прерывания по времени суток;

5 - прерывание по времени суток не заблокировано;

6 - (и последующие - зарезервированы -).

21.5 Прерывания с задержкой обработки (time-delay interrupts)

Прерывания с задержкой обработки (time-delay interrupts) - это такое прерывание, которое позволяет обеспечить выполнение функции таймера задержки без использования стандартных функций таймера. В STEP 7 предусмотрены организационные блоки с OB 20 по OB 23 специально для обслуживания прерывания с задержкой обработки (time-delay interrupts). Тем не менее, то, какие из данных четырех организационных блоков доступны, зависит от типа CPU.

Приоритеты OB обработки прерываний с задержкой обработки программируются при конфигурировании оборудования (в данных конфигурирования оборудования); системные функции используются для целей управления.

21.5.1 Обработка прерываний с задержкой обработки (time-delay interrupts)

Общие замечания

Прерывание с задержкой обработки запускается при вызове системной

функции SFC 32 SRT_DINT; данная системная функция передает операционной системе значение временного интервала и номер выбранного организационного блока. После того, как истекает заданный временной интервал, происходит вызов определенного организационного блока.

Вы можете отменить обслуживание прерывания с задержкой обработки (time-delay interrupts) посредством системной функции SFC 33 CAN_DINT, в результате чего связанный с этим прерыванием OB не будет более вызываться.

Вы можете также запросить информацию о состоянии прерывания с задержкой обработки, используя системную функцию SFC 34 QRY_DINT.

Прерывания с задержкой обработки во время запуска

Во время "холодного" (cold) перезапуска или полного (complete) перезапуска операционная система стирает все установки, сделанные для прерывания с задержкой обработки (time-delay interrupts). При "теплом" (warm) перезапуске установки сохраняются, пока идет обработка в рабочем режиме RUN, в то время как "оставшаяся часть" цикла считается частью подпрограммы запуска (start-up).

Вы можете запускать прерывание с задержкой обработки в подпрограмме запуска (start-up), вызывая системную функцию SFC 32. В момент, когда будет закончен временной интервал прерывания (интервал отсчета задержки) CPU должен находиться в состоянии рабочего режима RUN, для того чтобы обеспечить выполнение связанного с прерыванием организационного блока. Если это условие не выполняется, CPU будет ждать окончания выполнения подпрограммы запуска (start-up), и лишь затем вызовет для выполнения связанный с прерыванием организационный блок; все это происходит до обработки первого сегмента основной программы.

Прерывания с задержкой обработки и ситуации сбоя (ошибки)

Если вызывается OB обработки прерывания с задержкой обработки (time-delay interrupts), но данный OB не запрограммирован, то операционная система вызывает OB 85 (организационный блок обработки ошибок, возникающих при выполнении программы). Если OB 85 не запрограммирован, CPU переходит в режим STOP. Если истек временной интервал (интервал отсчета задержки) и вызывается OB обработки данного прерывания, для которого в пользовательской программе уже выполняется соответствующий организационный блок, то операционная система вызывает OB 80 (организационный блок обработки временных ошибок [timing error]) или переходит в режим STOP, если OB 80 недоступен в пользовательской программе.

Прерывание с задержкой обработки, которое поступило, но не было обработано из-за того, что выполнялась параметризация CPU, не сможет быть обработано, даже если соответствующий организационный блок был запрограммирован. В этом случае CPU переходит в режим STOP.

Блокировка (disabling), задержка (delaying) и разблокировка (enabling)

Вызов организационных блоков обработки прерываний с задержкой

обработки может быть заблокирован (disable) и разблокирован (enable) с помощью системных функций SFC 39 DIS_IRT и SFC 40 EN_IRT соответственно, а также задержан на время (delay) и разблокирован (enable) с помощью системных функций SFC 41 DIS_AIRT и SFC 42 EN_AIRT соответственно.

21.5.2 Конфигурирование прерываний с задержкой обработки (time-delay interrupts) с помощью STEP 7

Обработка прерываний с задержкой обработки может быть запрограммирована при конфигурировании оборудования (в данных конфигурирования оборудования). Выберите CPU, откройте объект с помощью опций: *Edit -> Object Properties (Правка -> Свойства объекта)* и в диалоговом окне выберите вкладку "Interrupts" ("Прерывания").

Для контроллеров S7-300 (за исключением CPU 318) для блока обработки прерывания установлен по умолчанию фиксированный приоритет 3. Для систем S7-400 (а также для CPU 318) для всех возможных организационных блоков (в соответствии со спецификациями CPU) Вы можете устанавливать значение приоритета из диапазона от 2 до 24 или равное 0; при этом при установлении для блока приоритета со значением 0 блок перестанет выполняться. Никогда не назначайте один и тот же приоритет дважды, так как при этом могут быть потеряны прерывания в случае, если больше, чем 12 прерываний с одинаковым приоритетом будут инициированы одновременно.

При сохранении результатов конфигурирования оборудования STEP 7 записывает скомпилированные данные в *System data* (Системные данные) в автономной пользовательской программе *Blocks* (Блоки), откуда Вы сможете выгрузить данные параметризации в CPU, пока CPU находится в состоянии режима STOP. Эти данные вступают в силу немедленно после загрузки.

21.5.3 Системные функции для прерываний с задержкой обработки (time-delay interrupts)

Для управления прерываниями с задержкой обработки (time-delay interrupts) предусмотрены следующие системные функции:

- SFC 32 SRT_DINT
  функция для установки задержки запуска OB прерывания;
- SFC 33 CAN_DINT
  функция для отмены прерывания с задержкой обработки (time-of-day interrupt);
- SFC 34 QRY_TINT
  функция для запроса о состоянии прерывания с задержкой обработки (time-of-day interrupt).

Параметры для вышеуказанных системных функций представлены в таблице 21.4.

Таблица 21.4 Параметры для системных функций для управления прерываниями с задержкой обработки (time-delay interrupts)

OB	Параметр	Описание	Тип данных	Содержание, описание
32	OB_NR	INPUT	INT	Номер OB для вызова после истечения отсчета интервала задержки
	DTIME	INPUT	TIME	Временной интервал задержки (допустимые значения - в диапазоне: T#1ms…T#1m)
	SIGN	INPUT	WORD	Идентификатор задания в стартовой информации соответствующего OB при его вызове (произвольные символы)
	RET_VAL	OUTPUT	INT	Информация об ошибках
33	OB_NR	INPUT	INT	Номер OB, вызов которого необходимо отменить.
	RET_VAL	OUTPUT	INT	Информация об ошибках
34	OB_NR	INPUT	INT	Номер OB, о состоянии которого необходимо сделать запрос.
	RET_VAL	OUTPUT	INT	Информация об ошибках
	STATUS	OUTPUT	WORD	Состояние прерывания с задержкой обработки (time-of-day interrupt)

SFC 32 SRT_DINT
Запуск прерывания с задержкой обработки (time-delay interrupt)

Прерывание с задержкой обработки (time-delay interrupt) запускается с помощью системной функции SFC 32 SRT_DINT. При вызове функции SFC 32 SRT_DINT начинается отсчет времени запрограммированной задержки (периода времени). Как только истекает заданный период временной задержки, CPU вызывает запрограммированный OB и передает значение временной задержки и идентификатор задания в область стартовой информации этого блока.

Идентификатор задания определяется в параметре SIGN для системной функции SFC 32. Вы можете прочитать такое же значение в байтах 6 и 7 в области стартовой информации связанного с прерыванием с задержкой обработки блока OB. Значение задержки (периода времени) устанавливается с шагом приращения, равным 1 мс. Точность для параметра "задержка обработки" также составляет 1 мс.

Необходимо отметить, что процесс выполнения блока OB, связанного с прерыванием с задержкой обработки, может быть сам задержан, если организационные блоки с более высокими приоритетами начнут обрабатываться в тот же период времени, что и вызванный OB, связанный с прерыванием с задержкой обработки (time-delay interrupt).

Вы можете изменить на новое значение величину временной задержки с помощью SFC 32. Новое значение параметра вступит в силу сразу после вызова SFC.

SFC 33 CAN_DINT
Отмена прерывания с задержкой обработки (time-delay interrupt)

Вы можете деактивировать прерывание с задержкой обработки (time-delay interrupt), вызвав системную функцию SFC 33 CAN_DINT. При этом соответствующий OB, связанный с прерыванием с задержкой обработки не будет больше вызываться.

SFC 34 QRY_DINT
Запрос о состоянии прерывания с задержкой обработки (time-delay interrupt)

Вы можете запросить информацию о состоянии прерывания с задержкой обработки, вызвав системную функцию SFC 34 QRY_DINT. Прерывание при этом выбирается с помощью параметра OB_NR (номер блока). Функция SFC 34 QRY_DINT возвращает требуемую информацию в параметре STATUS.

Если соответствующий бит содержит сигнал со значением "1", то это означает следующее:

0 - прерывание с задержкой обработки заблокировано операционной системой;

1 - новое прерывание с задержкой обработки отвергнуто;

2 - прерывание с задержкой обработки не активировано и не завершено;

3 - (- зарезервирован -);

4 - загружен OB прерывания с задержкой обработки;

5 - прерывание с задержкой обработки не заблокировано;

6 - (и последующие - зарезервированы -).

21.6 Прерывание мультипроцессорного режима

Прерывание мультипроцессорного режима позволяет обеспечить синхронную реакцию на событие во всех CPU, работающих в мультипроцессорном режиме. Прерывание мультипроцессорного режима запускается с помощью SFC 35 MP_ALM. Для обслуживания прерывания мультипроцессорного режима предусмотрен организационный блок OB 60, для которого установлен фиксированный приоритет, равный 25.

Общие замечания

Вызов системной функции SFC 35 MP_ALM вызывает выполнение организационного блока обработки прерывания мультипроцессорного режима. Если CPU работает в однопроцессорном режиме, то организационный блок OB 60 запускается немедленно. Если режим работы мультипроцессорный, то блок OB 60 запускается одновременно на всех CPU мультипроцессорной системы, что означает, что CPU, в котором вызвана функция SFC 35 отложит запуск блока OB 60 до момента, когда все другие CPU сообщат о своей готовности.

Обработка прерываний с задержкой обработки не программируется при конфигурировании оборудования (в данных конфигурирования оборудования) - эта функция заложена в каждом CPU, предназначенном для работы в мультипроцессорном режиме. Тем не менее, не смотря на это, минимально необходимое количество байтов в области локальных данных (по крайней мере, 20 байтов) должно быть все же зарезервировано на вкладке "Local Data" CPU для приоритетного класса 25.

Прерывание мультипроцессорного режима во время запуска

Прерывание мультипроцессорного режима запускается только в рабочем режиме RUN. Вызов системной функции SFC 35 MP_ALM при выполнении подпрограммы запуска (start-up) заканчивается с возвращением кода ошибки 32929 (W#16#80A1) в качестве значения функции.

Прерывание мультипроцессорного режима и ситуации сбоя (ошибки)

В случае, если вызывается функция SFC 35 в то время, когда OB 60 все еще обрабатывается, системная функция возвращает код ошибки 32928 (W#16#80A0) в качестве значения функции. И при этом организационный блок OB 60 не запускается ни на одном CPU.

Если блок OB 60 недоступен в одном из CPU во время вызова этого организационного блока или блокирован (disable), или задержано выполнение (delay) этого OB посредством системных функций, то функция SFC 35 не вызывает никаких действий в данном CPU и не сообщает при этом об ошибке.

Блокировка (disabling), задержка (delaying) и разблокировка (enabling)

Вызов организационного блока обработки прерывания с задержкой обработки может быть заблокирован (disable) и разблокирован (enable) с помощью системных функций SFC 39 DIS_IRT и SFC 40 EN_IRT соответственно, а также задержан на время (delay) и разблокирован (enable) с помощью системных функций SFC 41 DIS_AIRT и SFC 42 EN_AIRT соответственно.

SFC 35 MP_ALM
Запуск прерывания мультипроцессорного режима

Прерывание мультипроцессорного режима запускается с помощью системной функции SFC 35 MP_ALM. Параметры функции приведены в таблице 21.5.

Параметр JOB позволяет передать идентификатор задания в область стартовой информации блока. Вы можете прочитать такое же значение в байтах 6 и 7 в области стартовой информации связанных с прерыванием мультипроцессорного режима блоков OB во всех участвующих CPU.

Таблица 21.5 Параметры для системной функции SFC 35 MP_ALM

Параметр	Описание	Тип данных	Содержание, описание
JOB	INPUT	BYTE	Идентификатор задания (значения из диапазона: B#16#00 … B#16#0F)
RET_VAL	OUTPUT	INT	Информация об ошибках

21.7 Обработка прерываний

Для управления прерываниями и асинхронными ошибками предусмотрены следующие системные функции:

- SFC 39 DIS_IRT
  функция для блокировки (disable) запуска OB прерывания;
- SFC 40 EN_IRT
  функция для отмены блокировки (enable) запуска OB прерывания;
- SFC 41 DIS_AIRT
  функция для задержки на время (delay) запуска OB прерывания;
- SFC 42 EN_AIRT
  функция для отмены задержки (enable) запуска OB прерывания.

Параметры для вышеуказанных системных функций представлены в таблице 21.6.

Данные системные функции имеют воздействие на обработку всех прерываний и на обработку всех асинхронных ошибок. Системные функции SFC 36 … SFC 38 обеспечивают управление обработкой синхронных ошибок (см. главу 23 "Обработка ошибок").

Таблица 21.6 Параметры для системных функций для управления прерываниями с задержкой обработки (time-delay interrupts)

OB	Параметр	Описание	Тип данных	Содержание, описание
39	MODE	INPUT	BYTE	Режим блокировки (Disable mode) (см. текст)
	OB_NR	INPUT	INT	Номер OB (см. текст)
	RET_VAL	OUTPUT	INT	Информация об ошибках
40	MODE	INPUT	BYTE	Режим разблокирования (Enable mode) (см. текст)
	OB_NR	INPUT	INT	Номер OB (см. текст)
	RET_VAL	OUTPUT	INT	Информация об ошибках
41	RET_VAL	OUTPUT	INT	(Новое) число вызовов функции задержки
42	RET_VAL	OUTPUT	INT	Оставшееся число вызовов задержек

SFC 39 DIS_IRT
Блокировка (disable) запуска OB прерывания

Системная функция SFC 39 DIS_IRT позволяет заблокировать обслуживание новых прерываний и асинхронных ошибок. Запросы на обработку всех новых прерываний и асинхронных ошибок отбрасываются. Если прерывание или асинхронная ошибка имеет место после вызова SFC 39 DIS_IRT, соответствующий организационный блок не выполняется; если этот организационный блок не существует, то CPU не переходит в режим STOP.

Функция SFC 39 (disable) остается в силе для всех приоритетных классов до тех пор, пока ее действие не будет отменено с помощью вызова системной функции SFC 40 EN_IRT. После полного перезапуска обработка всех прерываний и асинхронных ошибок разблокируется.

Параметры MODE и OB_NR используются для определения, обработка какого из прерываний или обработка какой из асинхронных ошибок должна быть заблокирована. Если параметр MODE = B#16#00, то блокируется обработка всех прерываний и асинхронных ошибок. Если параметр MODE = B#16#01, то блокируется обработка класса прерываний, первый номер OB которого определен в параметре OB_NR.

Например, если MODE = B#16#01 и OB_NR = 40, то в этом случае блокируется обработка всех аппаратных прерываний; если OB_NR = 80, то в этом случае блокируется обработка всех асинхронных ошибок. Если MODE = B#16#02, то блокируется обработка всех прерываний или всех асинхронных ошибок, первый номер OB которых Вы определите в параметре OB_NR.

Вне зависимости от того, активна функция SFC 39 (disable) или не активна, операционная система вносит каждое новое прерывание или асинхронную ошибку в диагностический буфер.

SFC 40 EN_IRT
Отмена блокировки (разблокировка - enable) запуска OB прерывания

Системная функция SFC 40 EN_IRT позволяет разблокировать обслуживание новых прерываний и асинхронных ошибок, если они ранее были заблокированы с помощью функции SFC 39 DIS_IRT. Все новые прерывания и асинхронные ошибки после вызова функции SFC 40 EN_IRT будут вызывать для обработки соответствующие организационные блоки; если этот организационный блок не существует в пользовательской программе, то CPU перейдет в режим STOP (кроме случая, когда этот организационный блок является OB 81, т.е. является блоком обработки ошибок источника питания).

Параметры MODE и OB_NR используются для определения, обработка какого из прерываний или обработка какой из асинхронных ошибок должна быть разблокирована. Если параметр MODE = B#16#00, то разрешается обработка всех прерываний и асинхронных ошибок. Если параметр MODE = B#16#01, то разблокируется обработка класса прерываний, первый номер OB которого определен в параметре OB_NR. Если MODE = B#16#02, то разблокируется обработка всех прерываний или всех асинхронных ошибок, первый номер OB которых Вы определили в параметре OB_NR.

SFC 41 DIS_AIRT
Задержка на время (delay) запуска OB прерывания

Системная функция SFC 41 DIS_AIRT позволяет отложить на время (delay) обслуживание новых прерываний и асинхронных ошибок. "Задержка на время" означает, что операционная система сохраняет информацию о всех новых прерываниях и асинхронных ошибках, которые имели место после вызова функции SFC 41 DIS_AIRT, и обслуживает их после того, как истечет временной интервал (период) задержки. Если была вызвана функция SFC 41, то программа в текущем организационном блоке (в текущем приоритетном классе) не будет прерываться при приходе прерываний с более высоким приоритетом; при этом прерывания и асинхронные ошибки не будут потеряны.

Функция SFC 41 (delay) остается в силе, пока не завершится выполнение текущего OB, или пока действие функции не будет отменено с помощью вызова системной функции SFC 42 EN_AIRT.

Вы можете вызывать функцию SFC 41 последовательно несколько раз. Параметр RET_VAL показывает общее число вызовов. Вы должны при этом обеспечить точно такое же число вызовов функции SFC 42, сколько их было для функции SFC 41, чтобы снять блокировку с обработки всех прерываний и асинхронных ошибок.

SFC 42 EN_AIRT
Отмена задержки (разблокировка - enable) запуска OB прерывания.

Системная функция SFC 42 EN_AIRT позволяет разрешить обслуживание новых прерываний и асинхронных ошибок, которое было заблокировано после вызова функции SFC 41 DIS_AIRT. Вы должны при этом обеспечить точно такое же число вызовов функции SFC 42, сколько их было для функции SFC 41 (в текущем организационном блоке). Параметр RET_VAL показывает число вызовов функции SFC 41 DIS_AIRT, которые остаются в силе; если RET_VAL = 0, то все прерывания и асинхронные ошибки разблокированы (будут обслужены без задержки).

Если была вызвана функция SFC 42 (enable), и при этом функция SFC 41 (delay) ранее не вызывалась, то параметр RET_VAL будет содержать значение 32896 (W#16#8080).

22 Параметры перезапуска

22.1 Общие замечания

22.1.1 Режимы работы

После включения электропитания перед началом выполнения основной программы (main program) CPU выполняет подпрограмму перезапуска (restart routine). Режим запуска START-UP (ЗАПУСК) является одним из режимов работы CPU, наряду с режимами STOP (СТОП) и RUN (РАБОТА). Эта глава описывает поведение CPU при переходе от режима START-UP и, наоборот, к режиму START-UP, а также при выполнении собственно подпрограммы перезапуска (restart routine).

После включения ***(1)*** электропитания CPU находится в режиме STOP (рис.22.1). Если на передней панели CPU переключатель стоит в положении RUN или в положении RUN-P, то CPU переключается в режим START-UP ***(2)***, а после этого - в режим RUN ***(3)***. Если происходит "неисправимая" ошибка, пока CPU находится в режиме START-UP или RUN, или если Вы переключите переключатель на передней панели CPU в положение STOP, то CPU переключается в режим STOP ***(4)*** ***(5)***.

Пользовательская программа может быть протестирована в пошаговом режиме с использованием "точек прерывания" в рабочем режиме HOLD (ПАУЗА). Вы можете переключаться в этот режим и из режима RUN, и из режима START-UP, а затем, после окончания тестирования программы, Вы можете переключиться опять в исходные режимы ***(6)*** ***(7)***. Вы также можете переключить CPU в режим STOP из режима HOLD ***(8)***.

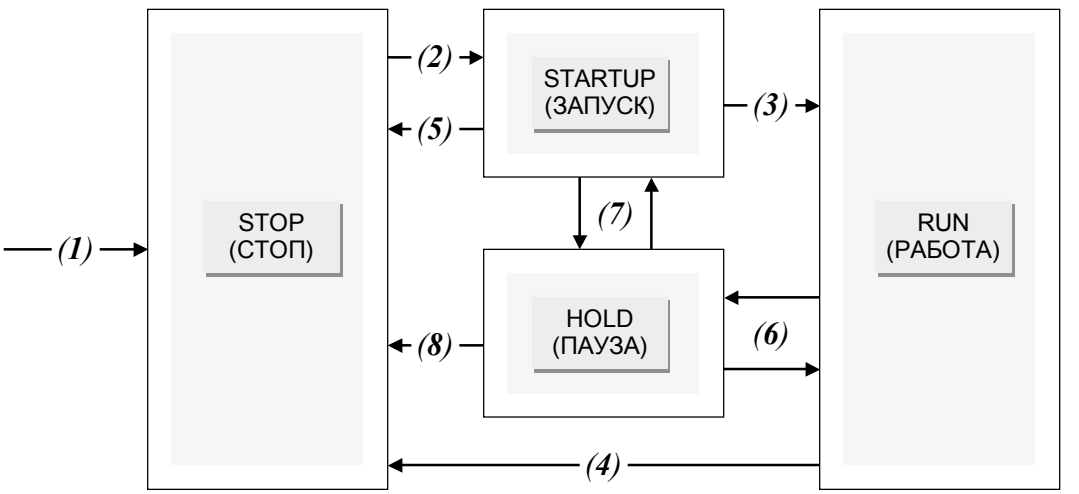


Рис. 22.1 Режимы работы CPU

При параметризации CPU на вкладке "Restart" ("Перезапуск") Вы можете определять параметры перезапуска, такие как максимально разрешенное количество времени для сигналов "Ready" ("Готов") от модулей после включения питания, а также указывать, должен ли CPU запускаться, если данные конфигурации не согласуются с "фактической конфигурацией", или определять режим перезапуска CPU.

SIMATIC S7 предоставляет пользователю следующие три режима перезапуска, а именно: "холодный" перезапуск *(cold restart),* "полный" перезапуск *(complete restart)* и "теплый" перезапуск *(warm restart).* При "холодном" перезапуске и "полном" перезапуске основная программа всегда выполняется с начала. При "теплом" перезапуске основная программа продолжает выполняться с той точки, в которой ее выполнение было прервано, и она "заканчивает" прерванный цикл выполнения. Другие прерывания вызываются такими событиями, как синхронные ошибки, которые могут происходить при сканировании программы, и асинхронные ошибки, которыми могут являться, например, диагностические прерывания. Обработка этих событий обсуждается в главе 23 "Обработка ошибок".

В S7 CPU выпуска до октября 1998 г. предусмотрены "полный" перезапуск *(complete restart)* и "теплый" перезапуск *(warm restart).* При этом функционально "полный" перезапуск соответствует "теплому" перезапуску.

Вы можете однократно просканировать подпрограмму запуска START-UP. Специально для этих целей в STEP 7 имеются организационные блоки - OB 102 (для "холодного" перезапуска), OB 100 (для "полного" перезапуска) и OB 101 (для "теплого" перезапуска). В блоках уже проведена параметризация модулей, кроме CPU и запрограммированы установки "по умолчанию" для Вашей основной программы.

22.1.2 Режим HOLD (ПАУЗА)

CPU переключается в режим HOLD (ПАУЗА) в случае, когда Вы тестируете пользовательскую программу с использованием "точек прерывания" (в пошаговом режиме). При этом должен светиться светодиодный индикатор "STOP", а индикатор "RUN" должен мигать.

В режиме HOLD (ПАУЗА) выходные модули заблокированы. Запись в эти модули приводит к изменения состояний сигналов в памяти, но не переключает состояний сигналов на выходах модулей. При этом модули не могут быть вновь разблокированы, пока не будет выключен режим HOLD.

В режиме HOLD (ПАУЗА) все процессы, в которых присутствует функция измерения времени, не функционируют (кроме функции часов реального времени). К этим функциям относятся таймеры, тактовые меркеры, счетчики отработанного времени, функции отслеживания времени цикла и минимального времени цикла сканирования, а также обслуживание прерывания по времени суток и прерывания с задержкой обработки.

Исключение: функция часов реального времени в этом режиме продолжает нормально работать.

При каждом последующем переходе к следующему оператору в программе, обрабатываемой в режиме "пошагового" ("single step") тестирования, значения таймеров инкрементируются в соответствии с величиной шага - таким образом имитируется динамическое поведение программы при "обычном" ее сканировании.

В режиме HOLD (ПАУЗА) CPU способен быть "пассивным" коммуникационным партнером, что означает, что он может принимать глобальные данные или принимать участие в одностороннем обмене данными.

При сбое электропитания в то время, когда CPU находится в режиме HOLD, CPU с резервной батареей питания при восстановлении нормального электропитания переходят в режим STOP, тогда как CPU без резервной батареи автоматически выполняют полный перезапуск.

22.1.3 Блокировка выходных модулей (disable)

В режимах STOP и HOLD модули заблокированы (заблокированы выходы, OD-сигнал). Заблокированные модули имеют на своих выходах "нулевой" сигнал (сигнал нулевой величины) или, если они оснащены такой функцией, то имеют на своих выходах так называемый "сигнал замещения" (или значение замещения). С помощью таблицы переменных Вы можете управлять выходами модулей, используя функцию "Enable Peripheral Outputs" ("Разблокировать периферийные выходы") даже в режиме STOP.

Во время перезапуска выходные модули остаются в заблокированном состоянии. Только после того, как начинается цикл сканирования основной программы, выходные модули разблокируются. Состояния сигналов из памяти модулей (но не в области образа процесса!) поступают на выходы.

В то время, пока выходные модули остаются в заблокированном состоянии, отдельные биты памяти модуля могут быть установлены или посредством прямого доступа (с помощью операторов передачи [transfer] с адресацией области PQ), или посредством передачи таблицы выходов образа процесса. Если CPU отменяет сигнал блокировки (Disable), то состояния сигналов из памяти модуля поступают на внешние выходы модуля.

При "холодном" перезапуске (OB 102) и "полном" перезапуске (OB 100) данные образа процесса и данные в памяти модуля стираются. Если необходимо сканировать входы в OB 102 или в OB 100, Вы сначала должны загрузить состояния сигналов из модуля, используя прямой доступ. После этого Вы можете устанавливать входы (переносить их, например, с помощью операторов загрузки [load] из адресной области PI в адресную область I), а затем работать со входами. Если необходимо установить определенные выходы при переходе от полного перезапуска к циклическому сканированию программы (перед вызовом OB 1), то Вы должны использовать прямой доступ при адресации выходных модулей.

В данном случае не будет достаточно только лишь установить выходы (в области образа процесса), так как таблица выходов образа процесса не пересылается в модули в конце подпрограммы полного перезапуска.

При "теплом" перезапуске "старые" таблицы выходов и входов образа процесса, которые были действительными на момент выключения электропитания установки или при переводе ее в режим STOP, используются в OB 101 и с момента возобновления до окончания цикла сканирования программы. В конце этого цикла таблица выходов образа процесса пересылается в память модуля (но при этом еще не вызывает изменение выходных сигналов модуля на его внешних выходах, так как в это время выходные модули пока еще находятся в заблокированном состоянии).

Вы можете провести параметризацию CPU, чтобы очистить таблицу выходов образа процесса и память модуля в конце процедуры "теплого" перезапуска. Перед переходом к обработке OB 1 CPU отменяет сигнал блокировки (Disable), после этого состояния сигналов из памяти модуля передаются на его внешние выходы.

22.1.4 Организационные блоки для перезапуска

При "холодном" перезапуске CPU вызывает организационный блок OB 102; при "полном" перезапуске CPU вызывает организационный блок OB 100. При отсутствии блока OB 100 или блока OB 102 CPU немедленно начинает выполнять цикл сканирования программы.

При "теплом" перезапуске CPU однократно вызывает организационный блок OB 101 перед началом обработки основной программы. При отсутствии организационного блока OB 101 CPU продолжает выполнять сканирование программы с точки прерывания ее выполнения в последний раз.

Стартовая информация во временных локальных данных имеет одинаковый формат для всех организационных блоков. В таблице 22.1 представлена стартовая информация блока OB 100. Причина перезапуска указывается в запросе на перезапуск (байт 1):

- B#16#81 - ручной режим "полного" перезапуска (OB 100)
- B#16#82 - автоматический режим "полного" перезапуска (OB 100)
- B#16#83 - ручной режим "теплого" перезапуска (OB 101)
- B#16#84 - автоматический режим "теплого" перезапуска (OB 101)
- B#16#85 - ручной режим "холодного" перезапуска (OB 102)
- B#16#86 - автоматический режим "холодного" перезапуска (OB 102)

Номер события остановки (STOP) и дополнительная информация более точно определяют событие перезапуска (они сообщают, например, о том, был ли ручной "холодный" перезапуск инициирован с помощью переключателя режимов). С помощью этой информации Вы можете соответствующим образом настроить обусловленную событиями подпрограмму перезапуска.

Таблица 22.1 Стартовая информация для OB перезапуска

Байт	Имя	Тип данных	Описание
0	OB100_EV_CLASS	BYTE	Класс события
1	OB100_STRTUP	BYTE	Запрос на перезапуск (см. текст)
2	OB100_PRIORITY	BYTE	Приоритетный класс
3	OB100_OB_NUMBR	BYTE	Номер OB
4	OB100_RESERVED_1	BYTE	Зарезервирован
5	OB100_RESERVED_2	BYTE	Зарезервирован
6..7	OB100_STOP	WORD	Номер события остановки (STOP)
8..11	OB100_STRT_INFO	DWORD	Дополнительная информация по текущему перезапуску
12..19	OB100_DATE_TIME	DT	Дата и время свершения события

22.2 Включение питания (Power-Up)

22.2.1 Режим STOP (СТОП)

CPU переходит в режим STOP в следующих случаях:

- когда CPU выключается;
- когда переключатель режимов переключается с режима RUN (РАБОТА) в режим STOP (СТОП);
- когда во время сканирования программы происходит "неустранимая" ошибка;
- когда выполняется системная функция SFC 46 STP;
- когда приходит запрос на остановку CPU от коммуникационной функции (запрос на переход в режим STOP (СТОП) от программатора или коммуникационного функционального блока от другого CPU).

CPU записывает причину перехода в режим STOP в диагностический буфер. Находясь в данном режиме, Вы можете также прочитать информацию CPU с помощью программатора, чтобы локализовать источник возможной проблемы.

В режиме STOP (СТОП) пользовательская программа не сканируется. CPU восстанавливает установки: или значения, которые Вы задавали при конфигурировании оборудования при параметризации CPU, или установки, принимаемые по умолчанию, и устанавливает модули в определенное исходное состояние.

В режиме STOP (СТОП) CPU может принимать глобальные данные посредством GD-коммуникаций и выполнять односторонний "пассивный" обмен данными. В режиме STOP часы реального времени продолжают функционировать.

Вы можете параметризировать CPU в режиме STOP, например, Вы можете установить MPI-адрес, переслать или модифицировать программу

пользователя или выполнить сброс памяти CPU.

22.2.2 Сброс памяти (Memory Reset)

Сброс памяти (Memory Reset) приводит CPU к исходному состоянию. Вы можете инициировать сброс памяти, используя программатор только в режиме STOP (СТОП) или при помощи переключателя режимов работы. Для этого необходимо удерживать в положении MRES по крайней мере 3 секунды, затем - отпустить максимум на 3 секунды и вновь удерживать в положении MRES по крайней мере 3 секунды.

CPU при сбросе памяти стирает пользовательскую программу целиком - и в рабочей (work) и в загрузочной (load) RAM-памяти. Информация в системной памяти (меркеры, таймеры и счетчики) также стирается независимо от установок реманентности.

CPU при сбросе памяти устанавливает параметры всех модулей (включая и свои собственные) в состояния, принятые для них как "значения по умолчанию". Как исключение, только параметры для MPI-подсети не изменяются, поэтому к CPU, память которого была "сброшена", все еще остается возможным доступ посредством MPI-шины. Также сброс памяти не влияет на диагностический буфер, на часы реального времени и измерители времени наработки (часы учета рабочего времени).

Если используется модуль памяти Flash EPROM, то CPU копирует пользовательскую программу из модуля памяти в рабочую (work) память. CPU также копирует все данные конфигурации, которые он сможет найти в модуле памяти.

22.2.3 Реманентность (Retentivity)

Область памяти является реманентной, если ее содержимое сохраняется, при выключении источника питания, также как и при переходе от режима STOP (СТОП) к режиму RUN (РАБОТА) после включения электропитания (при "холодном" перезапуске и при "теплом" перезапуске).

Реманентной область памяти может быть для меркеров, таймеров, счетчиков и, для S7-300, для блоков данных. Размер области данных, которая может быть объявлена реманентной, зависит от типа CPU. Вы можете определить области данных как реманентные на вкладке "Retentivity" ("Реманентность") при параметризации CPU.

Установки реманентности сохраняются в блоках системных данных (SDB) в загрузочной (load) памяти, то есть в модуле памяти. Если модуль памяти имеет тип RAM, то для постоянного сохранения там данных Вы должны использовать резервную батарею.

При работе с резервной батареей состояния всех сигналов меркеров, таймеров, счетчиков, сохраняются. Пользовательская программа и данные пользователя также остаются сохраненными. При наличии резервной батареи питания не играет роли тип используемого модуля памяти - RAM или флэш-EPROM.

Если используется модуль памяти Flash EPROM и отсутствует резервная батарея питания, то S7-300 и S7-400 ведут себя по-разному. В контроллерах S7-300 состояния сигналов меркеров, таймеров, счетчиков, объявленных ременентными, сохраняются, тогда как в S7-400 - не сохраняются.

Содержание ременентных блоков данных также остается неизменным в S7-300. Необходимо отметить, что в S7-300 содержание ременентных областей данных сохраняется в CPU, а не в модуле памяти.

Остальные блоки данных в контроллерах S7-300 и все блоки данных в контроллерах S7-400 копируются из модуля памяти в рабочую (work) память как кодовые блоки. Единственные блоки, содержание которых сохраняется, это блоки, находящиеся в модуле памяти. Блоки данных, которые создаются с помощью системной функции SFC 22 CREAT_DB, не являются ременентными. После перезапуска блоки данных имеют содержание, которое хранится в модуле памяти, т.е. то самое содержание, с которым они были запрограммированы.

22.2.4 Определение параметров для перезапуска

При параметризации CPU на вкладке "Startup" ("Запуск") Вы можете определить режим перезапуска, сделав следующие установки:

- "Restart when the set configuration is not the same as the actual configuration" (перезапуск когда набор параметров конфигурации не совпадает с фактической конфигурацией).
  Перезапуск выполняется, если конфигурация параметров оборудования не согласуется с фактической конфигурацией.
- "Hardware test at complete restart (warm restart)" (тестирование оборудования при "полном" перезапуске (при "теплом" перезапуске)).
  S7-300 CPU выполняют тестирование оборудования при включении электропитания.
- "Delete PIQ at warm restart" (очищать область таблицы выходов образа процесса при "теплом" перезапуске).
  S7-400 CPU очищают области всех таблиц выходов образа процесса при "теплом" перезапуске.
- "Disable warm restart at manual restart" (запретить "теплый" перезапуск при ручном режиме перезапуска).
  "Теплый" перезапуск в ручном режиме запрещен.
- "Restart following POWER UP" (перезапуск после включения питания).
  Определение типа перезапуска после включения питания.
- "Monitoring time for ready signal of the modules" (значение времени мониторинга, отведенного модулям на посылку сигнала готовности).
  Если истекло время, отведенное модулям на посылку сигнала готовности, CPU возвращается к режиму STOP; событие записывается в диагностический буфер (особенно важно при включении питания в стойках расширения).
- "Monitoring time for transferring the parameters to the modules" (значение времени мониторинга, отведенного на пересылку параметров модулям).

Если истекает время, отведенное на пересылку параметров модулям, CPU возвращается к режиму STOP; событие записывается в диагностический буфер. (В случае возникновения такой ошибки Вы можете при параметризации CPU задать самое большое значение времени мониторинга для пересылки параметров модулям - без сброса памяти. Если Вы пересылаете системные данные "пустого" проекта, в котором задано новое значение времени для пересылки параметров модулям, то параметризация модулей заканчивается внутри отрезка времени, равного "старому" значению времени мониторинга).

- "Monitoring time for warm restart" (значение времени мониторинга, в течение которого сохраняется разрешение на "теплый" перезапуск). Если время между моментом выключения и моментом включения питания или между моментом перехода к режиму STOP и моментом перехода к режиму RUN больше, чем значение времени мониторинга, CPU возвращается к режиму STOP. Значение времени мониторинга, равное 0, выключает данный режим контроля времени.

22.3 Типы перезапуска

22.3.1 Режим запуска (START-UP)

CPU выполняет перезапуск в следующих случаях:

- когда выключается источник питания;
- когда переключатель режимов переключается с режима STOP (СТОП) в режим RUN (РАБОТА) или RUN-P;
- когда приходит соответствующий запрос от коммуникационной функции (запрос, инициированный программатором или коммуникационным функциональным блоком от другого CPU).

Ручной (Manual) перезапуск инициируется посредством ключа или с помощью коммуникационной функции. *Автоматический (Automatic)* перезапуск инициируется при включении источника питания.

Подпрограмма перезапуска может иметь любую длину, и не существует ограничений на время ее выполнения; при ее обработке функция мониторинга цикла сканирования находится в неактивном состоянии.

Во время выполнения подпрограммы перезапуска никакие прерывания не обслуживаются. Исключения составляют ошибки, которые обрабатываются также как и в режиме RUN (производится вызов соответствующих организационных блоков для обработки ошибок).

При выполнении подпрограммы перезапуска CPU обновляет таймеры, измерители времени наработки и часы реального времени. Во время перезапуска выходные модули находятся в заблокированном состоянии, т.е. выходные сигналы не передаются на внешние выходы. Блокировка выходов отменяется только в конце обработки подпрограммы перезапуска и перед началом цикла сканирования основной программы.

Выполнение подпрограммы перезапуска может быть прервано, например,

при использовании переключателя режимов или при сбое в системе электропитания. После прерывания выполнения подпрограммы перезапуска, ее выполнение возобновляется сначала после включения питания. Если происходит прерывание подпрограммы во время "полного" перезапуска, то "полный" перезапуск необходимо возобновить. Если происходит прерывание подпрограммы во время "теплого" перезапуска, то после этого возможен перезапуск любого типа.

На рисунке 22.2 представлено поведение CPU во время перезапуска.

22.3.2 "Холодный" перезапуск (Cold Restart)

При холодном перезапуске CPU устанавливает свои собственные параметры и параметры модулей в запрограммированное исходное состояние, стирает все данные в системной памяти (включая реманентные), вызывает организационный блок OB 102 и затем выполняет основную программу в OB 1 с самого начала.

Исполняемая в настоящий момент программа и текущие данные в рабочей (work) памяти удаляются и с ними - также блоки данных, созданные SFC; программа из загрузочной (load) памяти перезагружается. (В отличие от процедуры сброса памяти в данном случае загрузочная (load) RAM-память не стирается).

Ручной режим "холодного" перезапуска (Manual cold restart)

"Холодный" перезапуск в ручном режиме выполняется в следующих случаях:

- с помощью переключателя режимов на CPU: если переключатель режимов на CPU удерживался по крайней мере в течение 3 секунд в положении MRES при переходе из режима STOP (СТОП) в режим RUN (РАБОТА) или RUN-P;
- с помощью коммуникационных функций от программатора PG или SFB с другого CPU: при этом переключатель режимов должен быть в положении RUN (РАБОТА) или RUN-P.

Ручной режим "холодного" перезапуска всегда может быть инициирован, пока CPU не выдаст запрос на сброс памяти.

Автоматический режим "холодного" перезапуска (Automatic cold restart)

"Холодный" перезапуск в автоматическом режиме инициируется при включении источника питания. Автоматический режим "холодного" перезапуска выполняется в следующих случаях:

- если CPU не был в режиме STOP (СТОП), когда было выключено питание;
- если переключатель режимов находится в положениях RUN (РАБОТА) или RUN-P.

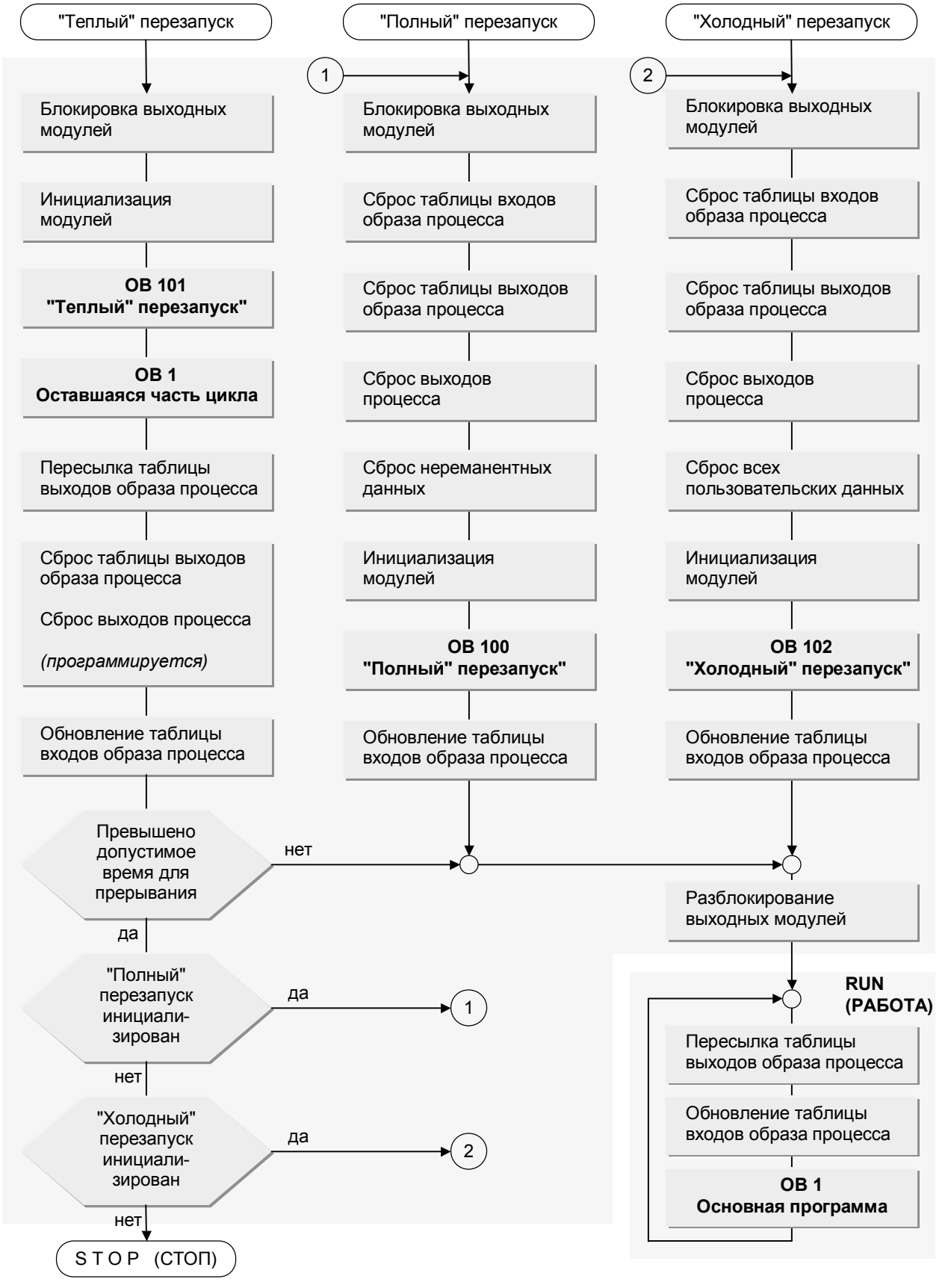


Рис. 22.2 Действия CPU во время перезапуска

- если процесс "холодного" перезапуска был прерван выключением или перебоем питания;
- если задан параметр "Automatic cold restart on power up" (автоматический "холодный" перезапуск при включении питания).

При работе без батареи резервного питания CPU выполняет автоматический "нереманентный" (т.е. с потерей всех данных пользователя) "полный" перезапуск. Сначала CPU автоматически сбрасывает память, затем копирует пользовательскую программу из модуля памяти в рабочую (work) память. В этом случае модуль памяти должен быть типа Flash EPROM.

22.3.3 "Полный" перезапуск (Complete Restart)

При полном перезапуске CPU устанавливает свои собственные параметры и параметры модулей в запрограммированное исходное состояние, стирает нереманентные данные в системной памяти, вызывает организационный блок OB 100 и затем выполняет основную программу в OB 1 с самого начала.

Ручной режим "полного" перезапуска (Manual complete restart)

"Полный" перезапуск в ручном режиме выполняется в следующих случаях:

- с помощью переключателя режимов на CPU: при переходе из режима STOP (СТОП) в режим RUN (РАБОТА) или RUN-P (в случае S7-400 CPU) и с переключателем режима перезапуска, находящемся в положении CRST;
- с помощью коммуникационных функций от программатора PG или SFB с другого CPU: при этом переключатель режимов должен быть в положении RUN (РАБОТА) или RUN-P.

Ручной режим "полного" перезапуска всегда может быть инициирован, пока CPU не выдаст запрос на сброс памяти.

Автоматический режим "полного" перезапуска (Automatic complete restart)

"Полный" перезапуск в автоматическом режиме инициируется при включении источника питания. Автоматический режим "полного" перезапуска выполняется в следующих случаях:

- если CPU не был в режиме STOP (СТОП), когда было выключено питание;
- если переключатель режимов находится в положениях RUN (РАБОТА) или RUN-P.
- если процесс "полного" перезапуска был прерван выключением или перебоем питания;
- если задан параметр "Automatic complete restart on power up" (автоматический "полный" перезапуск при включении питания).

Положение переключателя режимов перезапуска не играет роли, если инициирован автоматический "полный" перезапуск.

При работе без батареи резервного питания CPU выполняет автоматический "нереманентный" (т.е. с потерей всех данных пользователя) "полный" перезапуск. Сначала CPU автоматически сбрасывает память, затем копирует пользовательскую программу из модуля памяти в рабочую (work) память. В этом случае модуль памяти должен быть типа Flash EPROM.

22.3.4 "Теплый" перезапуск (Warm Restart)

"Теплый" перезапуск возможен только в системах S7-400.

В случае перехода к режиму STOP (СТОП) или при сбое в системе питания CPU сохраняет всю информацию о прерываниях, так же как и состояние всех внутренних регистров, которые имеют значение для обработки пользовательской программы. Поэтому при "теплом" перезапуске возможно продолжение выполнения программы, начиная с точки, в которой ее выполнение было прервано. При этом программа, выполнение которой возобновляется, может быть как основной программой, так и подпрограммой для обработки прерывания или ошибки. Кроме того, информация о всех "старых" прерываниях сохраняется, поэтому все они будут обслужены.

Так называемая "оставшаяся незавершенной часть цикла" ("residual cycle"), начиная с точки программы, в которой ее выполнение было прервано, и до конца программы, обрабатывается CPU после "теплого" перезапуска и считается частью собственно процедуры перезапуска. Никакие (новые) прерывания при этом не обслуживаются. Выходные модули заблокированы и находятся в своем исходном состоянии.

"Теплый" перезапуск возможен только тогда, когда не было сделано никаких изменений в пользовательской программе в то время, пока CPU находился в состоянии STOP, таких как изменение блока.

При параметризации CPU Вы можете определить отрезок времени, в течение которого сохраняется разрешение для CPU на выполнение "теплого" перезапуска после прерывания (в диапазоне от 100 мс до 1 часа). Если прерывание длится большее время, то будет разрешен только "полный" перезапуск. Длительность прерывания считается от момента перехода системы из режима RUN (РАБОТА) в режим STOP (СТОП) или от момента сбоя электропитания до момента возвращения в режим RUN (после выполнения организационного блока OB 101 и выполнения "оставшаяся незавершенной части цикла программы").

Ручной режим "теплого" перезапуска (Manual warm restart)

"Теплый" перезапуск в ручном режиме выполняется в следующих случаях:

- если переключатель режимов на CPU был переключен из положения RUN (РАБОТА) или RUN-P в режим STOP (СТОП), при переключателе

режима перезапуска, находящемся в положении WRST (только для CPU, имеющих переключатель режима перезапуска);

- с помощью коммуникационных функций от программатора PG или SFB с другого CPU: при этом переключатель режимов должен быть в положении RUN (РАБОТА) или RUN-P.

Ручной режим "теплого" перезапуска может быть инициирован, если только на вкладке "Restart" ("Перезапуск") окна параметризации CPU отменена блокировка "теплого" перезапуска. Перевод в режим STOP должен быть выполнен вручную или с помощью переключателя режимов, или с помощью коммуникационной функции; CPU может выполнить "теплый" перезапуск только в том случае, если CPU находится в режиме STOP.

Автоматический режим "теплого" перезапуска (Automatic warm restart)

"Теплый" перезапуск в автоматическом режиме инициируется при включении источника питания. Автоматический режим "теплого" перезапуска выполняется только в следующих случаях:

- если CPU не был в режиме STOP (СТОП), когда было выключено питание;
- если переключатель режимов находится в положениях RUN (РАБОТА) или RUN-P, когда CPU был включен;
- если задан параметр "Automatic warm restart on power up" (автоматический "теплый" перезапуск при включении питания);
- если батарея резервного питания вставлена в батарейный отсек и находится в рабочем состоянии.

Положение переключателя режима перезапуска не имеет значения при выполнении "теплого" перезапуска.

22.4 Установление адреса модуля

Вы можете установить адрес модуля с помощью следующих системных функций:

- SFC 5 GADR_LGC
  Функция для установления логического адреса канала модуля;
- SFC 50 RD_LGADR
  Функция для установления всех логических адресов модуля;
- SFC 49 LGC_GADR
  Функция для установления адреса слота модуля.

В таблице 22.2 представлены параметры вышеуказанных системных функций.

Данные функции SFC используют параметры IOID в качестве общих параметров для логических адресов (= адрес в области I/O). Параметр IOID вводит адрес B#16#54, установленный для периферийных входов (PI) или адрес B#16#55, установленный для периферийных выходов (PQ).

Таблица 22.2 Параметры системных функций для установления адресов модуля

SFC	Параметр	Объявление	Тип данных	Описание
5	SUBNETID	INPUT	BYTE	Идентификатор подсети
	RACK	INPUT	WORD	Номер стойки
	SLOT	INPUT	WORD	Номер слота
	SUBSLOT	INPUT	BYTE	Номер субмодуля
	SUBADDR	INPUT	WORD	Смещение адресной области в пользовательских данных модуля
	RET_VAL	OUTPUT	INT	Информация об ошибках
	IOID	OUTPUT	BYTE	Идентификатор области
	LADDR	OUTPUT	WORD	Логический адрес канала
50	IOID	INPUT	BYTE	Идентификатор области
	LADDR	INPUT	WORD	Логический адрес модуля
	RET_VAL	OUTPUT	INT	Информация об ошибках
	PEADDR	OUTPUT	ANY	Слово в памяти для PI-адреса
	PECOUNT	OUTPUT	INT	Число возвращенных PI-адресов
	PAADDR	OUTPUT	ANY	Слово в памяти для PQ-адреса
	PACOUNT	OUTPUT	INT	Число возвращенных PQ-адресов
49	IOID	INPUT	BYTE	Идентификатор области
	LADDR	INPUT	WORD	Логический адрес модуля
	RET_VAL	OUTPUT	INT	Информация об ошибках
	AREA	OUTPUT	BYTE	Идентификатор области
	RACK	OUTPUT	WORD	Номер стойки
	SLOT	OUTPUT	WORD	Номер слота
	SUBADDR	OUTPUT	WORD	Смещение адресной области в пользовательских данных модуля

Параметр LADDR содержит I/O-адрес в PI- или PQ-областях, которые соответствуют определенным каналам. Если канал равен 0, то его адрес совпадает с начальным адресом модуля.

Рассматриваемые системные функции позволяют определить заложенные в данных конфигурации логические адреса (начальные адреса модулей) и адреса слотов (размещение модулей в монтажной стойке или в станции для распределенной периферии).

SFC 5 GADR_LGC
Установление логического адреса канала модуля

Системная функция SFC 5 GADR_LGC возвращает значение логического адреса канала, если Вы зададите адрес слота ("географический" адрес). Введите номер подсети в параметр SUBNETID, если модуль принадлежит к распределенной периферии (I/O), или значение B#16#00, если модуль установлен в монтажную стойку. Параметр RACK определяет номер стойки или, в случае распределенных I/O, номер станции. Если модуль не занимает слота подмодуля, то введите B#16#00 в параметр SUBSLOT. Параметр SUBADDR содержит смещение адреса в пользовательских

данных в модуле (например, W#16#0000 устанавливается для начального адреса модуля).

SFC 49 LGC_GADR
Установление адреса слота модуля

Системная функция SFC 49 LGC_GADR возвращает адрес слота модуля, если Вы зададите произвольный логический адрес слота. Если вычесть смещение адреса (параметр SUBADDR) из определенного адреса в области пользовательских данных, то Вы получите начальный адрес модуля. Значение параметра AREA определяет систему, в которой работает модуль (см. табл. 22.3).

Таблица 22.3 Расшифровка выходных параметров функции SFC 49 LGC_GADR

AREA (Область)	System (Система)	Значения параметров RACK, SLOT, SUBADDR
0	S7-400	RACK = Номер стойки SLOT = Номер слота SUBADDR = Адрес смещения от начального адреса
1	S7-300	
2	Распределенные I/O	RACK, SLOT, SUBADDR
3	S5 область P	RACK = Номер стойки SLOT = Номер слота в корпусе адаптера SUBADDR = Адрес смещения от начального адреса
4	S5 область Q	
5	S5 область IM3	
6	S5 область IM4	

SFC 50 RD_LGADR
Установление логических адресов модуля

Для S7-400 Вы можете назначить адреса байтов пользовательских данных в модуле (для несмежных байтов; адресация по возрастанию).

Системная функция SFC 50 RD_LGADR возвращает все логические адреса для модуля, если Вы зададите произвольный адрес из области пользовательских данных.

Параметры PEADDR и PAADDR определяют область компонентов WORD ("пословный" ANY-указатель; например:

P#DBzDBXy.x WORD *nnn*).

Функция SFC 50 RD_LGADR в параметрах PECOUNT и PACOUNT представляет пользователю число записей, возвращенных в эти области.

22.5 Параметризация модулей

Задание параметров для модулей производится с помощью следующих системных функций:

- SFC 54 RD_DPARM

Функция для считывания предопределенных параметров;

- SFC 55 WR_PARM
  Функция для записи динамических параметров;
- SFC 56 WR_DPARM
  Функция для записи предопределенных параметров;
- SFC 57 PARM_MOD
  Функция для выполнения параметризации модуля;
- SFC 58 WR_REC
  Функция для внесения записи данных;
- SFC 59 RD_REC
  Функция для считывания записи данных.

В таблице 22.4 представлены параметры вышеуказанных (и многих других) системных функций.

Таблица 22.4 Параметры системных функций для передачи данных

Параметр	Параметр представлен в системной функции SFC						Объяв-ление	Тип данных	Значения параметров RACK, SLOT, SUBADDR
REQ	-	55	56	57	58	59	INPUT	BOOL	REQ =1 означает запрос записи
IOID	54	55	56	57	58	59	INPUT	BYTE	B#16#54 = периферийные выходы (PI) B#16#55 = периферийные входы (PQ)
LADDR	54	55	56	57	58	59	INPUT	WORD	Начальный адрес модуля
RECNUM	54	55	56	-	58	59	INPUT	BYTE	Номер записи данных
RECORD	-	55	-	-	58	-	INPUT	ANY	Запись данных
RET_VAL	54	55	56	57	58	59	OUTPUT	INT	Информация об ошибках
BUSY	-	55	56	57	58	59	OUTPUT	BOOL	BUSY =1 означает незавершенность задания
RECORD	54	-	-	-	-	59	OUTPUT	ANY	Запись данных

Следующие записи данных могут быть перенесены с помощью вышеуказанных системных функций:

№ записи	Содержание для считывания	Содержание для записи
0	Диагностические данные	Параметры
1	Диагностические данные	Параметры
2 … 127	Пользовательские данные	Пользовательские данные
128 … 255	Диагностические данные	Параметры

Общие замечания по поводу параметризации модулей

Для некоторых S7-модулей могут быть назначены параметры, что означает, что для параметров модуля могут быть назначены такие значения, которые отличаются от значений, принятых "по умолчанию".

Для определения параметров модуля необходимо открыть в окне утилиты конфигурирования оборудования выбранный модуль и заполнить таблицу в соответствующем диалоговом окне. При передаче системных данных объекта *System Data* в разделе *Blocks (Блоки)* в PLC происходит также пересылка параметров модуля.

CPU осуществляет пересылку параметров в модули автоматически в следующих случаях:

- при перезапуске;
- когда модуль устанавливается в сконфигурированный слот;
- после "возвращения" ("return") монтажной стойки или станции распределенной периферии.

Параметры модулей могут быть разделены на статические параметры и динамические параметры. С помощью утилиты конфигурирования оборудования Hardware Configuration Вы можете задать параметры и того, и другого типа. Вы можете также изменять динамические параметры во время выполнения программы, вызывая системные функции SFC. При выполнении программы перезапуска параметры, установленные в модулях с помощью функций SFC, переписываются (заменяются и сохраняются в CPU) значениями, заданными с помощью утилиты конфигурирования оборудования Hardware Configuration.

Параметры для сигнальных модулей располагаются в двух записях данных: статические параметры - в "записи данных 0" ("data record 0"), динамические параметры - в "записи данных 1" ("data record 1"). Пользователь может использовать для пересылки в модуль сразу обеих записей функцию SFC 57 PARM_MOD, для пересылки в модуль только "записи данных 0" или только "записи данных 1" - функцию SFC 56 WR_DPARM и для пересылки в модуль только "записи данных 1" - функцию SFC 55 WR_PARM. Для пересылки эти записи данных должны быть в системных блоках данных в CPU.

После параметризации модуля для S7-400 заданные значения не должны быть актуализированы, пока бит 2 ("Operating mode" - "Режим работы") в байте 2 записи 0 диагностических данных не получит значение "RUN" ("Выполнить") (Значение может быть считано с помощью системной функции SFC 59 RD_REC).

Что касается адресации при передаче данных: необходимо использовать младший начальный адрес модуля (параметр LADDR) совместно с идентификатором, показывающим будете ли Вы определять этот адрес как адрес входа или адрес выхода (параметр IOID). Если Вы назначили одинаковый начальный адрес и для области входов и для области выходов, то используйте идентификатор для входа. Используйте идентификатор I/O независимо от того, хотите Вы выполнить операцию чтения (Read) или записи (Write).

Для определения областей для компонентов формата BYTE в параметре RECORD используйте данные типа ANY. Это может быть переменная типа массива ARRAY, структуры STRUCT или пользовательского типа UDT, или ANY-указатель типа BYTE (например, P#DBzDBXy.x BYTE *nnn*). Если Вы используете переменную, она должна быть "сложной" переменной - отдельные компоненты массива или структуры в качестве переменной не допускаются.

SFC 54 RD_DPARM
Считывание предопределенных параметров

Функция для считывания предопределенных параметров SFC 54 RD_DPARM переносит запись данных с номером, определенным в параметре RECNUM, из соответствующего системного блока данных SDB в область назначения, определенную в параметре RECORD.

С помощью данной функции Вы можете, например, считать эту запись данных и записать в модуль с помощью системной функции SFC 58 WR_REC.

SFC 55 WR_PARM
Запись динамических параметров

Функция для записи динамических параметров SFC 55 WR_PARM переносит запись данных, адресованную с помощью параметра RECORD в модуль, определенный в параметрах IOID и LADDR. Номер записи данных определяется в параметре RECNUM. Предпосылкой корректности выполнения данной функции является условие, заключающееся в том, что запись данных находится в соответствующем системном блоке данных SDB и что она содержит только динамические параметры.

После того, как задание инициализировано, данная системная функция считывает запись данных целиком; операция пересылки при этом может быть распределена на несколько циклов сканирования программы. Пока операция пересылки не закончена параметр BUSY будет содержать значение "1".

SFC 56 WR_DPARM
Запись предопределенных параметров

Функция для записи предопределенных параметров SFC 56 WR_DPARM переносит запись данных с номером, определенным в параметре RECNUM, из соответствующего системного блока данных SDB в модуль, определенный в параметрах IOID и LADDR.

Операция пересылки при этом может быть распределена на несколько циклов сканирования программы. Пока операция пересылки не закончена параметр BUSY будет содержать значение "1".

SFC 57 PARM_MOD
Параметризация модуля

Функция для выполнения параметризации модуля SFC 57 PARM_MOD переносит все записи данных, запрограммированные при параметризации модуля посредством утилиты конфигурирования оборудования Hardware Configuration.

Операция пересылки при этом может быть распределена на несколько циклов сканирования программы. Пока операция пересылки не закончена параметр BUSY будет содержать значение "1".

SFC 58 WR_REC
Запись записей данных

Функция для записи записей данных SFC 58 WR_REC переносит запись данных, адресованную с помощью параметра RECORD и значения числа записей из параметра RECNUM в модуль, определенный в параметрах IOID и LADDR. Операция пересылки начинается, когда параметр REQ принимает значение "1". После того, как задание инициализировано, данная системная функция считывает запись данных целиком.

Операция пересылки при этом может быть распределена на несколько циклов сканирования программы. Пока операция пересылки не закончена параметр BUSY будет содержать значение "1".

SFC 59 RD_REC
Считывание записей данных

Когда параметр REQ принимает значение "1", функция для считывания записей данных SFC 59 RD_REC считывает запись данных, адресованную с помощью параметра RECNUM из модуля и помещает ее в область назначения RECORD. Область назначения должна быть больше или, по крайней мере, должна быть равной длине записи данных. Если пересылка данных завершается без ошибок, то параметр RET_VAL будет содержать число переданных байтов.

Операция пересылки при этом может быть распределена на несколько циклов сканирования программы. Пока операция пересылки не закончена параметр BUSY будет содержать значение "1".

Необходимо отметить, что в S7-300 выпуска до февраля 1997 года SFC считывает столько данных из определенной записи данных, сколько область назначения может вместить. При этом размер области назначения не может превышать размера записи данных.

23 Обработка ошибок

Об ошибках или сбоях, обнаруженных в модулях или собственно в CPU, CPU сообщает различными способами:

- при ошибках, возникших при выполнении арифметических операций (переполнение, некорректное число формата REAL), - посредством установки битов состояния (бит состояния OV, например, сообщает о переполнении числа);
- при ошибках, обнаруженных при выполнении пользовательской программы ("synchronous errors" - "синхронные ошибки"), - посредством вызова организационных блоков OB 121 и OB 122;
- при обнаруженных в программируемом контроллере ошибках, которые не связаны с выполнением пользовательской программы ("asynchronous errors" - "асинхронные ошибки"), - посредством вызова организационных блоков из ряда OB 80 … OB 87.

CPU сигнализирует о том, что произошла ошибка или отказ и, в некоторых случаях, указывает на причину ошибки (сбоя) посредством светоиндикаторов - светодиодов индикации ошибок, расположенных на передней панели. В случае, если произошла неисправимая ошибка (например, неверный код оператора), CPU сразу переходит в режим STOP.

Если CPU находится в режиме STOP, Вы можете использовать программатор PG и функции для работы с информацией CPU для считывания содержимого стека блоков (B stack), стека прерываний (I stack) и стека локальных данных (L stack), что позволит Вам сделать заключение о причинах возникновения ошибок.

Системные средства диагностики могут обнаруживать ошибки/сбои в модулях и помещать информацию о них в диагностический буфер. Информация о переходах CPU из режима в режим (также как и информация о причинах перехода CPU в режим STOP) помещается в диагностический буфер.

Содержимое диагностического буфера сохраняется при переходе системы в режим STOP, при сбросе памяти, а также при сбое в системе электропитания. Поэтому содержимое диагностического буфера может быть считано после последующего восстановления электропитания и выполнения подпрограммы перезапуска с помощью программатора PG.

При работе с CPU новых выпусков Вы можете при параметризации CPU задавать число вводимых записей в диагностический буфер, которые должны там сохраняться.

23.1 Синхронные ошибки

Операционная система CPU генерирует сигнал о том, что обнаружена синхронная ошибка, если произошла ошибка, имеющая прямое отношение к сканируемой программе. В случае, если организационный блок OB обработки синхронных ошибок не запрограммирован, то CPU сразу переходит в режим STOP. Среди синхронных ошибок различаются:

- **ошибки программирования**, для обработки которых вызывается блок OB 121 и
- **ошибки программирования**, для обработки которых вызывается блок OB 122.

В таблице 23.1 представлена стартовая информация для вышеуказанных организационных блоков

Таблица 23.1 Стартовая информация для OB для обработки синхронных ошибок

Имя переменной	Тип данных	Описание, содержание
OB12x_EV_CLASS	BYTE	OB12x_EV_CLASS = B#16#25: вызов OB 121 обработки ошибок программирования; OB12x_EV_CLASS = B#16#29: вызов OB 122 обработки ошибок доступа.
OB12x_SW_FLT	BYTE	Код ошибки (см. раздел 23.2.1 "Фильтры для ошибок")
OB12x_PRIORITY	BYTE	Приоритетный класс, в котором произошла ошибка
OB12x_OB_NUMBR	BYTE	Номер OB (B#16#79 или B#16#80)
OB12x_BLK_TYPE	BYTE	Тип блока, в котором имело место прерывание (только для S7-400): OB: B#16#88; FB: B#16#8E; FC: B#16#8C.
OB121_RESERVED_1 OB122_MEM_AREA	BYTE	Назначение байтов (B#15#xy): 7… (x) …4: 1 доступ к биту; 2 доступ к байту; 3 доступ к слову; 4 доступ к двойному слову 3… (y) …1: 0 I/O область PI или PQ; 1 табл. вх. I образа процесса; 2 табл. вых. Q образа процесса; 3 меркеры M; 4 блок глобальных данных DB; 5 экземплярный блок данных DI; 6 временные локальные данные L; 7 временные локальные данные блока - предшественника V
OB121_FLT_REG OB122_MEM_ADDR	WORD	OB121: источник ошибки • ошибочный адрес (при доступе чтения/записи) • ошибочная область (при ошибке доступа к области) • ошибочный номер блока, функция таймера/счетчика OB122: адрес, по которому произошла ошибка
OB12x_BLK_NUM	WORD	Номер блока, в котором произошла ошибка (только для S7-400)
OB12x_PRG_ADDR	WORD	Адрес ошибки в блоке, который вызвал ошибку (только для S7-400)
OB12x_DATE_TIME	DT	Дата и время обнаружения ошибки программирования

Блок OB для обработки синхронных ошибок имеет такой же приоритет как и блок, в котором произошла ошибка. Поэтому из OB для обработки синхронных ошибок возможен доступ к регистрам блока, в котором

произошло прерывание, и также поэтому программа в этом OB для обработки синхронных ошибок (при определенных условиях с измененным содержанием) может возвращать значения регистров в блок, в котором произошло прерывание.

Необходимо отметить, что при вызове OB для обработки синхронных ошибок, 20 байтов его стартовой информации также передаются в L-стек приоритетного класса, в котором произошла ошибка, как и другие временные локальные данные блока OB для обработки синхронных ошибок и локальные данные организационных блоков, вызываемых в данном OB.

В случае S7-400 в OB для обработки синхронных ошибок может быть вызван другой OB для обработки синхронных ошибок. Глубина вложения вызовов блоков для OB для обработки синхронных ошибок составляет 3 для S7-400 CPU и 4 для S7-300 CPU.

Пользователь может запрещать (disable) и разрешать (enable) вызовы OB для обработки синхронных ошибок с помощью системных функций SFC 36 MSK_FLT, SFC 37 DMSK_FLT, SFC 38 READ_ERR.

23.2 Обработка синхронных ошибок

Следующие системные функции обеспечивают выполнение обработки синхронных ошибок:

- SFC 36 MSK_FLT
  Системная функция SFC 36 MSK_FLT служит для маскирования синхронных ошибок (обеспечивает блокирование вызовов блоков OB);
- SFC 37 DMSK_FLT
  Системная функция SFC 37 DMSK_FLT служит для демаскирования синхронных ошибок (обеспечивает разрешение вызовов блоков OB);
- SFC 38 READ_ERR
  Системная функция SFC 38 READ_ERR служит для считывания регистра ошибок.

Операционная система вносит информацию о синхронной ошибке в диагностический буфер независимо от использования системных функций SFC 36, SFC 37 и SFC 38.

В таблице 23.2 представлены параметры вышеуказанных системных функций.

23.2.1 Фильтрация ошибок

Для того, чтобы управлять системными функциями, обеспечивающими обработку синхронных ошибок, используются "фильтры для ошибок" ("error filters"). В фильтре для ошибок программирования для каждой ошибки программирования предназначается один бит; в фильтре для ошибок доступа для каждой ошибки доступа предназначается один бит.

Таблица 23.2 Параметры системных функций для обработки синхронных ошибок

SFC	Параметр	Объяв-ление	Тип данных	Описание
36	PRGFLT_SET_MASK	INPUT	DWORD	Новый (дополнительный) фильтр для ошибок программирования
	ACCFLT_SET_MASK	INPUT	DWORD	Новый (дополнительный) фильтр для ошибок доступа
	RET_VAL	OUTPUT	INT	W#16#0001 означает, что прежние установки фильтра заменены новыми
	PRGFLT_MASKED	OUTPUT	DWORD	Фильтр для ошибок программирования после "установки"
	ACCFLT_MASKED	OUTPUT	DWORD	Фильтр для ошибок доступа после "установки" битов
37	PRGFLT_RESET_ MASK	INPUT	DWORD	Новый (дополнительный) фильтр для ошибок программирования
	ACCFLT_RESET_ MASK	INPUT	DWORD	Новый (дополнительный) фильтр для ошибок доступа
	RET_VAL	OUTPUT	INT	W#16#0001 означает, что после изменения фильтр содержит биты, которые не установлены (в данном фильтре)
	PRGFLT_MASKED	OUTPUT	DWORD	Фильтр для ошибок программирования после "сброса"
	ACCFLT_MASKED	OUTPUT	DWORD	Фильтр для ошибок доступа после "сброса" битов
38	PRGFLT_QUERY	INPUT	DWORD	Запрос фильтра для ошибок программирования
	ACCFLT_QUERY	INPUT	DWORD	Запрос фильтра для ошибок доступа
	RET_VAL	OUTPUT	INT	W#16#0001 означает, что после изменения фильтр содержит биты, которые не установлены (в данном фильтре)
	PRGFLT_CLR	OUTPUT	DWORD	Фильтр для ошибок программирования с сообщениями об ошибке
	ACCFLT_CLR	OUTPUT	DWORD	Фильтр для ошибок доступа с сообщениями об ошибке

При определении фильтра ошибок Вы устанавливаете соответствующий бит, который соответствует синхронной ошибке, для которой Вы должны выполнить маскирование, демаскирование или запрос о состоянии регистра ошибок. После запроса системная функция SFC 38 возвращает "1" в битах, соответствующих синхронным ошибкам, которые все еще маскированы, или имели место.

Фильтр ошибок доступа показан в таблице 23.3. Колонка кода ошибки Error Code показывает содержание переменной OB122_SW_FLT в стартовой информации для OB 122.

Фильтр ошибок программирования показан в таблице 23.4. Колонка кода ошибки Error Code показывает содержание переменной OB121_SW_FLT в стартовой информации для OB 121.

S7-400 CPU различают два типа ошибок доступа: обращение к несуществующему модулю и некорректное обращение к существующему модулю. Если в модуле произошел сбой во время работы, то этот модуль помечается как "несуществующий" приблизительно через 150 мс.

после попытки доступа к нему, и фиксируется ошибка доступа к I/O при каждой последующей попытке обращения к данному модулю.

Таблица 23.3 Фильтр ошибок доступа

Бит	Код ошибки (Error Code)	Содержание
2	B#16#42	Ошибка доступа к I/O при считывании S7-300: модуль или не существует или не выдает подтверждения S7-400: существующий модуль не выдает подтверждения после первого обращения к I/O (превышение времени)
3	B#16#43	Ошибка доступа к I/O при записи S7-300: модуль или не существует или не выдает подтверждения S7-400: существующий модуль не выдает подтверждения после первого обращения к I/O (превышение времени)
4	B#16#44	Только для S7-400: Ошибка доступа к I/O при попытке считывания в несуществующий модуль или (в случае более поздних проверок) повторные обращения к модулям, которые не выдают подтверждения
5	B#16#45	Только для S7-400: Ошибка доступа к I/O при попытке записи в несуществующий модуль или (в случае более поздних проверок) повторные обращения к модулям, которые не выдают подтверждения

Таблица 23.4 Фильтр ошибок программирования

Бит	Код ошибки (Error Code)	Содержание
1	B#16#21	Ошибка при преобразования BCD-числа (при преобразовании обнаружена "псевдо-тетрада")
2	B#16#22	Ошибка размера области при чтении (адрес располагается за пределами допустимой области)
3	B#16#23	Ошибка размера области при записи (адрес располагается за пределами допустимой области)
4	B#16#24	Ошибка размера области при чтении (в указателе на область задана недопустимая область)
5	B#16#25	Ошибка размера области при записи (в указателе на область задана недопустимая область)
6	B#16#26	Некорректный номер таймера
7	B#16#27	Некорректный номер счетчика
8	B#16#28	Ошибка адреса при чтении (адрес бита <>0 при обращении к байту, слову, двойному слову и косвенной адресации)
9	B#16#29	Ошибка адреса при записи (адрес бита <>0 при обращении к байту, слову, двойному слову и косвенной адресации)
16	B#16#30	Ошибка записи в глобальный блок данных (для блока установлена защита от записи)
17	B#16#31	Ошибка записи в экземплярный блок данных (для блока установлена защита от записи)
18	B#16#32	Некорректный номер глобального блока данных (DB регистр)
19	B#16#33	Некорректный номер экземплярного блока данных (DI регистр)
20	B#16#34	Некорректный номер функции (FC)
21	B#16#35	Некорректный номер функционального блока (FB)
26	B#16#3A	Вызываемый блок данных (DB) не существует
28	B#16#3C	Вызываемая функция (FC) не существует
30	B#16#3E	Вызываемый функциональный блок (FB) не существует

CPU также выдает сообщение об ошибке доступа к I/O, когда совершается попытка обращения к несуществующему модулю независимо от того, выполняется попытка прямого доступа к модулю (через область I/O) или попытка косвенного доступа к модулю (через область образа процесса).

Биты фильтров ошибок, непоказанные в приведенных таблицах, не имеют отношения к обработке синхронных ошибок.

23.2.2 Маскирование синхронных ошибок

Системная функция SFC 36 MSK_FLT позволяет блокировать вызовы OB обработки синхронных ошибок с помощью фильтров ошибок. "1" в фильтре ошибок индицирует те синхронные ошибки, при появлении которых не будут вызываться организационные блоки для их обработки (в таких случаях говорят, что синхронные ошибки "маскированы" - "masked").

При маскировании синхронных ошибок с помощью фильтров ошибок данные маскирования добавляются к данным маскирования, которые хранятся в памяти операционной системы. Системная функция SFC 36 возвращает значение функции, равное W#16#0001, показывающее, что в сохраненных данных маскирования уже установлен хотя бы один бит из вновь устанавливаемых согласно определению входных параметров функции.

Функция SFC 36 возвращает "1" в выходных параметрах для всех маскированных ошибок на текущий момент.

Если происходит синхронная ошибка, которая ранее была маскирована, то соответствующий ей блок OB не вызывается, а информация об ошибке поступает в диагностический буфер. Блокировка (Disable) вызова OB касается текущего приоритетного класса (приоритетного уровня - "priority level"). Например, если Вы заблокировали вызов OB обработки синхронной ошибки в основной программе, данный блок OB все же будет вызываться в случае возникновения этой ошибки в подпрограмме, обслуживающей прерывание, которое происходит в основной программе.

23.2.3 Демаскирование синхронных ошибок

Системная функция SFC 37 DMSK_FLT позволяет разблокировать (разрешить) вызовы OB обработки синхронных ошибок с помощью фильтров ошибок. Вы должны ввести "1" в фильтр ошибок для обозначения тех синхронных ошибок, для которых вновь должны вызываться организационные блоки для их обработки (в таких случаях говорят, что синхронные ошибки "демаскированы" - "unmasked"). Установленные пользователем биты соответствуют определенным битам в регистре ошибок, которые теперь будут сброшены. Системная функция SFC 37 возвращает значение функции W#16#0001, показывающее, что в уже сохраненных данных маскирования нет ни одного установленного (маскирующего соответствующую ошибку) бита, который должен быть сброшен в соответствии с вновь определенными входными параметрами для системной функции (для демаскирования соответствующей ошибки).

Функция SFC 37 возвращает "1" в выходных параметрах для всех маскированных ошибок на текущий момент.

Если происходит синхронная ошибка, которая была демаскирована, то вызывается соответствующий ей блок OB, а информация об ошибке поступает в диагностический буфер. Блокировка (Disable) вызова OB касается текущего приоритетного класса (приоритетного уровня - "priority level").

23.2.4 Считывание регистра ошибок

Системная функция SFC 38 READ_ERR служит для считывания регистра ошибок с помощью фильтров ошибок. Вы должны ввести "1" в фильтр ошибок для обозначения тех синхронных ошибок, состояние битов которых должно быть считано. Системная функция SFC 38 возвращает значение функции W#16#0001, если в сохраненных данных маскирования не установлен (для маскирования) хоть один бит из тех битов, для которых производится запрос на считывание при определении входных параметров функции.

Функция SFC 36 возвращает "1" в выходных параметрах для выбранных ошибок, если эти ошибки происходили, и сбрасывает в регистре ошибок биты тех ошибок, для которых производится запрос. При этом с помощью данной функции считывается информация о синхронных ошибках, которые относятся к текущему приоритетному классу (приоритетному уровню - "priority level").

23.2.5 Ввод "заменяющего" значения (значения замены) (Substitute Value)

Системная функция SFC 44 REPL_VAL позволяет пользователю ввести "заменяющее" значение или значение замены в аккумулятор 1 (accumulator 1) из организационного блока OB обработки синхронной ошибки. Вы можете использовать системную функцию SFC 44 REPL_VAL, когда нет больше возможности считывать никакие значения из модуля (например, в случае, когда модуль неисправен). Если Вы запрограммировали системную функцию SFC 44, всякий раз при попытке доступа к модулю будет вызываться организационный блок OB 122 (блок обработки ошибки доступа). Когда Вы вызываете функцию SFC 44, Вы можете загрузить значение замены в аккумулятор. При этом сканирование программы продолжится далее с данным значением замены.

Параметры для SFC 44 REPL_VAL представлены в таблице 23.5.

Таблица 23.5 Параметры для системной функции SFC 44 REPL_VAL

SFC	Параметр	Объяв-ление	Тип данных	Содержание, описание
44	VAL	INPUT	DWORD	Значение замены
	RET_VAL	OUTPUT	INT	Информация об ошибках

23.3 Асинхронные ошибки

Асинхронные ошибки - это такие ошибки, которые могут произойти независимо от выполнения (сканирования) программы. Если возникает какая-либо асинхронная ошибка, операционная система вызывает один из нижеуказанных организационных блоков:

OB 80 - блок обработки ошибки времени (timing error);

OB 81 - блок обработки ошибки в блоке питания (power supply error);

OB 82 - блок обработки диагностического прерывания (diagnostic interrupt);

OB 83 - блок обработки прерывания установки/удаления модуля (insert/remove module interrupt);

OB 84 - блок обработки отказа аппаратной части CPU (CPU hardware fault);

OB 85 - блок обработки ошибки выполнения программы (program execution error);

OB 86 - блок обработки отказа стойки (rack failure);

OB 87 - блок обработки коммуникационной ошибки (communication error).

Вызов блока OB 82 (блок обработки диагностического прерывания) рассматривается в разделе 23.4 "Системная диагностика".

Для S7-400H существуют еще три дополнительных организационных блока для обработки асинхронных ошибок:

OB 70 - блок обработки ошибки резервирования периферии (I/O redundancy errors);

OB 72 - блок обработки ошибки резервирования CPU (CPU redundancy errors);

OB 73 - блок обработки ошибки резервирования коммуникаций (communication redundancy errors).

Выполнение данных организационных блоков, запускаемых для обработки асинхронных ошибок, может быть заблокировано (disable) или, наоборот, разрешено (enable) с помощью системных функций SFC 39 DIS_IRT и SFC 40 EN_IRT соответственно, а может также быть заблокировано на время (отложено), а затем вновь разблокировано с помощью системных функций SFC 41 DIS_AIRT и SFC 42 EN_AIRT соответственно.

Ошибки времени (Timing Error)

Операционная система вызывает организационный блок OB 80, если происходит одна из следующих ошибок:

- превышается время мониторинга цикла сканирования программы;
- ошибка при попытке доступа к организационному блоку (OB, к которому делается попытка обращения, еще обрабатывается или

вызывается слишком часто в данном приоритетном классе);

- ошибка прерывания по времени суток (ошибка TOD-прерывания) (время, установленное для TOD-прерывания, прошло из-за перевода часов реального времени вперед или из-за более позднего перехода в режим RUN [РАБОТА]).

Если блок OB 80 недоступен в момент, когда произошла ошибка времени, то CPU переходит в режим STOP (СТОП). CPU также переходит в режим STOP, если организационный блок OB 80 вызывается второй раз в том же самом цикле сканирования программы.

Ошибки в блоке питания
(Power Supply Error)

Операционная система вызывает организационный блок OB 81, если происходит одна из следующих ошибок:

- если разряжена хоть одна резервная батарея в центральном контроллере или в блоке расширения;
- если не подается напряжение от батареи в центральном контроллере или в блоке расширения;
- если отказал источник питания на 24 В в центральном контроллере или в блоке расширения.

Организационный блок OB 81 вызывается для обработки как "приходящих" ("incoming"), так и для "исходящих" ("outgoing") событий. Если блок OB 81 недоступен в момент, когда произошла ошибка в блоке питания, CPU продолжает функционирование.

Прерывание установки/удаления модуля
(Insert/Remove Module Interrupt)

Операционная система выполняет мониторинг модульной конфигурации системы, проверяя ее один раз в секунду. Каждый раз, когда какой-либо модуль устанавливается в стойку или удаляется из стойки в режимах RUN (РАБОТА), STOP (СТОП) или START-UP (ЗАПУСК), соответствующая информация поступает в диагностический буфер и в список состояний системы.

Кроме того, при этом операционная система вызывает организационный блок OB 83, если CPU находится в режиме RUN (РАБОТА). Если блок OB 83 недоступен в момент, когда произошло прерывание установки или удаления модуля, CPU переходит в режим STOP (СТОП).

Может пройти максимум одна секунда, прежде чем будет возбуждено "прерывание установки/удаления модуля" (insert/remove module interrupt) после установки или удаления модуля. Вследствие чего за этот промежуток времени (между удалением модуля и генерацией соответствующего прерывания) может произойти ошибка доступа или ошибка, связанная с обновлением отображения процесса.

Если соответствующий модуль вставляется в сконфигурированный слот, CPU автоматически параметризирует этот модуль, используя записи данных, ранее сохраненные в данном CPU. Организационный блок OB 83

вызывается тогда только для того, чтобы сигнализировать о том, что подключенный модуль готов к работе.

Отказ аппаратной части CPU
(CPU Hardware Fault)

Операционная система вызывает организационный блок OB 84, если происходит или исчезает ошибка интерфейса (подсети MPI или PROFIBUS DP). Если блок OB 84 недоступен в момент, когда произошел отказ аппаратной части CPU, то CPU переходит в режим STOP (СТОП).

Ошибки выполнения программы
(Program Execution Errors)

Операционная система вызывает организационный блок OB 85, если происходит одна из следующих ошибок:

- если поступает запрос на запуск организационного блока, который не был загружен;
- ошибка при попытке доступа операционной системы к блоку (например, отсутствует экземплярный блок данных для вызываемого системного функционального блока SFB);
- ошибка доступа к периферии во время (автоматического) обновления образа процесса (со стороны системы).

В S7-400 CPU и CPU 318 организационный блок OB 85 вызывается при каждой ошибке доступа к периферии (I/O) (со стороны системы), например, во время обновления образа процесса в каждом цикле. При этом при каждом обновлении образа процесса в соответствующие байты в таблице входов образа процесса записываются либо нулевые значения, либо значения замены.

В S7-300 CPU (за исключением CPU 318) организационный блок OB 85 не вызывается в случае ошибки доступа к периферии (I/O) во время автоматического обновления образа процесса. При первой ошибке доступа к периферии в соответствующие байты в таблице входов образа процесса записываются либо нулевые значения, либо значения замены, после чего образ процесса больше не обновляется.

В соответствующим образом оснащенных CPU Вы можете использовать параметризацию CPU для установления режима вызова блока OB 85 для случаев ошибки доступа со стороны системы к периферии (I/O):

- организационный блок OB 85 вызывается при каждой ошибке доступа; в соответствующий входной байт записывается либо нулевое значение, либо значение замены;
- организационный блок OB 85 вызывается при первой ошибке доступа к периферии с атрибутом "incoming" ("приходящее событие"); при этом в соответствующие байты в таблице входов образа процесса записываются либо нулевые значения, либо значения замены, после чего образ процесса больше не обновляется; если ошибка устраняется, блок OB 85 вновь вызывается, но с атрибутом "outgoing" "уходящее событие", после чего образ процесса продолжает обновляться в обычном режиме;

- организационный блок OB 85 не вызывается в случае ошибки доступа; в соответствующий входной байт один раз записывается либо нулевое значение, либо значение замены, после чего образ процесса больше не обновляется.

Если блок OB 85 недоступен в момент, когда произошла ошибка выполнения программы, то CPU переходит в режим STOP (СТОП).

Отказ стойки
(Rack Failure)

Операционная система вызывает организационный блок OB 86, если она обнаружила отказ стойки (авария в системе питания, обрыв провода, отказ интерфейсного модуля IM), отказ в подсети или в станции периферии. Организационный блок OB 86 вызывается и в случае "приходящего события", и в случае "уходящего события".

В мультипроцессорном режиме организационный блок OB 86 вызывается во всех CPU, если произошел отказ стойки.

Если блок OB 86 недоступен в момент, когда произошел отказ стойки, то CPU переходит в режим STOP (СТОП).

Коммуникационные ошибки
(Communication Error)

Операционная система вызывает организационный блок OB 87, если происходит коммуникационная ошибка. Ниже представлены примеры некоторых коммуникационных ошибок:

- при использовании обмена посредством глобальных данных обнаружена неправильная идентификация фрейма или длина фрейма;
- невозможна передача диагностического сообщения;
- ошибка синхронизации часов;
- информация о состоянии (статусе) GD не может быть внесена в блок данных.

Если блок OB 87 недоступен в момент, когда произошла коммуникационная ошибка, то CPU переходит в режим STOP (СТОП).

Ошибки резервирования периферии
(I/O Redundancy Errors)

Операционная система H CPU вызывает организационный блок OB 70, если происходят ошибки резервирования периферии - потеря резервирования в подсети PROFIBUS-DP, например, в случае отказа шины в активном ведущем (master) DP-устройстве или в случае отказа интерфейса ведомого (slave) DP-устройства.

Если блок OB 70 недоступен в момент, когда произошла ошибка резервирования периферии, CPU продолжает функционирование.

Ошибки резервирования CPU
(CPU Redundancy Errors)

Операционная система H CPU вызывает организационный блок OB 72,

если происходит одна из следующих ошибок:

- потеря резервирования CPU;
- ошибка сравнения (например, в RAM-памяти, в PIQ-области);
- переключение на резервное ведущее устройство;
- ошибка синхронизации;
- ошибка в подмодуле SYNC;
- прерывание обновления.

Если блок OB 72 недоступен в момент, когда произошла ошибка резервирования CPU, CPU продолжает функционирование.

23.4 Системная диагностика

23.4.1 Диагностические события и диагностический буфер

Системная диагностика представляет из себя систему обнаружения, оценки и сообщения об ошибках, происходящих в программируемых контроллерах. Примеры ошибок: ошибки в пользовательской программе, отказы модулей, обрывы проводов в сигнальных модулях. Примеры *диагностических событий*:

- диагностические прерывания, поступающие от модулей, которые оснащены системой диагностики;
- сообщения о системных ошибках и переходах CPU из одного рабочего режима в другой;
- пользовательские сообщения, создаваемые посредством системных функций.

Модули со встроенной системой диагностики могут генерировать диагностические события двух видов: программируемые диагностические события и непрограммируемые диагностические события.

Программируемые диагностические события создаются только при условии, что пользователь устанавливает параметры, разрешающие работу системы диагностики, тогда как непрограммируемые диагностические события генерируются всегда и не зависят от параметров, разрешающих или запрещающих работу системы диагностики.

В случае поступления диагностического события:

- включаются светоиндикаторы (светодиоды), расположенные на передней панели CPU;
- сигнал о диагностическом событии передается в операционную систему CPU;
- генерируется диагностическое прерывание, если Вы установили параметры, разрешающие генерацию этого прерывания (по умолчанию диагностические прерывания блокируются).

Сигналы о всех диагностических событиях, передаваемые в операционную систему CPU, заносятся в *диагностический буфер (diagnostic buffer)* в том порядке, в котором они поступают, с фиксацией даты и времени поступления. Диагностический буфер - это область памяти в CPU с резервным питанием от батареи, которая сохраняет свое содержимое, даже когда выполняется сброс памяти. Диагностический буфер представляет из себя кольцевой буфер, размер которого определяется типом CPU. Когда диагностический буфер заполняется, то каждое вновь поступающее сообщение записывается в буфер поверх самого старого сообщения.

Пользователь может в любой момент считать содержимое диагностического буфера с помощью программирующего устройства PG. В блоке параметров *System Diagnostics* системной диагностики CPU Вы можете определить, желаете ли Вы расширить диапазон сообщений системной диагностики (включить сообщения о всех вызовах организационных блоков). Вы можете также активизировать опцию пересылки последнего диагностического сообщения в адрес определенного узла в MPI-подсети перед тем, как CPU перейдет в режим STOP (СТОП).

23.4.2 Запись пользовательских сообщений в диагностический буфер

Системная функция SFC 52 WR_USMSG позволяет записать сообщение в диагностический буфер, которое может быть послано в адрес всех узлов MPI-шины.

Параметры для системной функции SFC 52 WR_USMSG представлены в таблице 23.6.

Таблица 23.6 Параметры для функции SFC 52 WR_USMSG

SFC	Параметр	Объяв-ление	Тип данных	Содержание, описание
52	SEND	INPUT	BOOL	Если SEND = "1", то пересылка разрешена
	EVENTN	INPUT	WORD	Идентификатор события
	INFO1	INPUT	ANY	Дополнительная информация 1 (одно машинное слово)
	INFO2	INPUT	ANY	Дополнительная информация 1 (одно двойное слово)
	RET_VAL	OUTPUT	INT	Информация об ошибках

Запись в диагностическом буфере по формату соответствует записи системного события, например, стартовой информации организационного блока. В допустимых пределах Вы сами можете выбрать собственный идентификатор сообщения (записи) (параметр EVENTN) и дополнительную информацию (в параметрах INFO1 и INFO2).

Идентификатор ID соответствует первым двум байтам записи в буфере (см. рис. 23.1).

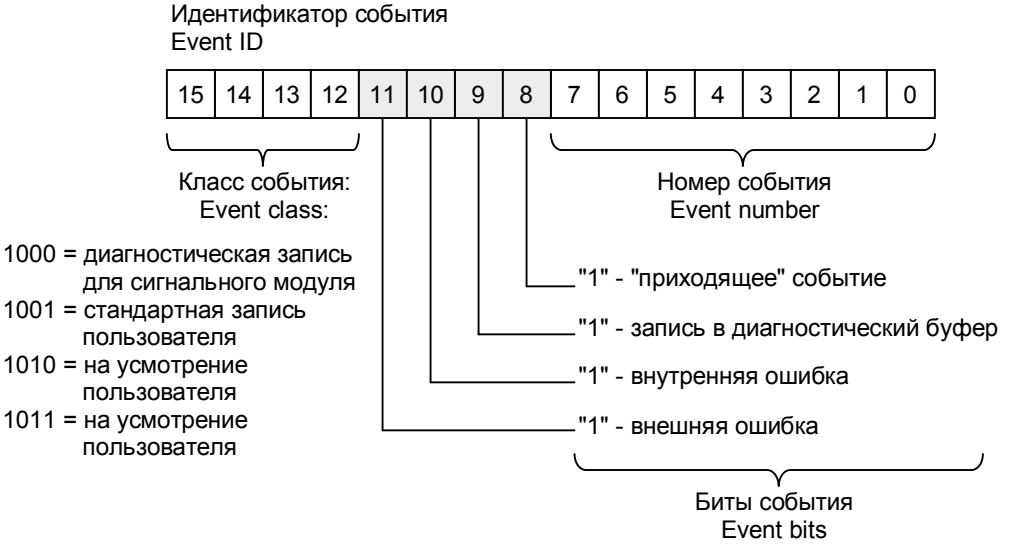


Рис. 23.1 Идентификатор ID события, записываемого в диагностический буфер

Разрешенные для пользователя сообщения относятся к классам 8 (диагностическая запись для сигнального модуля), 9 (стандартная запись пользователя), A и B (произвольные сообщения пользователя).

Для дополнительной информации (INFO1) предназначены байты 7 и 8 (одно слово) и для дополнительной информации (INFO2) предназначены байты с 9 по 12 (двойное слово). Содержимое обеих переменных отдано на усмотрение пользователя.

Для того, чтобы переслать диагностическое сообщение в адрес соответствующего узла установите параметр SEND в состояние "1".

Даже если пересылка сообщения невозможна (например, из-за того, что узел не указан, или из-за того, что буфер для пересылки (Send) переполнен) ввод данной записи будет сделан в диагностический буфер (если установлен бит 9 в идентификаторе события).

23.4.3 Проверка диагностического прерывания

Если диагностическим событием является "приходящее" или "уходящее" диагностическое прерывание, то операционная система прерывает сканирование пользовательской программы и вызывает организационный блок OB 82. Если блок OB 82 незапрограммирован, то CPU переходит в режим STOP (СТОП) при возникновении диагностического прерывания. Выполнение организационного блока OB 82 может быть заблокировано (disable) или, наоборот, разрешено (enable) с помощью системных функций SFC 39 DIS_IRT и SFC 40 EN_IRT соответственно, а может также быть заблокировано на время (отложено), а затем вновь разблокировано с помощью системных функций SFC 41 DIS_AIRT и SFC 42 EN_AIRT соответственно.

В первом байте стартовой информации B#16#39 устанавливается для "приходящего" диагностического прерывания, а B#16#38 устанавливается для "уходящего" диагностического прерывания. В шестом байте представлен идентификатор адреса (B#16#54 устанавливается для входов и B#16#55 - для выходов). Следующая переменная INT содержит адрес модуля, который генерирует диагностическое прерывание. В следующих четырех байтах содержится диагностическая информация, выдаваемая данным модулем.

В блоке OB 82 Вы можете использовать системную функцию SFC 59 RD_REC ("read data record" - "считать запись данных"), которая позволяет получить детальную информацию об ошибке. Диагностическая информация является консистентной до момента выхода из блока OB 82, то есть данные остаются "замороженными" ("frozen"). Выход из OB 82 подтверждает диагностическое прерывание в модуле.

Диагностические данные модуля содержатся в записях данных DS0 и DS1. Запись данных DS0 содержит четыре байта диагностических данных, описывающих текущее состояние модуля. Содержимое этих четырех байтов идентично содержимому байтов с 8 по 11 стартовой информации блока OB 82.

Запись данных DS1 содержит четыре байта из записи данных DS0 и, кроме того, диагностические данные, которые определяются типом модуля.

23.4.4 Считывание списка состояний системы

Список состояний системы (SSL - "system status list") описывает текущее состояние программируемого контроллера. С помощью "информационных" функций этот список может быть считан (но не может быть изменен). Вы можете считать часть списка (то есть "подсписок") с помощью системной функции **SFC 51 RDSYSST**. Подсписок - это "виртуальный" список, что означает, что такие списки могут быть представлены операционной системой CPU только по запросу.

В таблице 23.7 представлены параметры для системной функции SFC 51 RDSYSST.

Таблица 23.7 Параметры для функции SFC 51 RDSYSST

SFC	Параметр	Объяв-ление	Тип данных	Содержание, описание
51	REQ	INPUT	BOOL	Если REQ = "1", то инициируется операция считывания
	SZL_ID	INPUT	WORD	Идентификатор подсписка
	INDEX	INPUT	WORD	Тип или номер объекта подсписка
	RET_VAL	OUTPUT	INT	Информация об ошибках
	BUSY	OUTPUT	BOOL	Если BUSY = "1", то операция считывания еще не завершена
	SZL_HEADER	OUTPUT	STRUCT	Длина и число записей данных
	DR	OUTPUT	ANY	Поле для считанных записей данных

Если параметр REQ = "1", то инициируется операция считывания. Параметр BUSY = "0" сообщает пользователю о том, что операция считывания уже завершилась. Операционная система может выполнить несколько асинхронных операций считывания "как бы одновременно". Ресурсы CPU могут использоваться многими "потребителями". Если функция SFC 51 сообщает пользователю о том, что недостаточно ресурсов (посредством значения функции W#16#8085), то Вы должны будете повторить запрос на считывание.

Содержимое параметров SZL_ID и INDEX зависит от CPU. Параметр SZL_HEADER имеет тип данных STRUCT, с переменными LENGTHDR (тип данных WORD) и N_DR (тип WORD) в качестве составляющих структуры. В параметре LENGTHDR содержится длина записи данных, а в параметре N_DR содержится число записей данных, которые должны быть считаны.

Параметр DR используется для определения переменной или области данных, в которые функция SFC 51 должна переслать считанные записи данных. Например, P#DB200.DBX0.0 WORD 256 описывает область размером в 256 слов данных в блоке данных DB 200, начиная с DBB 0. Если предоставляемая область назначения имеет недостаточный размер, то в нее будет внесено максимально возможное количество записей данных. При этом данные переносятся только полными записями данных. Область назначения должна иметь размер, позволяющий вместить по крайней мере одну запись данных.

Обработка переменных

Данный раздел книги рассматривает вопросы обработки сложных переменных. Для обработки сложных переменных требуются в первую очередь знание структуры данных различных типов, умение использовать косвенную адресацию и способность определять адреса переменных во время выполнения программы.

Доступ к переменным простых **типов данных (data types)** возможен непосредственно с помощью STL-операторов, если Вы имеете дело с двоичными логическими операциями, с операциями с памятью (memory functions) или с операциями загрузки (load) или пересылки (transfer). Для сложных типов данных и пользовательских типов данных прямой доступ в настоящее время возможен только к их отдельным компонентам. И если Вам все-таки необходимо организовать доступ к этим типам данных, то Вы должны знать внутреннюю структуру таких переменных.

Косвенная адресация (indirect addressing) предоставляет доступ к операндам, адреса которых заранее неизвестны (до момента выполнения программы). Пользователь может выбирать между косвенной адресацией посредством памяти (memory-indirect addressing) и косвенной адресацией посредством регистра (register-indirect addressing). Вы можете дождаться выполнения программы, для использования области операнда. Косвенная адресация позволяет Вам получать доступ к переменным сложных типов и пользовательских типов, используя абсолютную адресацию.

При использовании **прямого доступа к переменным (direct variable access)** используется текущий адрес локальной переменной. Если Вы определили адреса, Вы можете обработать соответствующие локальные переменные (в том числе и параметры блока), относящиеся к любому типу данных. В двух предыдущих главах содержится информация, требуемая для этих целей.

Несколько развернутых примеров собраны в разделе 26.4 "Краткое описание примера фрейма сообщения". В данном разделе объясняется, как обрабатываются сложные переменные. Примеры "Данные фрейма сообщения" ("Message Frame Data"), "Подготовка фрейма сообщения" ("Preparing a Message Frame"), и "Контроль времени" ("Clock Check") имеют дело с данными пользовательского типа и применяют переменные сложного типа с использованием системных и стандартных функций. В примерах "Контрольная сумма" ("Checksum") и "Преобразование данных" ("Data Item Conversion") описывается, как обращаться к параметрам, относящимся к сложным типам данных, с помощью косвенной адресации.

В примере "Сохранение фрейма сообщения" ("Save Message Frame") показано, как использовать системную функцию SFC 20 BLKMOV для пересылки данных из области, адрес которой неизвестен до начала выполнения программы.

24 Типы данных

Простые, сложные и пользовательские типы данных; объявление и структура разных типов данных.

25 Косвенная адресация

Указатели на область, указатели на DB, ANY-указатели; косвенная адресация посредством памяти (memory-indirect addressing) и косвенная адресация посредством регистра (register-indirect addressing); внутризонная (area-internal) адресация и межзонная (area-crossing) адресация; использование адресных регистров.

26 Прямой доступ к переменным

Адресация локальных переменных; сохранение переменных;. сохранение данных при передаче параметров; "переменная" ANY-указатель; пример фрейма сообщения.

24 Типы данных

Тип данных определяет их свойства и характеристики, особое представление содержимого одного или больше, чем одного связанных адресов и допустимых областей. STEP 7 предоставляет пользователю возможность предопределения данных, которые он может компилировать, как и определенные им пользовательские типы данных. Типы данных доступны везде. Они могут использоваться в любом блоке.

В разделе 3.7 "Переменные и константы" представлен обзор всех типов данных и соответствующее представление констант.

Данная глава содержит детальную информацию о простых и сложных типах данных; в главе показана структура соответствующих переменных. Вы узнаете, как создаются и используются пользовательские типы данных.

Вы можете найти примеры для типов данных на дискете, прилагаемой к данной книге, в разделе "Variable Handling" ("Обработка переменных") в функциональных блоках FB 101, FB 102 и FB 103 или в исходном файле Chap_24.

24.1 Простые типы данных

Переменные простых типов имеют максимальную длину, равную одному двойному слову; поэтому они могут быть обработаны с помощью функций загрузки (load) и пересылки (transfer) или с помощью двоичных логических операций.

24.1.1 Объявление простых типов данных

Данные простого типа могут занимать один бит, один байт, одно слово и одно двойное слово.

Объявление (Declaration)

```
varname : datatype := pre-assignment;
```

varname - имя переменной

datatype - простой тип данных

pre-assignment - (предопределенное) фиксированное значение переменной

Идентификаторы типов данных (например, BOOL, REAL) являются ключевыми словами; они могут быть записаны также и в нижнем регистре. Переменная простого типа данных может быть объявлена в таблице символов глобальной или локальной переменной в разделе объявлений.

Предопределение (Pre-assignment)

Переменная может быть предопределена во время объявления (в отличие от параметров функции или временных переменных). Значение предопределения должно относиться к тому же типу данных, что и сама переменная.

Применение

Вы можете применять переменные простых типов в качестве фактических параметров для соответствующим образом объявленных параметров блока (того же типа данных POINTER или ANY) или Вы можете организовать к ним доступ посредством обычных STL-операторов (например, двоичная проверка, функции загрузки [load]).

Сохранение переменных

Переменные простых типов сохраняются таким же образом, как соответствующие адреса. Для этого допустимо использовать все адресные области, включая параметры блоков.

24.1.2 Типы данных BOOL, BYTE, WORD, DWORD, CHAR

Переменная типа BOOL представляет собой значение бита (например, входа I 1.0). Переменные типов данных BYTE, WORD и DWORD представляют собой последовательности битов, состоящие соответственно из 8, 16 и 32 битов; при этом отдельные биты не проверяются. В главе 3 "SIMATIC S7" рассматриваются возможные представления как констант.

Специальной формой этих типов данных являются BCD-числа (числа в двоично-десятичном коде) и значения счетчиков, которые используются в сочетании с собственно функцией счетчика, а также тип данных CHAR, представляющий символ в коде ASCII (см. рис. 24.1).

BCD-числа (числа в двоично-десятичном коде)

BCD-числа не имеют специального идентификатора в STL. Вы можете вводить BCD-число с типом данных 16# (шестнадцатеричное), используя только цифры из ряда 0 … 9.

BCD-числа встречаются при загрузке значений таймеров и счетчиков и при работе с функциями преобразования. Тип данных S5TIME# может использоваться для задания значения таймера при запуске функции таймера (см. ниже), а для определения значения счетчика используется тип данных: 16# или C#. Значение счетчика с типом данных C# - это BCD-число из диапазона 000 … 999, в котором знак всегда равен 0.

В общем случае BCD-числа - это беззнаковые числа. При использовании в функции преобразования знак BCD-числа хранится в крайней левой (старшей) тетраде. Это ведет к потере одной тетрады BCD-числа для его диапазона.

В случае, если BCD-число хранится в шестнадцатиразрядном слове, знак хранится в старшей тетраде в бите 15. При этом, если состояние сигнала данного бита равно "0", то знак числа положительный, если состояние сигнала равно "1", то знак - отрицательный. Знак не влияет на значения отдельных тетрад. Аналогичное назначение применяется для 32-разрядного слова.

Диапазон числа составляет от 0 до 999 для 32-разрядных BCD-чисел.

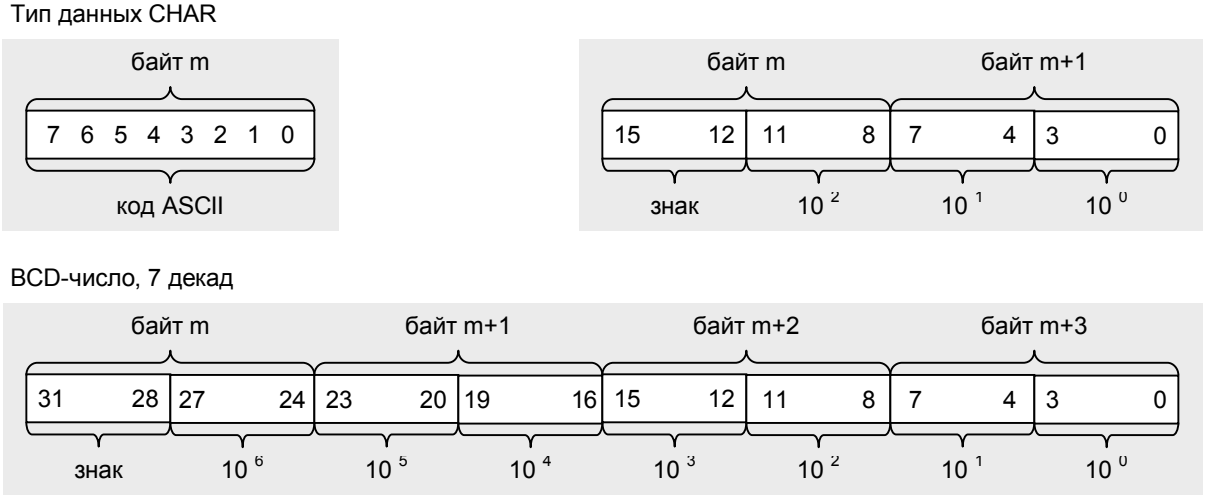


Рис. 24.1 Представление BCD-чисел (чисел в двоично-десятичном коде) и данных типа CHAR

CHAR

Переменная типа CHAR (character - символ) занимает один байт. Тип данных CHAR представляет отдельный символ в формате ASCII. Например, символ 'A'. Вы можете также записать любой печатаемый символ в одинарных кавычках.

В операторах загрузки (load) в языке программирования STL используются также некоторые специальные символы, показанные в таблице 24.1. Например, оператор L '$$' загружает знак доллара в коде ASCII.

Кроме того, Вы можете использовать другие специальные формы типа данных CHAR, когда загружаете в аккумулятор символы в коде ASCII. Например, оператор L 'a' загружает один символ (в данном случае символ a) с выравниванием вправо в аккумуляторе; оператор L 'aa' загружает два символа, а оператор L 'aaaa' соответственно загружает четыре символа.

Таблица 24.1 Специальные символы для переменной типа CHAR

Символ CHAR	Hex	Значение
$$	24_{hex}	Символ доллара
$'	27_{hex}	Одинарная кавычка
$L или $l	$0A_{hex}$	Новая строка
$P или $p	$0C_{hex}$	Новая страница
$R или $r	$0D_{hex}$	Возврат каретки
$T или $t	09_{hex}	Табулятор

24.1.3 Представление чисел

В данном разделе совместно рассматриваются типы данных INT, DINT и REAL. На рис. 24.2 представлено назначение битов для этих типов данных.

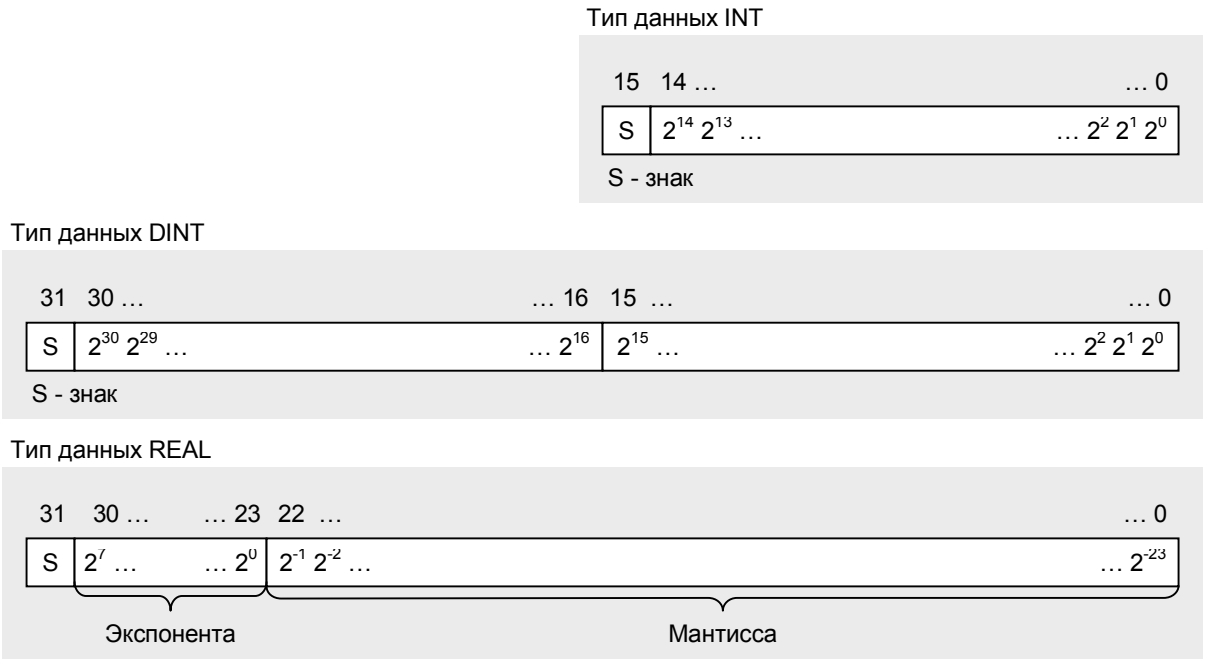


Рис. 24.2 Назначение битов для данных типов INT, DINT и REAL

INT

Переменная типа INT представляет собой целое (целое число), которое сохраняется как 16-разрядное число с фиксированной запятой. Тип данных INT не имеет специального идентификатора.

Переменная типа данных INT занимает одно слово (word).

Состояния сигналов от бита 0 до бита 14 определяют модуль числа; состояние бита 15 определяет знак числа (S). Если состояние сигнала данного бита равно "0", то число положительное, если состояние сигнала равно "1", то число отрицательное. Отрицательные числа представляются как инвертированные числа (two's complement).

Диапазон INT-числа составляет:

от + 32 767 ($7FFF_{hex}$)

до - 32 768 (8000_{hex}).

DINT

Переменная типа DINT представляет собой целое число, которое сохраняется как 32-разрядное число с фиксированной запятой. Целое число хранится в формате DINT, если оно больше +32767 или меньше, чем -32768 или, если перед этим числом стоит специальный идентификатор L#.

Переменная типа данных DINT занимает одно двойное слово (duobleword). Состояния сигналов от бита 0 до бита 30 определяют модуль числа; состояние бита 31 определяет знак числа. Если состояние сигнала данного бита равно "0", то число положительное, если состояние сигнала равно "1", то число отрицательное. Отрицательные числа представляются как инвертированные числа (two's complement).

Диапазон DINT-числа составляет:

от + 2 147 483 647 ($7FFF\ FFFF_{hex}$)

до - 2 147 483 648 ($8000\ 0000_{hex}$).

Пример для STL: с помощью оператора L -100 Вы загружаете в аккумулятор целое INT-число, а с помощью оператора L# -100 Вы загружаете в аккумулятор целое DINT-число. Для данных двух случаев различие заключается в назначении битов левого слова аккумулятора: в случае с INT-числом левое слово аккумулятора содержит значение 0000_{hex}, а в примере с DINT-числом левое слово аккумулятора содержит значение $FFFF_{hex}$.

Пример для SCL: если Вы определили значение константы как -100, то в случаях, когда предстоит выполнить операцию с данным числом и переменной DINT-формата, редактор автоматически преобразует данное INT-число в DINT-формат ("неявное" преобразование типа).

REAL

Переменная типа REAL представляет собой дробное число, которое сохраняется как 32-разрядное число с плавающей запятой. Целое число сохраняется в формате REAL, если в нем за десятичной точкой записан ноль.

Пример для STL: в то время как 100 и L#100 указывает на целое число соответственно в INT- и в DINT-формате, в REAL формате Вы должны записать число 100 либо как 100.0, либо как 1.0e+2 (спецификация с десятичной точкой, в первом случае без экспоненты, во втором случае с экспонентой).

Пример для SCL: в формате REAL Вы можете определить значение константы в любом численном представлении. Значение 100, например, в случаях, когда предстоит выполнить операцию с данным числом и переменной REAL-формата, редактор автоматически преобразует данное число в REAL-формат ("неявное" преобразование типа).

В экспоненциальном представлении Вы можете определить целое или дробное число с помощью 7 значащих цифр со знаком с последующим символом "e" или "E". Вслед за символом "e" или "E" пишется значение показателя степени по основанию 10 ("экспонента"). Преобразование числа REAL-формата во внутреннее представление числа с плавающей запятой выполняется системой STEP 7.

Для REAL-чисел существует различие между числами, которые могут быть представлены с общей точностью ("нормированные" или "нормализованные" ("normalized") числа с плавающей запятой) и числами, которые представляются как числа с ограниченной точностью ("ненормированные" или "денормализованные" ("denormalized") числа с плавающей запятой). Диапазон значений нормированных чисел с плавающей запятой составляет:

от $-3.402\,823 \cdot 10^{+38}$ до $-1.175\,494 \cdot 10^{-38}$

и

от $+1.175\,494 \cdot 10^{-38}$ до $+3.402\,823 \cdot 10^{+38}$.

Диапазон значений ненормированных чисел с плавающей запятой составляет:

от $-1.175\,494 \cdot 10^{-38}$ до $-1.401\,298 \cdot 10^{-45}$

и

от $+1.401\,298 \cdot 10^{-45}$ до $+1.175\,494 \cdot 10^{-38}$.

S7-300 CPU не могут выполнять вычисления с ненормированными числами с плавающей запятой. Битовая структура в этом случае такова, что ненормированное число с плавающей запятой представляется как ноль. Если результат вычисления попадает в диапазон ненормированных чисел с плавающей запятой, то значение этого результата представляется нулевым с установкой битов состояния OV и OS ("нарушение численного диапазона").

CPU выполняет вычисления с полной точностью представления чисел с плавающей запятой. Из-за ошибок округления при преобразовании результаты, отображаемые программатором, могут отличаться от теоретически точного представления результата.

Переменная типа REAL внутренне состоит из трех компонентов:
- знак,
- 8-разрядная экспонента по основанию 2,
- 32-разрядная мантисса.

Знак может принимать значения "0" (положительное число) и "1" (отрицательное число). Экспонента сохраняется как константа, увеличенная на 1 (смещена на +127), так что она имеет диапазон значений от 0 до 255. Мантисса представляет собой дробную часть. Целая часть мантиссы (integer component) не хранится, так как она или всегда равна 1 (в случае нормированных чисел с плавающей запятой), или всегда равна 0 (в случае ненормированных чисел с плавающей

запятой). В таблице 24.2 показаны внутренние границы диапазона числа с плавающей запятой.

Таблица 24.2 Внутренние границы диапазона числа с плавающей запятой

Знак	Экспонента	Мантисса	Значение
0	255	не равно 0	Некорректное число с плавающей запятой (не число)
0	255	0	+ бесконечность
0	1…254	любое значение	Положительное нормированное число с плавающей запятой
0	0	не равно 0	Положительное ненормированное число с плавающей запятой
0	0	0	+ ноль
1	0	0	- ноль
1	0	не равно 0	Положительное ненормированное число с плавающей запятой
1	1…254	любое значение	Положительное нормированное число с плавающей запятой
1	255	0	- бесконечность
1	255	не равно 0	Некорректное число с плавающей запятой (не число)

24.1.4 Представление времени

В данном разделе совместно рассматриваются типы данных S5TIME, DATE, TIME и TIME_OF_DAY. На рис. 24.3 представлено назначение битов для этих типов данных.

Один тип данных TIME_OF_DAY в рассматриваемой категории типов принадлежит также к сложным типам данных, так как для его представления требуется 8 байтов.

S5TIME

Переменные типа S5TIME используются для инициализации таймера из SIMATIC-функций для базовых языков программирования STL, LAD и FBD (язык SCL использует для этих целей представление данных типа TIME). Тип данных S5TIME занимает 16-разрядное слово со структурой "1+3 тетрады" (см. рис. 24.3).

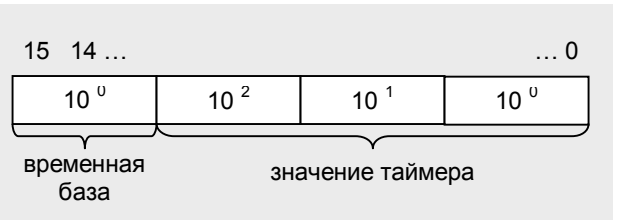
Время задается в часах, минутах, секундах и миллисекундах. Преобразование числа S5TIME-формата во внутреннее представление выполняется системой STEP 7. Внутренне число представляется как BCD-число (число в двоично-десятичном коде) из диапазона от 000 до 999. Временная база (time base) может принимать следующие значения: 10 мс (0000), 100 мс (0001), 1 с (0010), 10 с (0011). Время определяется как результат произведения временной базы на значение таймера.

Примеры:

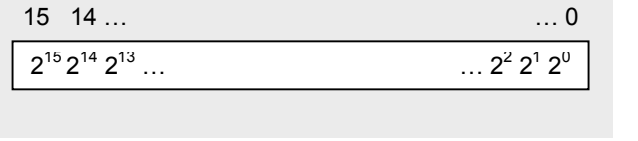
S5TIME#500ms (= 0050_{hex})

S5T#2h46m30s (= 3999_{hex})

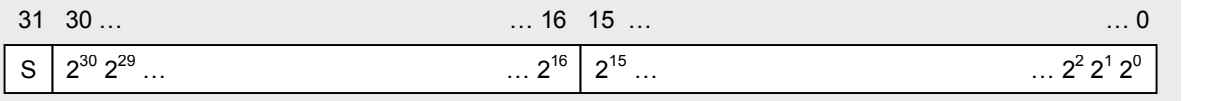
Тип данных S5TIME



Тип данных DATE



Тип данных TIME



S - знак

Тип данных TIME_OF_DAY

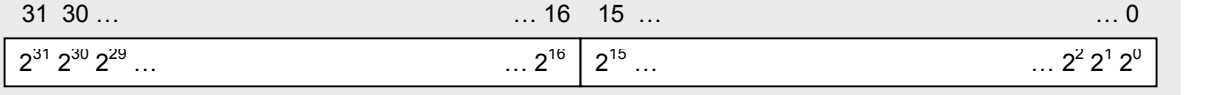


Рис. 24.3 Назначение битов для типов S5TIME, DATE, TIME и TIME_OF_DAY

DATE

Переменная типа DATE хранится в машинном слове как беззнаковое число с фиксированной запятой. Содержание переменной соответствует числу дней, начиная с 01.01.1990 г. Представление числа содержит год, месяц и день, разделенные тире.

Примеры:

DATE#1990-01-01 (= 0000_{hex})

D#2168-12-31 (= $FF62_{hex}$)

TIME

Переменная типа TIME занимает одно двойное слово. Представление числа содержит спецификации дня (d), часов (h), минут (m), секунд (s) и

миллисекунд (ms); при этом отдельные спецификации могут быть пропущены. Содержание переменной интерпретируется как число миллисекунд и сохраняется в 32-разрядном числе со знаком с фиксированной запятой.

Примеры:

TIME#24d20h31m23s647ms (= 7FFF FFFF$_{hex}$)

TIME#0ms (= 0000 0000$_{hex}$)

T#-24d20h31m23s647ms (= 8000 0000$_{hex}$)

В SCL данное представление используется для выражения длительности в функциях SIMATIC-таймера. Редактор автоматически преобразует число, заданное в формате TIME, в число формата S5TIME (структура 1+3 тетрады) с округлением вниз, если это необходимо.

"Десятичное" представление для данных типа TIME также возможно, например, TIME#2.25h или T#2.25h. Такое представление в SCL допустимо только для положительных значений.

Примеры:

TIME#0.0h (= 0000 0000$_{hex}$)

TIME#24.855134d (= 7FFF FFFF$_{hex}$)

TIME_OF_DAY

Переменная типа TIME_OF_DAY занимает одно двойное слово. Содержание переменной интерпретируется как число миллисекунд (ms), начиная с начала дня (0.00 часов) и сохраняется в 32-разрядном беззнаковом числе с фиксированной запятой. Спецификация числа состоит из часов, минут и секунд, разделенных двоеточием. В спецификации возможно указывать число миллисекунд с записью после числа секунд и отделением от этого числа точкой. Значение для миллисекунд может быть опущено.

Примеры:

TIME_OF_DAY#00:00:00 (= 0000 0000$_{hex}$)

TOD#23:59:59.999 (= 0526 5BFF$_{hex}$)

24.2 Сложные типы данных

К сложным типам данных относятся такие данные, которые целиком не могут быть непосредственно обработаны с помощью STL-операторов, но допускаются в SCL-выражениях. Система STEP 7 определяет следующие четыре сложных типа данных:

- DATA_AND_TIME
  дата и время суток (в двоично-десятичном или в BCD-коде)
- STRING
  символьная строка, в которой может быть до 254 символов

- ARRAY
  массив ("field" - "поле", комбинация переменных одного типа)
- STRUCT
  структура (комбинация переменных разного типа)

Данные типов STRING, ARRAY и STRUCT должны быть заранее определены (предопределены) пользователем - для типа STRING задается длина символьной строки, а для типов ARRAY и STRUCT определяются состав и размеры данных-компонентов.

Вы можете объявить переменные сложных типов только в блоках глобальных данных, в экземплярных блоках данных как временные локальные данные или как параметры блока.

24.2.1 Тип данных DATA_AND_TIME

Данные типа DATA_AND_TIME заключают в себе информацию о дате и времени суток. Вместо идентификатора DATA_AND_TIME Вы можете использовать сокращение DT.

Объявление (Declaration)

varname : DATA_AND_TIME *:= pre-assignment;* *или* *varname :* DT *:= pre-assignment;*

varname - имя переменной

pre-assignment - предопределенное значение переменной

DATA_AND_TIME или DT - ключевые слова; они могут быть записаны и в нижнем регистре.

Предопределение (Pre-assignment)

Переменная может быть предопределена на этапе объявления (не как параметр блока в функции, как входной или выходной параметр в функциональном блоке или как временная переменная). Значение предопределения должно относиться к тому же типу данных, что и сама переменная. Значение предопределения должно иметь следующий формат:

DATA_AND_TIME#год-месяц-день-часы:минуты:секунды.миллисекунды
или
DT#год-месяц-день-часы:минуты:секунды.миллисекунды

Спецификация миллисекунд может быть опущена (см. табл. 24.3).

Применение

Переменные типа DT Вы можете применять в качестве соответствующим образом объявленных параметров блока (того же типа данных - DT или ANY); например, они могут быть скопированы с помощью системной функции SFC 20 BLKMOV. Для обработки этих переменных используются

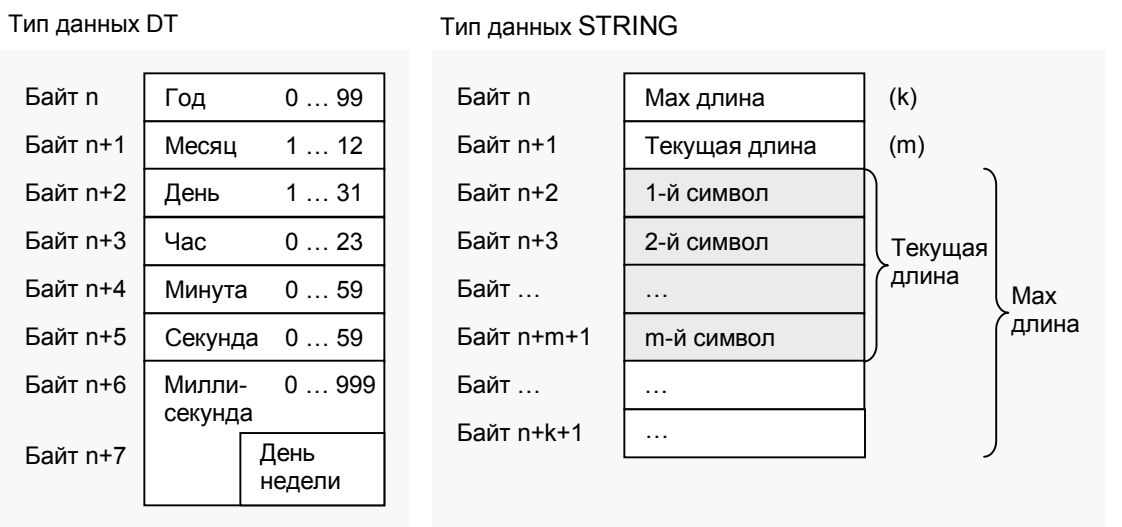
стандартные функциональные блоки ("IEC-functions" - "IEC-функции").

Структура переменных

Переменные типа DATA_AND_TIME или DT занимают 8 байтов (см. рис 24.4). Переменные начинаются на границе слов (начиная с байта с четным адресом). Все спецификации доступны в BCD-формате (в формате двоично-десятичного числа).

Таблица 24.3 Примеры объявления переменных типов DT и STRING

Имя	Тип данных	Начальное значение	Комментарий
Date1	DT	DT#1990-01-01-00:00:00	Минимальное значение переменной типа DT
Date2	DATA_AND_TIME	DATA_AND_TIME# 2089-12-31-23:59:59.999	Максимальное значение переменной типа DT
First_name	STRING[10]	'Jack'	Переменная типа STRING, заняты 4 символа из 10
Last_name	STRING[7]	'Daniels'	Переменная типа STRING, заняты все 7 символов из 7
NewLine	STRING[2]	'RL'	Переменная типа STRING, занята специальными символами
EmptyString	STRING[16]	' '	Переменная типа STRING, без заполнения



День недели: 1 = Воскресенье
7 = Суббота

Рис. 24.4 Структура переменных типов DT и STRING

24.2.2 Тип данных STRING

Данные типа STRING - символьная строка, в которой может быть до 254 символов.

Объявление (Declaration)

varname : STRING[Max число] *:= pre-assignment;*

varname - имя переменной

pre-assignment - предопределенное значение переменной

STRING - ключевое слово; оно может быть записано также в нижнем регистре.

Max число - параметр, определяющий максимальное число символов (длину строки), которые могут быть записаны в данную переменную. Таким образом, такая переменная может содержать от 0 до 254 символов. Данный параметр при записи также может быть опущен; при этом редактор будет использовать для такой переменной стандартную (максимально возможную) длину, равную 254 байта. При использовании функций FC редактор не допускает спецификацию параметра "Max число", и он будет использовать для такой переменной стандартную длину, равную 254 байта.

Предопределение (Pre-assignment)

Переменная может быть предопределена на этапе объявления (не как параметр блока в функции, как входной или выходной параметр в функциональном блоке или как временная переменная). Значение предопределения задается символами в ASCII-коде, заключенными в одинарные кавычки, или с предшествующим знаком доллара, в случае применения особых символов (см. табл. 24.3).

Если значение предопределения короче, чем объявленная максимальная длина, то неиспользованные позиции в поле переменной не занимаются. При дальнейшей обработке такой STRING-переменной учитываются только символы, занимающие позиции в поле переменной. Как значение предопределения допускается "пустая строка" ("emptystring").

Применение

Переменные типа STRING Вы можете применять в качестве параметров блоков типа STRING или ANY; например, они могут быть скопированы с помощью системной функции SFC 20 BLKMOV. Для обработки этих переменных используются стандартные функциональные блоки ("IEC-functions" - "IEC-функции").

Информацию об особенностях применения этих переменных в SCL Вы можете найти в разделе 27.5.2 "Назначения для переменных типов DT и STRING".

Структура переменных

Переменная типа STRING (строка символов) имеет максимальную длину 256 символов, из которых 254 байтов могут использоваться собственно для хранения данных. Переменные типа STRING начинаются на границе слов (начинаются с байта с четным адресом).

При применении STRING-переменных пользователь определяет для них максимальную длину. "Текущая" длина (то есть, фактически занимаемая строкой символов, равная числу "непустых" символов) определяется при

предопределении STRING-переменной или при ее обработке. В первом байте переменной содержится значение максимальной (заданной при объявлении переменной) длины; во втором байте содержится значение текущей длины переменной. Начиная с 3-го по счету байта располагаются данные - строка символов в формате ASCII (см. рис 24.4).

24.2.3 Тип данных ARRAY

Данные типа ARRAY представляет собой массив (или "поле" - "field"), содержащий в себе комбинацию фиксированного числа переменных одинакового типа.

Объявление (Declaration)

fieldname : ARRAY[*minIndex..maxIndex*] OF *datatype := pre-assignment;*
или
fieldname : ARRAY[$minIndex_1$ ..$maxIndex_1$,...,$minIndex_6$ ..$maxIndex_6$,]
OF *datatype := pre-assignment;*

ARRAY и OF - ключевые слова; они могут быть записаны также и в нижнем регистре.

fieldname - имя переменной типа ARRAY массив (поле);

pre-assignment - предопределенное значение переменной;

minIndex - нижний предел массива (поля);

maxIndex - верхний предел массива (поля);

Указанные пределы являются целыми числами типа INT из диапазона: -32768 … +32767; значение *maxIndex* должно быть больше или равно значению *minIndex.* Массив может быть многомерным. При этом каждая размерность должна также определяться своими нижним и верхним значениями. Максимальное число размерностей для массива ARRAY составляет 6:

$minIndex_1..maxIndex_1,.., minIndex_6..maxIndex_6$

datatype - параметр, определяющий тип данных для элементов массива; в рассматриваемом контексте это может быть любой тип данных, даже пользовательский, исключая собственно только тип ARRAY.

Предопределение (Pre-assignment)

Пользователь может предопределить на этапе объявления отдельные элементы массива (в отличие от параметров функции, проходных параметров в функциональном блоке или временных переменных). Тип данных предопределения должен соответствовать типу, заявленному для переменной.

Нет необходимости определять все элементы массива; если число предопределяемых элементов меньше числа элементов массива, то только первые элементы получают значения. Число предопределяемых элементов не может быть больше числа элементов массива. Значения

предопределения должны разделяться запятыми. Многократное назначение одного и того же значения может быть выполнено с помощью взятия значения в скобки и указания перед скобками коэффициента повторения (см. табл. 24.4).

Таблица 24.4 Примеры объявления переменных типа ARRAY

Имя	Тип данных	Начальное значение	Комментарий
Meas_Val	ARRAY[1..24]	0.4, 1.5, 11 (2.6, 3.0)	Переменная типа ARRAY с 24 элементами типа REAL
	REAL		
Time_Of_Day	ARRAY[-10..10]	21 (TOD#08:30:00)	Массив переменных TOD с 21 элементом
	TIME_OF_DAY		
Result	ARRAY[1..24,1..4]	96 (L#-0)	Двумерный массив с 96 элементами
	DINT		
Character	ARRAY[1..2,3..4]	2 ('a'), 2 ('b')	Двумерный массив с 4 элементами
	CHAR		

Применение

Переменные типа ARRAY Вы можете применять в качестве параметров блоков типа ARRAY с такой же структурой или как параметры блоков типа ANY. Например, они могут быть скопированы с помощью системной функции SFC 20 BLKMOV. Вы можете также определить отдельные элементы массива в качестве параметров блока типа ARRAY, если параметры блока относятся к тому же типу данных, что и эти элементы.

Если отдельные элементы массива относятся к простым типам данных, то Вы можете обрабатывать их с помощью обычных STL-операторов.

Доступ к элементу массива обеспечивается по имени массива и индексу в квадратных скобках. Индекс имеет фиксированное значение в STL и не может быть изменен во время выполнения программы (не может быть переменной).

Индекс в SCL может быть переменной или выражением с данными типа INT; значение индекса может быть изменено во время выполнения программы.

Многомерные массивы

Массив может быть многомерным. Максимальное число размерностей для массива ARRAY составляет 6. Многомерный массив подобен одномерному. Во время объявления переменных диапазоны индексов по каждой размерности отделяются друг от друга запятыми и заключаются в квадратные скобки.

Обеспечивая доступ к элементам многомерного массива в STL, Вы всегда должны указывать индексы всех размерностей массива. В SCL возможно организовывать доступ к части массива (см. раздел 27.5.4 "Назначение массивов").

Структура переменных

Переменные типа ARRAY всегда начинаются на границе слов (т.е., начинаются с байта с четным адресом). Переменная типа ARRAY занимает в памяти область до следующей границы слова.

Элементы массива типа BOOL начинаются с младшего бита; элементы массива типа BYTE и CHAR начинаются с расположенного справа байта (см. рис. 24.5, левая часть). При этом элементы массива следуют один за другим по порядку.

В многомерных массивах элементы хранятся, выстроенными в цепочки (в порядке следования размерностей), начиная с первой размерности (см. рис. 24.5, правая часть). Для элементов, имеющих длину 1 бит или 1 байт, новая размерность всегда начинается *со следующего байта*, а для элементов других типов новая размерность начинается *со следующего слова* (то есть, со следующего четного байта).

[1)] n = четный байт

Рис. 24.5 Структура переменной типа ARRAY

24.2.4 Тип данных STRUCT

Тип данных STRUCT определяет структуру данных, состоящую из фиксированного числа компонентов, которые могут относиться к различным типам данных.

Объявление (Declaration)

```
structname : STRUCT
    komp1name : datatype := pre-assignment;
    komp2name : datatype := pre-assignment;
    …
    END_STRUCT;
```

STRUCT и END_STRUCT - ключевые слова; они могут быть записаны также и в нижнем регистре.

structname - имя структуры;

pre-assignment - предопределенное значение переменной;

komp1name, komp2name - имена отдельных компонентов структуры;

datatype - тип данных имена отдельных компонентов структуры;

Предопределение (Pre-assignment)

Пользователь может предопределить на этапе объявления отдельные компоненты структуры (в отличие от параметров функции, проходных параметров в функциональном блоке или временных переменных). Тип данных предопределения должен соответствовать типу, заявленному для переменной.

Таблица 24.5 Пример объявления переменной типа STRUCT

Имя		Тип данных	Начальное значение	Комментарий
MotCont		STRUCT		Переменная простой структуры с 4 компонентами
	On	BOOL	FALSE (ЛОЖЬ)	Переменная MotCont.On типа BOOL
	Off	BOOL	TRUE (ИСТИНА)	Переменная MotCont.Off типа BOOL
	Delay	S5TIME	S5TIME#5s	Переменная MotCont.Delay типа S5TIME
	maxSpeed	INT	5000	Переменная MotCont.maxSpeed типа S5TIME
		END_STRUCT		

Применение

Переменные типа STRUCT Вы можете целиком применять в качестве параметров блоков типа STRUCT с такой же структурой или как параметры блоков типа ANY. Например, Вы можете скопировать с помощью системной функции SFC 20 BLKMOV содержимое переменной типа STRUCT. Вы можете также определить отдельные компоненты структуры в качестве параметров блока типа STRUCT, если параметры блока относятся к тому же типу данных, что и эти компоненты.

Если отдельные компоненты структуры относятся к простым типам данных, то Вы можете обрабатывать их с помощью обычных STL-операторов.

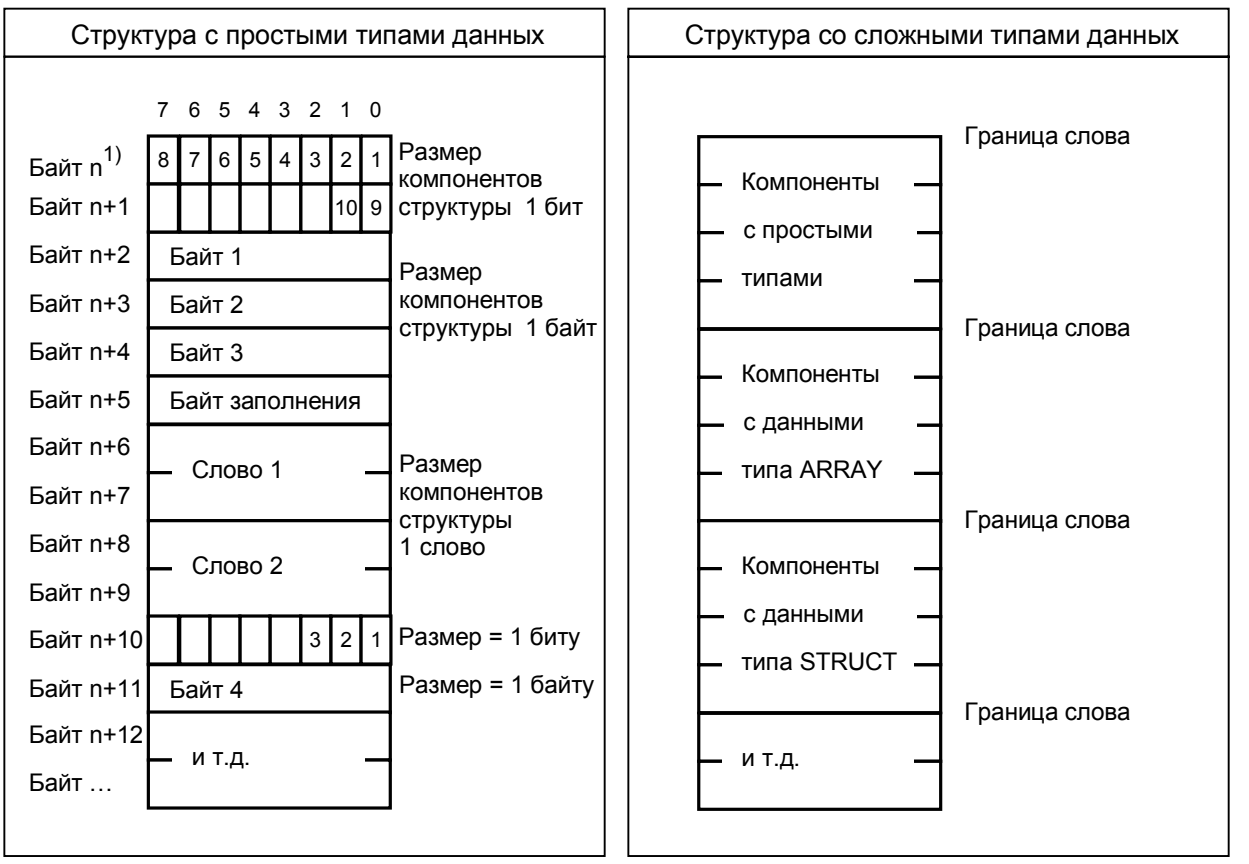
Доступ к компоненту структуры обеспечивается с использованием имени структуры и имени компонента, разделенных точкой.

Структура переменных

Переменные типа STRUCT всегда начинаются на границе слов (то есть, начинаются с байта с четным адресом); отдельные компоненты располагаются в памяти в том порядке, в котором они были описаны при объявлении. Переменная типа STRUCT занимает в памяти область до следующей границы слова.

Элементы массива типа BOOL начинаются с младшего бита; элементы массива типа BYTE и CHAR начинаются с расположенного справа байта (см. рис. 24.6). Компоненты других типов начинаются на границе слов.

Структура может быть *вложенной*. Вложенная структура - это такая структура, которая сама является компонентом другой структуры. Глубина вложения для структур может достигать шести вложений. Если отдельные компоненты структуры относятся к простым типам данных, то Вы можете обрабатывать их с помощью обычных STL-операторов. Отдельные имена компонентов структуры в полном имени разделяются точками.



1) n = четный байт

Рис. 24.6 Структура переменной типа STRUCT

24.3 Пользовательский тип данных

Пользовательский тип данных (UDT - "User-defined Data Type") определяет некоторую структуру данных (как комбинацию компонентов, которые могут относиться к различным типам данных), относящуюся к глобальным данным. Вы можете использовать пользовательский тип данных (UDT), если такая структура данных часто встречается в Вашей программе или Вы просто желаете дать имя структуре данных.

Вы можете создать структуру данных UDT либо при инкрементном редактировании программы, либо в текстовом редакторе, создавая исходный файл программы. Данные типа UDT могут быть созданы либо с использованием STL, либо с использованием SCL языков программирования одинаковым образом (Вы можете также запрограммировать данные UDT инкрементным способом с использованием языка SCL, если эти данные расположены в объекте *Blocks* (*Блоки*)).

Данные UDT являются глобальными, то есть, будучи один раз объявленными, они могут быть использованы во всех блоках. Доступ к данным UDT может быть организован с использованием имени (доступ по символу); для этого данным UDT Вы должны назначить абсолютный адрес в таблице символов (symbol table). Типу данных UDT (в таблице символов) соответствует абсолютный адрес.

Если Вы хотите передать в переменную структуру данных, определенную в UDT, назначьте переменной при ее объявлении пользовательский тип данных нужной структуры, как это делается при назначении "обычных" типов данных. Доступ к данным типа UDT может быть организован как по абсолютному адресу (UDT0 … UDT65535), так и по имени (символьная адресация).

Вы можете также определить тип данных UDT целому блоку данных. При программировании блока данных Вы можете назначить требуемый тип UDT блоку как структуре данных.

Пример "Данные фрейма сообщения" ("Message Frame Data") в разделе 26.4 "Краткое описание примера фрейма сообщения" объясняет, как работать с данными пользовательского типа.

24.3.1 Инкрементное программирование данных, определенных пользователем (UDT)

Вы можете создавать данные пользовательского типа (UDT) или с помощью утилиты SIMATIC Manager, выбрав сначала объект *Blocks (Блоки),* а затем - опции меню: *Insert -> S7 Block -> Data Type (Вставка -> S7 Block -> Тип данных)* или в редакторе, выбрав опции меню: *File -> New (Файл -> Создать)* и задав затем "UDTn" в строке "Имя объекта".

Двойной щелчок на объекте *UDT* в окне программы позволит открыть окно таблицы объявления данных, которая выглядит точно также, как таблица объявления блока данных. Структура UDT программируется точно также, как блок данных: с отдельными строками для имени (Name), типа (Type),

начального значения (Initial value) и комментария (Comment). Единственное различие между таблицами заключается в том, что здесь невозможно переключение к обзору данных (data view). (С помощью данных UDT Вы не можете создавать никаких переменных, а только собираете данные разных типов в одну структуру; по этой причине здесь не может быть фактических значений для компонентов структуры).

Начальные значения, которые Вы запрограммируете для структуры UDT, пересылаются в переменные при объявлении последних.

24.3.2 Применение данных UDT при создании исходных текстов программы

При программировании, ориентированном на создание исходных текстов программ, создание данных, определенных пользователем (UDT), выполняется при вводе данных типа STRUC ("структура"), заключенных между ключевыми словами TYPE и END_TYPE.

Объявление (Declaration)

```
TYPE udtname :
    STRUCT
    komp1name : datatype := pre-assignment;
    komp2name : datatype := pre-assignment;
    …
    END_STRUCT;
END_TYPE
```

TYPE, END_TYPE, STRUCT и END_STRUCT - ключевые слова; они могут быть записаны также и в нижнем регистре.

udtname - имя данных, определенных пользователем (UDT); вместо *udtname* Вы можете использовать абсолютный адрес UDT*n*.

pre-assignment - предопределенное значение переменной;

komp1name, komp2name - имена отдельных компонентов структуры;

datatype - тип данных отдельных компонентов структуры; здесь могут быть использованы все типы данных, кроме POINTER и ANY.

Предопределение (Pre-assignment)

Предопределение данных, определенных пользователем (UDT), производится точно также, как предопределение структуры STRUCT. структура данных также полностью соответствует типу данных STRUCT.

При предопределении данных, определенных пользователем (UDT), способ записи констант, принятый в языке программирования STL, применяется также и в SCL (см. обзор в разделе 3.7.3 "Простые типы данных").

Таблица 24.6 Пример объявления переменной типа UDT

Имя		Тип данных	Начальное значение	Комментарий
		STRUCT		
	Identifier	WORD	W#16#F200	Компонент UDT Identifier типа WORD
	Number	INT	0	Компонент UDT Number типа INT
	TimeofDay	TIME_OF_DAY	5000	Компонент UDT TimeofDay типа TIME_OF_DAY
		END_STRUCT		

25 Косвенная адресация

Косвенная адресация дает Вам возможность доступа к адресам, которые неизвестны до начала выполнения программы. С помощью косвенной адресации Вы также можете выполнять повторяющиеся разделы, части программы, такие, например, как циклы, и при этом при каждом проходе цикла Вы можете назначать разные адреса данных. В данной главе рассматриваются вопросы использования косвенной адресации при программировании на STL. Использование косвенной адресации при программировании на SCL рассматривается в разделе 27.2.3 "Косвенная адресация в SCL".

Так как при косвенной адресации адреса не вычисляются до выполнения программы, существует опасность того, что области памяти будут непреднамеренно перезаписаны. *В этом случае поведение программируемого контроллера может быть непредсказуемым! Будьте крайне осторожны при использовании косвенной адресации!*

Примеры для данной главы также представлены на дискете, прилагаемой к книге, в разделе "Variable Handling" ("Обработка переменных") в функциональном блоке FB 125 или в исходном файле Chap_25.

25.1 Указатели

Адрес при использовании косвенной адресации должен иметь такую структуру, в которой содержится адрес бита, адрес байта и, если необходимо, адрес области (*address area*). Поэтому такой адрес имеет специальный формат, имеющий название *указатель (Pointer).* Указатель используется для "указания" на адрес.

Система STEP 7 различает три типа указателей:

- указатели на область (area pointers)
  длина такого указателя составляет 32 бита и содержит определенный адрес;
- указатели на DB (DB pointers)
  длина такого указателя составляет 48 битов и содержит кроме адреса (как в указателе на область) номер блока данных;
- ANY-указатели (ANY pointers)
  длина такого указателя составляет 80 битов и содержит кроме информации, общей с указателем на DB, например, тип данных операнда.

Только первый тип указателей, т.е. указатели на область (area pointers), имеют значение для косвенной адресации. Указатели на DB и ANY-указатели используются при передаче параметров блока. Так как эти типы указателей содержат указатели на область, в данной главе описывается также и структура указателей на DB и ANY-указателей.

25.1.1 Указатели на область (area pointers)

Указатели на область (area pointers) содержат адрес операнда и, возможно, также адрес области, где находится операнд. Если адрес (идентификатор) области не указывается, то это *"внутризонный указатель" (area-internal pointer)*, иначе, если адрес (идентификатор) области присутствует, то это *межзонный указатель (area-crossing pointer)*.

Вы можете использовать указатель на область (area pointer) непосредственно и загружать (*load*) в аккумулятор или в адресный регистр, так как длина такого указателя составляет 32 бита. Ниже представлена форма записи для представления константы для двух случаев:

P#y.x для случая "внутризонного указателя" (area-internal pointer), например, P#22.0 и

P#Zy.x для случая "межзонного указателя" (area-crossing pointer), например, P#M22.0,

где x = адрес бита, x = адрес байта, Z = "адрес области". Как "адрес области" Вы используете идентификатор области. По значению бита 31 определяется тип указателя (внутризонный указатель или межзонный указатель) (см. рис. 25.1).

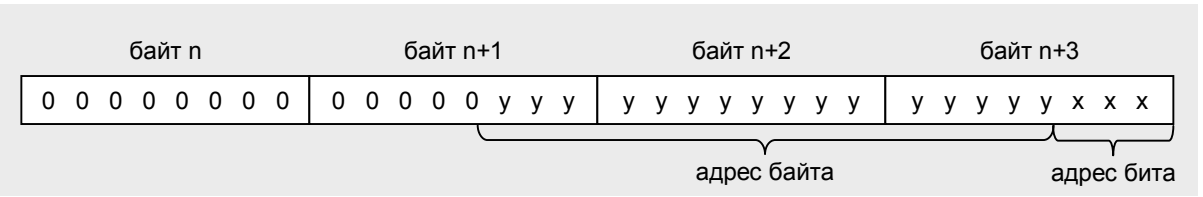
Указатель на область (area pointers) содержит адрес бита, который всегда должен быть определен, даже для числовых операндов. Для числовых операндов адрес бита равен 0 (нулю).

С помощью указателя на область P#M22.0 Вы можете, например, адресовать меркер (memory bit) M 22.0, а также байт в области меркеров (memory byte) MB 22, слово в области меркеров (memory word) MW 22 или двойное слово в области меркеров (memory duoble word) MW 22.

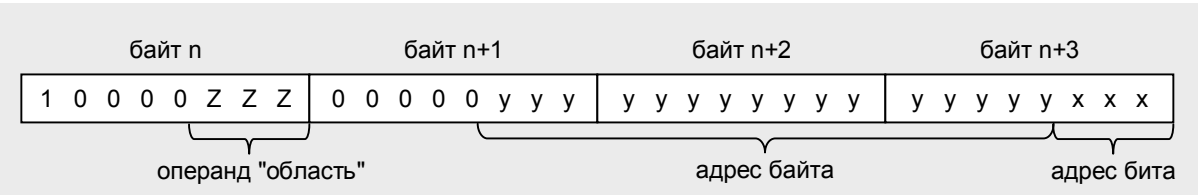
25.1.2 Указатели на DB (DB pointers)

Указатели на DB (DB pointers) содержит кроме адреса (как в указателе на область) номер блока данных в виде положительного целого числа типа INT. Этот номер идентифицирует блок данных, в то время как другая составная часть указателя содержит адрес в области глобальных данных или в области данных экземпляра. Во всех остальных случаях первые два байта указателя на DB будут содержать нуль.

Внутризонный указатель



Межзонный указатель



Адресная область:

0	0	0	Периферийные входы/выходы (P)
0	0	1	Входы (I)
0	1	0	Выходы (Q)
0	1	1	Меркеры (M)
1	0	0	Глобальные данные (DBX)
1	0	1	Экземплярные данные (DIX)
1	1	0	Временные локальные данные (L) [1]
1	1	1	Временные локальные данные вызывающего блока (V) [2]

[1] Не применяется при межзонной адресации

[2] Только при передаче параметра блока

Типы, используемые с ANY-указателем

Простые типы		**Сложные типы**
01	BOOL	0E DT
02	BYTE	13 STRING
03	CHAR	**Типы параметра**
04	WORD	17 BLOCK_FB
05	INT	18 BLOCK_FC
06	DWORD	19 BLOCK_DB
07	DINT	1A BLOCK_SDB
08	REAL	1C COUNTER
09	DATE	1D TIMER
0A	TOD	**Нуль-указатель (Zero-pointer)**
0B	TIME	
0C	S5TIME	00 NIL

Рис. 25.1 Структура указателя для косвенной адресации

Вы знакомы с форматом записи указателя по материалу, касающемуся вопроса полной адресации данных. Здесь также идентификатор блока данных и адрес данных разделяются точкой:

P#DataBlock.DataAddress

Пример:

P#DB 10.DBX 20.5

Вы не можете загружать данный указатель в аккумулятор; тем не менее, Вы можете применять его в качестве параметра блока с типом POINTER для указания адреса данных (кроме SCL). STEP 7 использует данный тип указателя для передачи фактических параметров.

25.1.3 ANY-указатели (ANY pointer)

Указатель ANY (ANY pointer) содержит кроме информации, общей с указателем на DB, тип данных адреса и коэффициент повторения. Это позволяет использовать такой указатель ANY для указания на область данных.

Указатель ANY используется в двух вариантах: для переменных с типом данных и для переменных с типом параметра. Если Вы используете указатель для переменных с типом данных, то указатель ANY содержит указатель на DB, тип и множитель повторения. Если Вы используете указатель для переменных с типом параметра, то указатель ANY содержит в дополнение к типу только номер вместо указателя на DB. Для функций таймера или счетчика тип повторяется в байте (n+6); байт (n+7) содержит B#16#00. Во всех остальных случаях эти два байта содержат значение W#16#0000.

Первый байт указателя ANY содержит синтаксическую структуру ID; в STEP 7 это всегда 10_{hex}. Тип определяет тип данных переменных, для которых данный указатель ANY применяется. Переменные простых типов, DT и STRING принимают тип, показанный на рисунке 25.1, и размер 1.

Если Вы используете переменную типа массива ARRAY или структуры STRUCT (а также пользовательского типа UDT) в качестве параметра ANY, то редактор создает указатель ANY для массива или структуры. Этот указатель ANY будет содержать идентификатор для байта BYTE (02_{hex}) как тип и число байтов, определяя этим самым длину переменной как ее размер (quantity). Тип данных отдельного элемента массива или структуры при этом не имеют значения. Указатель ANY с двойным количеством байтов таким образом указывает на массив из слов (WORD).

Исключение: указатель на массив, состоящий из элементов типа CHAR также используется с типом CHAR (03_{hex}).

Вы можете применять указатель ANY в качестве параметра блока типа "параметр ANY", если необходимо указать на переменную или на адресную область.

Представление констант для *типов данных* выполняется следующим образом:

P#[DataBlock.]Address Type Quantity = P#[БлокДанных.]Адрес Тип Размер.

Примеры:

- P#DB 11.DBX 30.0 INT 12
  область на 12 слов в блоке DB 11, начиная с DBB 30;
- P#M 16.0 BYTE 8
  область на 8 байт, начиная с MB 16;
- P#I 18.0 WORD 1
  слово входов IW 18;
- P#I 1.0 BOOL 1
  вход I 1.0.

Представление констант для *типов данных* выполняется следующим образом:

L# Number Type Quantity = L# Номер Тип Размер.

Примеры:

- L# 10 TIMER 1
  функция таймера T 10;
- L# 2 COUNTER 1
  функция счетчика Z 2.

Редактор использует указатель ANY, если тот согласован по типу и по размеру со спецификацией в представлении константы.

Необходимо отметить, что местоположение в памяти в указателе ANY для типов данных должно быть задано адресом бита.

Спецификация константы посредством указателя ANY имеет смысл, если Вы хотите получить доступ к области данных, для которой Вы не объявили переменной. В принципе, Вы можете также применить переменные или адреса в качестве параметра ANY. Например, представление "P#I 1.0 BOOL 1" идентично "I 1.0" или соответствующему символьному адресу.

С помощью типа параметра ANY Вы можете также объявить переменные во временных локальных данных. Вы можете использовать эти переменные для создания указателя ANY, который можно модифицировать во время выполнения программы (см. раздел 16.3.3 "Переменная "указатель ANY"").

Если Вы не выполняете никакого предопределения при объявлении параметра ANY в функциональном блоке, то редактор назначает значение 10_{hex} синтаксической структуре ID и 00_{hex} - всем остальным байтам. После этого редактор представляет такие (пустые) указатели ANY (при просмотре данных) следующим образом:

P#P0.0 VOID 0.

25.2 Типы косвенной адресации в STL

В данном разделе рассматриваются вопросы использования косвенной адресации в языке программирования STL; информацию по вопросам применения косвенной адресации в языке программирования SCL Вы можете найти в разделе 27.2.3 "Косвенная адресация в SCL".

25.2.1 Общая информация

Косвенная адресация возможна только с использованием абсолютных адресов. Вы не сможете применять косвенную адресацию с использованием символьных адресов (Вы должны также иметь возможность получать прямой доступ к отдельным элементам массива в STL). Если Вы желаете иметь косвенный доступ к переменной, Вы должны знать абсолютный адрес переменной. STL поддерживает прямой доступ к переменным (см. следующую главу).

Абсолютная адресация может применяться как:

- непосредственная адресация (immediate addressing);
- прямая адресация (direct addressing);
- косвенная адресация (indirect addressing).

Адресация посредством параметров блока - особая форма косвенной адресации: назначая фактические параметры параметрам блока, Вы определяете адреса, которые будут обработаны во время выполнения программы.

Мы имеем дело с *непосредственной адресацией (immediate addressing)*, когда численное значение (значение числа) задается непосредственно в операторе. Примером непосредственной адресации могут служить операция загрузки (load) значения константы в аккумулятор, операция сдвига фиксированного значения, а также установка или сброс результата логической операции посредством операций SET (УСТАНОВКА) и CLR (ОЧИСТКА).

С помощью *прямой адресации (direct addressing)* Вы получаете прямой доступ к адресу, например, A I 1.2 или L MW 122. Значение, с которым Вы хотите выполнить операцию, или значение, которое Вы хотите загрузить в аккумулятор, расположено по указываемому адресу, то есть в ячейке памяти. Вы адресуете данную ячейку памяти, указывая ее адрес непосредственно в операторе STL.

С помощью косвенной адресации (indirect addressing) STL-выражение указывает, где адрес может быть найден, вместо указания самого адреса. Мы различаем два типа косвенной адресации, различие между которыми определяется способом указания на адрес. Это:

- **косвенная адресация посредством памяти (memory-indirect addressing)**, которая использует адрес в системной памяти для определения адреса; пример: в выражении T QW [MD 220] адрес слова выходов, в которые должна быть сделана пересылка, размещен в двойном слове меркеров MD 220;
- **косвенная адресация посредством регистра (register-indirect addressing)**, которая использует адресный регистр для определения расположения адреса; пример: в выражении T QW [AR1,P#2.0] адрес слова выходов, в которые должна быть сделана пересылка, находится на 2 байта выше адреса, размещенного в адресном регистре AR1.

Вы можете использовать косвенную адресацию посредством регистра (register-indirect addressing) в двух вариантах: как *внутризонную косвенную адресацию посредством регистра (area-internal register-indirect addressing)* и как *межзонную косвенную адресацию посредством регистра (area-crossing register-indirect addressing).*

С помощью внутризонной косвенной адресации посредством регистра (area-internal register-indirect addressing) Вы можете запрограммировать в операторе адресную область, адрес для которой указывается в адресном регистре. Следовательно, адрес в адресном регистре может быть изменен в пределах адресной области (например: выражение L MW [AR1,P#0.0] обеспечивает загрузку [load] слова меркеров, адрес которого помещен в AR1).

С помощью межзонной косвенной адресации посредством регистра (area-crossing register-indirect addressing) Вы определяете в выражении только длину (размер) адреса (бит, байт, слово, двойное слово). Адрес для адресной области указывается в адресном регистре, при этом он может изменяться динамически (например: выражение L W [AR1,P#0.0] обеспечивает загрузку [load] слова, информация об адресной области и адрес которого находятся в AR1).

25.2.2 Косвенная адресация (Indirect Addresses)

Адреса, которые могут быть определены косвенно, могут быть разделены на две категории:

- адреса, для которых может быть назначен простой тип данных;
- адреса, для которых может быть назначен тип параметра.

Вы можете использовать косвенную адресацию посредством памяти и косвенную адресацию посредством регистра с первой категорией из указанных адресов, но со второй категорией адресов Вы можете использовать только косвенную адресацию посредством памяти (см. таблицу 25.1). Операнды, которые не могут иметь битовый адрес, также не требуют адреса бита в указателе, так что 16-разрядный номер вполне достаточен в качестве адреса (беззнаковое целое INT-число).

Таблица 25.1 Косвенная адресация

Адреса, которые могут быть определены косвенно	Адресация	Указатель
Периферийные входы/выходы (I/O), входы, выходы, меркеры, глобальные данные, экземплярные данные, временные локальные данные.	Косвенная адресация посредством памяти и косвенная адресация посредством регистра	Указатели на область: или внутризонный способ указания, или межзонный способ указания
Таймеры, счетчики, функции, функциональные блоки, блоки данных	Косвенная адресация посредством памяти	16-разрядный номер

Область указателей имеет теоретический размер в пределах от 0 до 65535 (байтовых адресов или номеров). На практике верхняя граница для адресов в каждом случае зависит от CPU. Диапазон адресов для битов (битовых адресов) занимает значения от 0 до 7.

25.2.3 Косвенная адресация посредством памяти (memory-indirect addressing)

При косвенной адресации посредством памяти (memory-indirect addressing) адрес указывается посредством адресованной ячейки памяти. Адрес должен иметь размер двойного слова, если требуется использовать указатель на область (area pointer), или же он должен иметь размер слова (WORD), если требуется при косвенной адресации использовать число в качестве указателя.

Адрес операнда может находиться в одной из ниже перечисленных адресных областей:

- область меркеров
  как абсолютный адрес или символьная переменная;
- L-стек (область временных локальных данных)
  как абсолютный адрес или символьная переменная;
- блок глобальных данных
  как абсолютный адрес;
  при использовании адресации глобальных данных пользователь должен обеспечить, чтобы требуемый блок данных был открыт с помощью DB-регистра; если, например, Вы указываете адрес адреса глобальных данных косвенным образом посредством двойного слова в глобальных данных, то все операции (вычисление адреса) должны быть произведены с учетом того, что все данные находятся в одном и том же блоке данных.
- экземплярный блок данных
  как абсолютный адрес или символьная переменная;
  существуют определенные ограничения при использовании экземплярных данных в качестве адреса; (об этом см. ниже).

Если Вы используете экземплярные данные в качестве адресов в функциях, обрабатывайте их точно таким же образом, как и адреса глобальных данных; при этом Вы должны использовать DI-регистр вместо DB-регистра. Символьная адресация не допускается в этом случае. Вы можете также использовать экземплярные данные в качестве адресов в функциональных блоках, но только если Вы скомпилировали их как блоки CODE_VERSION1 (без поддержки "мультиэкземплярности").

Косвенная адресация с указателем на область (area pointer)

Указатель на область (area pointer), используемый для косвенной адресации посредством памяти, всегда является внутризонным указателем (area-internal pointer). Это означает, что он содержит адрес байта и адрес бита. Если Вы хотите адресовать числовые операнды, то Вы должны всегда в качестве адреса бита указывать 0.

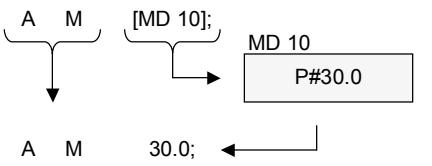
Пример: двойное слово MD 10 содержит указатель P#30.0. Выражение A M [MD 10] дает доступ к меркеру, адрес которого размещен в двойном слове MD 10; следовательно, с помощью этого выражения проверяется меркер M 30.0 (рис. 25.2).

С помощью выражения L MW [MD 10] Вы можете загрузить слово меркеров MW 30 в аккумулятор.

Вы можете использовать косвенную адресацию посредством памяти для всех бинарных операндов при использовании двоичных логических операций и операций с памятью, а также любых числовых операндов при использовании операций загрузки (load) и передачи (transfer).

Косвенная адресация посредством памяти с указателем на область

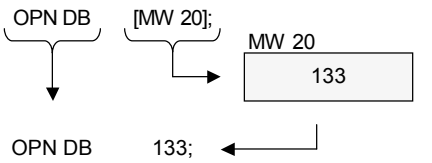
```
L    P#30.0;          //загрузка указателя в аккумулятор
                      //accumulator 1
T    MD 10;           //сохранение в двойном слове
                      //меркеров
L    MW [MD 10];      //назначение дискретного адреса
…
A    M [MD 10];       //назначение двоичного адреса
```



Косвенная адресация посредством памяти по номеру

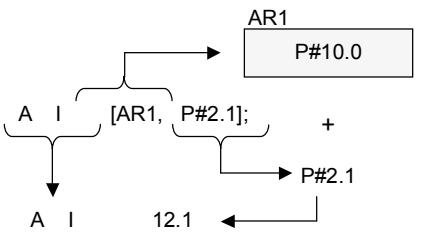
```
L    133;             //загрузка указателя в аккумулятор
                      //accumulator 1
T    MW 20;           //сохранение в слове меркеров

OPN DB [MW 20];       //открытие глобального блока DB
…
SP   T [MW 20];       //запуск таймера
```



Косвенная внутризонная адресация посредством регистра

```
LAR1 P#10.0;          //загрузка указателя в адресный
                      //регистр 1
…
L    MW [AR1,P#4.0]; //назначение дискретного
                      //адреса
…
A    I  [AR1,P#2.1];  //назначение двоичного адреса
```



Косвенная межзонная адресация посредством регистра

```
LAR1 P#M12.0;         //загрузка указателя в адресный
                      //регистр 1
…
L      B [AR1,P#4.0]; //назначение дискретного
                      //адреса
…
=      I  [AR1,P#0.7]; //назначение двоичного адреса
```

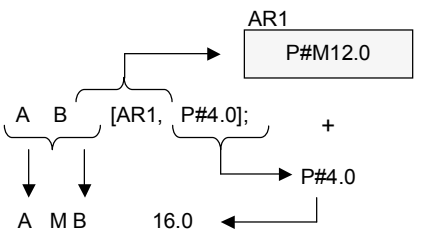


Рис. 25.2 Разновидности косвенной адресации

Косвенная адресация с использованием номера

Номер, используемый для косвенной адресации таймеров, счетчиков и блоков имеет длину 16 битов. Операнд, имеющий размер слова (WORD),

достаточен для хранения номера.

Пример: слово MW 20 содержит число (номер) 133. Выражение

OPN DB [MW 20]

открывает блок глобальных данных, номер которого размещен в слове меркеров MW 20. С помощью выражения

SP T [MW 20]

Вы можете запустить таймер T 133 в режиме импульса.

С помощью косвенной адресации Вы можете использовать любые операции с таймерами и счетчиками. Вы можете открывать блоки данных или с использованием DB-регистра (OPN DB […]), или с использованием DI-регистра (OPN DI […]). Если слово адреса содержит 0, то CPU выполняет NOP-операцию.

Вы можете косвенно адресовать вызовы кодовых блоков, используя UC FC[..] и CC FC[..] или UC FB[..] и CC FB[..]. Вызов блоков посредством операций UC или CC просто приводит к смене блока; при этом не происходит ни передачи параметров блока, ни открытия экземплярного блока.

25.2.4 Косвенная внутризонная адресация посредством регистра (Register-Indirect Area-Internal Addressing)

При косвенной внутризонной адресации посредством регистра (register-indirect area-internal addressing) адрес указывается посредством одного из двух адресных регистров. Содержимое адресного регистра является внутризонным указателем.

При косвенной адресации посредством регистра (register-indirect addressing) задается смещение в дополнение к информации адресного регистра. Это смещение добавляется к содержимому адресного регистра при выполнении соответствующей операции (без изменения содержимого адресного регистра). Указанное смещение имеет формат внутризонного указателя (area-internal pointer). Вы всегда должны задавать смещение при использовании данного вида адресации, и при этом Вы должны определять его константой. При косвенной адресации числовых операндов смещение всегда должно иметь нулевой битовый адрес (=0). При этом максимальное значение составляет P#8191.7.

Пример: Адресный регистр AR1 содержит указатель на область (area pointer) P#10.0 (при использовании оператора LAR1 Вы можете загрузить указатель непосредственно в адресный регистр AR1, об этом см. ниже). Выражение

A I [AR1.P#2.1]

добавляет указатель к адресному регистру AR1 и, таким образом, формирует адрес входа, состояние которого должно быть проверено. С помощью выражения

L MW [AR1.P#4.0]

Вы можете загрузить слово меркеров MW 14 в аккумулятор.

Внутризонная адресация с использованием межзонных указателей

Если адресный регистр содержит межзонный указатель (area-crossing pointer), и если Вы используете данный адресный регистр для операций с внутризонной адресацией данных, то адрес области в адресном регистре будет игнорироваться.

Пример: Следующие инструкции обеспечивают загрузку (load) в адресный регистр AR1 межзонного указателя (area-crossing pointer) на бит DBX 20.0 в глобальных данных, и после этого обеспечивается внутризонная адресация посредством адресного регистра AR1 в двойное слово меркеров. При выполнении следующего оператора выполняется загрузка в аккумулятор двойного слова MD 20.

```
LAR1    P#DBX 20.0;
L       MD[AR1,P#0.0];
```

25.2.5 Косвенная межзонная адресация посредством регистра (Register-Indirect Area-Crossing Addressing)

При косвенной межзонной адресации посредством регистра (register-indirect area-crossing addressing) адрес располагается в одном из двух адресных регистров. Содержимое адресного регистра является межзонным указателем (area-crossing pointer).

При межзонной адресации Вы дополнительно устанавливаете адресную область посредством указателя на область (area pointer) в адресный регистр. Если Вы используете косвенную адресацию Вы задаете только идентификатор ID для спецификации длины адреса: нет спецификации для бита, "B" - для байта, "W" - для слова, "D" - для двойного слова.

Как и при внутризонной адресации Вы используете в данном случае смещение, которое Вы должны определить с помощью фиксированного значения для битового адреса. Содержание адресного регистра не изменяется при задании смещения.

Пример: Адресный регистр AR1 содержит указатель на область (area pointer) P#12.0 (при использовании инструкции

```
LAR1 P#M12.0
```

Вы можете загрузить указатель непосредственно в адресный регистр AR1, об этом см. ниже). Инструкция

```
L B [AR1.P#4.0]
```

добавляет указатель P#4.0 к адресному регистру AR1 и, таким образом, формирует адрес слова меркеров, которое должно быть загружено (MB 16 в данном случае). С помощью инструкции

```
=  [AR1.P#0.7]
```

Вы можете назначить результат логической операции (т.е. RLO) меркеру M 12.7.

Вы не можете (в настоящее время) использовать межзонную адресацию для временных локальных данных (при этом CPU переходит в режим STOP). Вы должны использовать внутризонную адресацию, если

адресуемая область расположена в области временных локальных данных.

25.2.6 Резюме

Итак, в каких случаях использовать те или иные способы адресации? Если это возможно, то используйте косвенную внутризонную адресацию посредством регистра (register-indirect area-internal addressing). Язык программирования STL более всего приспособлен для использования такой адресации. Вы можете видеть указание на адресную область, доступ к которой необходим для выполнения операции. И кроме того, CPU обрабатывает программу с косвенной внутризонной адресацией посредством регистра (register-indirect area-crossing addressing) быстрее всего.

С другой стороны, косвенная адресация посредством памяти (memory-indirect addressing) обеспечивает преимущество, если в выполняющуюся программу включены больше, чем два указателя. Однако необходимо учитывать период "валидности" (корректности) указателя:

так, указатель в области меркеров может корректно использоваться без каких-либо ограничений при обработке программы целиком и даже на протяжении нескольких циклов сканирования программы;

указатель в блоке данных может корректно использоваться пока блок находится в открытом состоянии;

указатель в области временных локальных данных может корректно использоваться только во время обработки блока.

Если адресные области также должны быть доступны с использованием переменной адресации, то косвенная адресация посредством регистра является правильным выбором.

В таблице 25.2 представлены сравнительные данные вариантов косвенной адресации. Все показанные фрагменты программ ведут к одному и тому же результату - к установке выхода Q 4.7.

Таблица 25.2 Сравнительные данные вариантов косвенной адресации

Косвенная адресация посредством памяти	Косвенная внутризонная адресация посредством регистра	Косвенная межзонная адресация посредством регистра
L P#4.7	LAR1 P#4.7	LAR1 P#Q4.7
T MD 24		
S Q [MD 24]	S Q [AR1, P#0.0]	S [AR1, P#0.0]

25.3 Использование адресных регистров

Ниже показаны инструкции, которые возможны в языке программирования STL в связи с использованием адресных регистров: в форме списка инструкций (см. рис. 25.3.1) и в графической форме (см. рис. 25.3.2):

LAR1	-	Загрузка в адресный регистр AR1
LAR2	-	Загрузка в адресный регистр AR2
	P#Zy.x	межзонного указателя
	P#y.x	внутризонного указателя
LAR1	-	Загрузка в адресный регистр AR1 содержимого
LAR2	-	Загрузка в адресный регистр AR2 содержимого
	MD y	двойного слова меркеров
	LD y	двойного слова локальных данных
	DBD y	двойного слова глобальных данных
	DID y	двойного слова экземплярных данных [1)]
LAR1		Загрузка в адресный регистр AR1 содержимого аккумулятора Accu 1
LAR2		Загрузка в адресный регистр AR2 содержимого аккумулятора Accu 1
LAR1	AR2	Загрузка в адресный регистр AR1 содержимого адресного регистра AR2
TAR1	-	Пересылка содержимого адресного регистра AR1 в
TAR2	-	Пересылка содержимого адресного регистра AR2 в
	MD y	двойное слово меркеров
	LD y	двойное слово локальных данных
	DBD y	двойное слово глобальных данных
	DID y	двойное слово экземплярных данных [1)]
TAR1		Пересылка содержимого адресного регистра AR1 в аккумулятор Accu 1
TAR2		Пересылка содержимого адресного регистра AR2 в аккумулятор Accu 1
TAR1	AR2	Пересылка содержимого адресного регистра AR1 в адресный регистр AR2
CAR	-	Обмен содержимым между адресными регистрами
+AR1	-	Сложить содержимое аккумулятора Accu 1 и адресного регистра AR1
+AR2	-	Сложить содержимое аккумулятора Accu 1 и адресного регистра AR2
+AR1	P#Zy.x	Прибавить указатель к содержимому адресного регистра AR1
+AR2	P#y.x	Прибавить указатель к содержимому адресного регистра AR2

[1)] При использовании адресации такого типа необходимо учитывать ряд ограничений (см. ниже "Особенности косвенной адресации")

Рис. 25.3.1 Список инструкций языка программирования STL с использованием адресных регистров

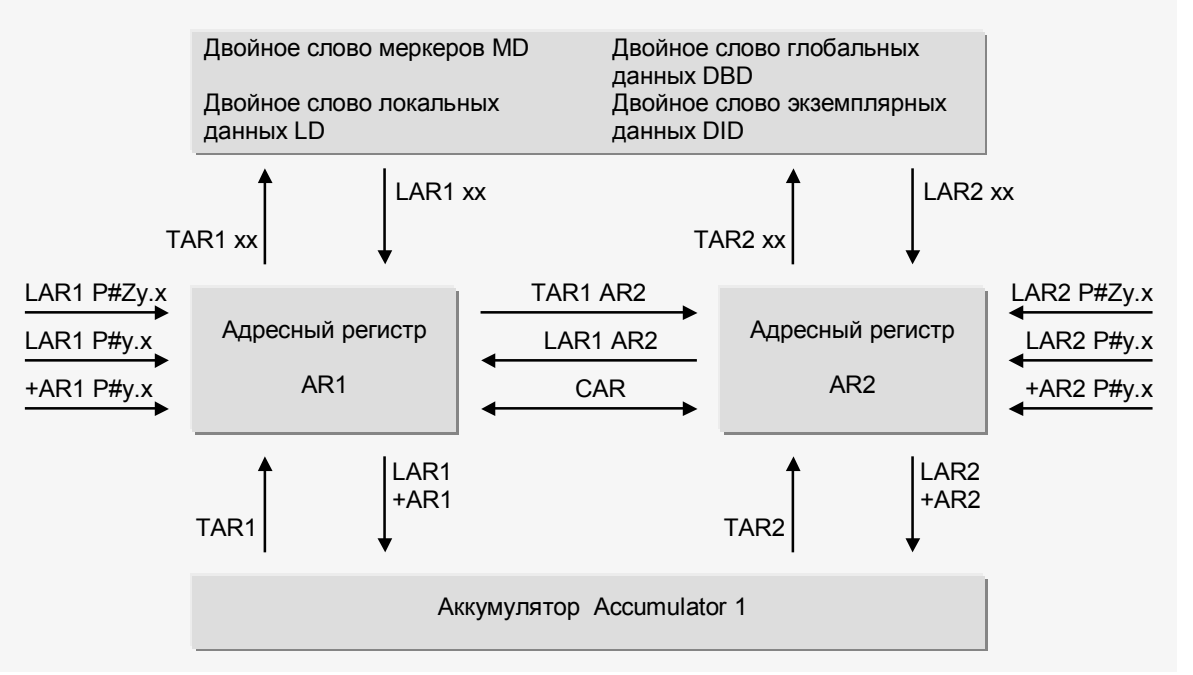


Рис. 25.3.2 Схема использования в STL адресных регистров

25.3.1 Загрузка в адресный регистр

Оператор LAR*n* загружает указатель на область (area pointer) в адресный регистр AR*n.* Исходными данными для загрузки в адресный регистр Вы можете выбрать внутризонный или межзонный указатель или двойное слово из области меркеров, из области временных локальных данных, из области глобальных данных или из области экземплярных данных. При этом содержимое двойного слова должно соответствовать формату указателя на область (area pointer).

В случае, если Вы не задаете адрес, оператор LAR*n* загружает содержимое аккумулятора accumulator 1 в адресный регистр AR*n.*

Если Вы используете инструкцию LAR1 AR2, то при выполнении данной инструкции происходит копирование содержимого адресного регистра AR2 в адресный регистр AR1.

Примеры:

```
LAR2  P#20.0;      //Загрузка указателя P#20.0 в AR2
L     P#24.0;      //Загрузка указателя P#24.0 в Accu1
LAR1  ;            //Загрузка содержимого аккумулятора
                   //Accumulator 1 в AR1
LAR1  MD 120;      //Загрузка содержимого MD 120 в AR1
LAR1  AR2;         //Загрузка содержимого AR2 в AR1
```

25.3.2 Пересылка из адресного регистра

Оператор TAR*n* выполняет пересылку указателя на область (area pointer) целиком из адресного регистра AR*n.* Как область назначения Вы можете выбрать двойное слово из области меркеров, из области временных локальных данных, из области глобальных данных или из области экземплярных данных.

В случае, если Вы не задаете адрес, оператор TAR*n* пересылает содержимое адресного регистра AR*n* в аккумулятор accumulator 1. При этом предыдущее содержимое аккумулятора accumulator 1 сдвигается в аккумулятор accumulator 2; предыдущее содержимое аккумулятора accumulator 2 теряется. Содержимое аккумуляторов accumulator 3 и accumulator 4 остается без изменения.

Если Вы используете инструкцию TAR1 AR2, то при выполнении данной инструкции происходит копирование содержимого адресного регистра AR1 в адресный регистр AR2.

Примеры:

```
TAR2  MD 140;      //Пересылка содержимого AR2 в MD 140
TAR1  ;            //Пересылка содержимого AR1 в
                   //аккумулятор Accumulator 1
TAR1  AR2;         //Пересылка содержимого AR1 в AR2
```

25.3.3 Обмен содержимым между адресными регистрами

Оператор CAR выполняет обмен содержимым между адресными регистрами AR1 и AR2.

Пример:

Пусть 8 байтов данных пересылаются между областью меркеров (MB 100) и областью данных (DB 20.DBB 200). Направление пересылки указывается в меркере M 126.6. Если меркер M 126.6 имеет состояние "0", то адресные регистры обмениваются содержимым. Если необходимо таким способом организовать пересылку данных между двумя блоками данных, загрузите оба регистра блока данных (с помощью инструкций OPN DB и OPN DI) содержимым адресных регистров и производите обмен данными с помощью оператора TDB.

```
      LAR1  P#M100.0;
      LAR2  P#DBX200.0;
      OPN   DB 20;
      A     M 126.6;
      JC    OV;
      CAR   ;
OV:   L     D[AR1,P#0.0];
      T     D[AR2,P#0.0];
      L     D[AR1,P#4.0];
      T     D[AR2,P#4.0];
```

Примечание: системная функция SFC 20 BLKMOV применяется для передачи больших объемов данных.

25.3.4 Операция сложения с содержимым адресного регистра

Пользователь может запрограммировать операцию сложения некоторого значения с содержимым адресного регистра для того, чтобы, например, инкрементировать адрес при каждом следующем проходе цикла в программе. Вы можете задать прибавляемое значение или как константу (как внутризонный указатель), или это значение может быть расположено в правом слове аккумулятора accumulator 1. Тип указателя в адресном регистре (внутризонный указатель или межзонный указатель) и адресной области при этом сохраняются.

Операции сложения с содержимым указателей

Инструкции +AR1 P#y.x и +AR2 P#y.x прибавляют указатель к содержимому указанного адресного регистра AR1 или AR2.

Необходимо отметить, что в данных инструкциях указатель на область (area pointer) может иметь максимальную величину P#4095.7. Если аккумулятор содержит величину, большую, чем P#4095.7, то число интерпретируется как инвертированное число с фиксированной запятой (two's complement) и вычитается.

Пример:

Пусть необходимо сравнить значение в машинном слове в области данных с некоторой величиной. Если сравниваемая величина больше, чем значение в области данных, то устанавливается меркер (получает значение "1"), в противоположном случае меркер сбрасывается (получает значение "0").

```
      OPN  DB 14;
      LAR1 P#DBX20.0;
      LAR2 P#M10.0;
      L    Quantity_Data;
Loop: T    LoopCounter;
      L    ComparisonVal;
      L    W[AR1,P#0.0];
      >I   ;
      =    [AR2,P#0.0];
      +AR1 P#2.0;
      +AR2 P#0.1;
      L    LoopCounter;
      LOOP Loop;
```

Операции сложения с содержимым аккумулятора

Инструкции +AR1 и +AR2 интерпретируют значение в аккумуляторе Accu1

как целое число формата INT, представляют его с соответствующим знаком как 24-разрядное число и прибавляют к содержимому указанного адресного регистра AR1 или AR2. Таким образом, указатель может быть также уменьшен. Нарушение верхней границы допустимого диапазона байтового адреса (0 … 65535) не имеет значения: "избыточные" биты отрезаются (см. рис. 25.4).

Необходимо отметить, что битовый адрес размещается в битах с номерами от 0 до 2. Если необходимо инкрементировать байтовый адрес в аккумуляторе Accumulator 1, то увеличение надо начинать с бита 3 (то есть, прибавляемое значение Вы должны сдвинуть на три разряда влево).

Пример: Пусть в блоке данных DB 14 16 байтов, адреса которых вычисляются с помощью указателя в двойном слове меркеров MD 220 и смещения в байте меркеров MB 18, должны быть очищены. Перед прибавлением к AR1 содержимое MB 18 должно быть смещено вправо на 3 разряда (SLW 3).

```
OPN   DB 14;
LAR1  MD 220;
L     MB 18;
SLW   3;
+AR1  ;
L     0;
T     DBD[AR1,P#0.0];
T     DBD[AR1,P#4.0];
T     DBD[AR1,P#8.0];
T     DBD[AR1,P#12.0];
```

Примечание:

Системная функция SFC 21 FILL используется для заполнения больших областей определенным значением бита.

Аккумулятор Accumulator 1

Адресный регистр

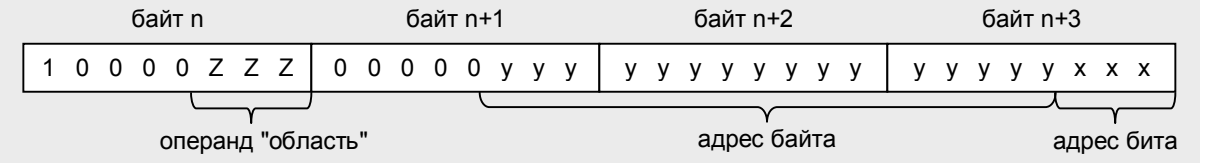


Рис. 25.4 Прибавление значения к адресному регистру

25.4 Особенности косвенной адресации

25.4.1 Использование адресного регистра AR1

В языке программирования STL адресный регистр AR1 используется для обращения к параметрам блока, которые пересылаются как DB-указатели. При использовании функций адресный регистр AR1 может применяться для обращения ко всем параметрам блока сложных типов данных, а в случае функциональных блоков это касается входных/выходных параметров сложных типов.

При использовании доступа к параметрам блока этого вида, например, для того, чтобы проверить битовый компонент структуры или чтобы записать целое INT-значение в элемент массива, содержание адресного регистра AR1 изменяется и также, изменяется содержание DB-регистра. Это происходит тогда, когда Вы передаете параметры блока данного типа в вызываемые блоки.

Если Вы используете адресный регистр AR1 (помимо косвенной адресации), при этом не должен быть возможен доступ к параметру блока (в соответствии с изложенным выше материалом). Другими словами, Вы должны сохранять содержимое адресного регистра AR1 до обращения к параметрам и вновь загружать регистр после выполнения операции доступа.

Пример:

Вы загружаете указатель в AR1 и используете этот адресный регистр для косвенной адресации. В определенный момент Вы хотите загрузить (load) значение компонента структуры *Motor.Act.* Перед загрузкой *Motor.Act* Вы сохраняете содержимое DB-регистра и адресного регистра AR1; после выполнения операции загрузки вышеназванного компонента структуры, Вы восстанавливаете состояния регистров (см. рис. 25.5, верхняя часть).

25.4.2 Использование адресного регистра AR2

При использовании функциональных блоков с возможностью работы в "мультиэкземплярном режиме" (блок версии 2), адресный регистр AR2 используется в STEP 7 как "базовый адресный регистр" ("Base address register") для экземплярных данных. При вызове экземпляра адресный регистр AR2 содержит P#DBX0.0 и при всех обращениях к параметрам блоков или к статическим локальным данным в FB используется косвенная внутризонная адресация области DI посредством данного регистра.

При вызове локального экземпляра производится инкрементирование "базового адреса" ("base address") при помощи +AR2 P#y.x, так что обращение к данным может быть выполнено относительно данного адреса внутри вызываемого функционального блока, который использует экземплярный блок данных вызывающего функционального блока. Таким образом, функциональные блоки могут быть вызваны и как автономные экземпляры, и как локальные экземпляры (к тому же в любой точке в функциональном блоке и даже несколько раз).

```
//********** Сохранение адресного регистра AR1 *********
...
VAR_TEMP
   AR1_Memory   : DWORD;
   DB_Memory    : WORD;
END_VAR
...
//Косвенная адресация с регистром AR1 и DB-регистром
LAR1  P#y.x;
OPN   DB z;
...
//Сохранение содержимого регистров
L     DBNO;
T     DB_Memory;
TAR1  AR1_Memory;
//Обращение к параметрам блока сложного типа
L     Motor. Act;
//Восстановление содержимого регистров
OPN   DB [DBMemory];
LAR1  AR1_Memory;
T     DBW[AR1,P#0.0];       //Сохранение   загруженного
                            //значения
//********** Сохранение адресного регистра AR2 *********
...
VAR_TEMP
   AR2_Memory : DWORD;
   DI_Memory  : WORD ;
END_VAR
...
//Сохранение содержимого регистров
L     DINO;
T     DI_Memory;
TAR2  AR2_Memory;
//Косвенная адресация с регистром AR2 и DI-регистром
LAR2  P#y.x;
OPN   DI z;
...
L     DIW[AR2,P#0.0]
...
//Восстановление содержимого регистров
OPN   DI [DI_Memory];
LAR2  AR2_Memory;
```

Рис. 25.5 Примеры сохранения адресных регистров AR1 и AR2

Если Вы программируете функциональный блок как блок версии 1,то есть, без "мультиэкземплярного режима", то STEP 7 не использует адресный регистр AR2.

Итак, Вы хотите использовать адресный регистр AR2 в функциональном блоке с возможностью работы в "мультиэкземплярном режиме", Вы должны сначала сохранить содержимое адресного регистра AR2 и DI-регистра до использования и вновь восстанавливать данные в регистрах после выполнения операции доступа. Вы не должны программировать никаких обращений к параметрам блока или к статическим локальным данным в области, в которой Вы используете адресный регистр AR2.

При использовании адресного регистра AR2 внутри функций никаких ограничений не накладывается.

Пример:

В функциональном блоке Вам необходимо использовать регистр AR2 и DI-регистр для выполнения косвенной адресации. Прежде всего Вы должны сохранить содержимое DI-регистра и адресного регистра AR2. При этом Вы не должны выполнять обращения к параметрам блоков или к статическим локальным данным до того, пока Вы не восстановите содержимое регистров (см. рис. 25.5, нижняя часть).

25.4.3 Ограничения на использование статических локальных данных

При использовании функциональных блоков, скомпилированных с CODE_VERSION1 (без "мультиэкземплярного режима"), Вы можете использовать все операторы (инструкции), описанные в данной главе без ограничений.

При использовании функциональных блоков с возможностью работы в "мультиэкземплярном режиме", редактор (Editor) обращается к экземплярным данным посредством адресного регистра AR2; то есть, все эти операции доступа носят косвенный характер. Это же касается косвенной адресации и обработки адресных регистров. Если Вы используете абсолютную адресацию для экземплярных данных, в которых Вы храните указатели на область (area pointers), редактор использует абсолютные адреса. Тем не менее, как только Вы начнете использовать символьную адресацию, редактор отбросит эту часть программы как попытку "двойной косвенной адресации" (duoble indirect addressing).

В таблице 25.3 представлены два таких примера: если Вы используете косвенную адресацию посредством памяти в функциональных блоках с возможностью работы в "мультиэкземплярном режиме", то Вы не можете непосредственно использовать указатель, который Вы желаете сохранять в статических локальных данных. Вы должны скопировать указатель во временные локальные данные, и после этого можете работать с ним. При этом Вы не можете непосредственно загрузить (load) указатель на статические локальные данные в адресный регистр и не можете также переслать содержимое адресного регистра непосредственно в указатель (второй пример).

Таблица 25.3 Различие в программировании FB версий 1 и 2

Для FB, скомпилированных с параметром CODE_VERSION1 (без "мультиэкземплярного режима")	Для FB, скомпилированных с параметром CODE_VERSION2 (с "мультиэкземплярным режимом")
VAR sPointer : DWORD; END_VAR	VAR sPointer : DWORD; END_VAR VAR_TEMP tPointer : DWORD; END_VAR
L MW[sPointer];	L sPointer; T tPointer; L MW[tPointer];
LAR1 sPointer;	L sPointer; LAR1;
TAR1 sPointer;	TAR1; T sPointer;

26 Прямой доступ к переменным

В данной главе рассматривается, как реализовать прямое обращение к локальным переменным с использованием их абсолютных адресов. "Обычные" STL-операторы позволяют обращаться к локальным переменным простых типов. Локальные переменные сложных типов, а также параметры блоков типа POINTER или ANY не могут быть обработаны "целиком". Для обработки таких переменных Вы сначала должны вычислить начальный адрес, по которому переменная хранится, а затем Вы можете обращаться к отдельным ее частям, используя косвенную адресацию. Таким же путем Вы можете также обрабатывать параметры блоков, относящиеся к сложным типам данных.

Примеры для данной главы также представлены на дискете, прилагаемой к книге, в библиотеке STL_Book в разделе "Variable Handling" ("Обработка переменных") в функциональном блоке FB 126 или в исходном файле Chap_26.

26.1 Загрузка адреса переменной

Следующие инструкции дают начальный адрес локальной переменной:

```
L     P#name;
LAR1  P#name;
LAR2  P#name;
```

Здесь операнд *name* - это имя локальной переменной. В этих инструкциях межзонный указатель (area-crossing pointer) загружается в аккумулятор Accumulator 1, в адресный регистр AR1 и в адресный регистр AR2 соответственно. Межзонный указатель содержит адрес первого байта переменной. Если имя *name* не может быть идентифицировано однозначно как локальная переменная, тогда добавьте символ "#" перед именем, так что инструкция изменится, например, следующим образом:

```
L     P##name;
```

В таблице 26.1 представлены области, в которых, в зависимости от вида блока, может быть размещена переменная *name*.

В функциях FC адрес параметра блока не может быть загружен непосредственно в адресный регистр.

Поэтому здесь следует использовать аккумулятор Accumulator 1, например:

```
L     P#name;
LAR1  ;
```

В функциональных блоках (FB), скомпилированных с ключевым словом CODE_VERSION1 (то есть, как блок версии 1, без "мультиэкземплярного режима"), абсолютный адрес переменной экземпляра может быть загружен (load).

В функциональных блоках (FB), скомпилированных как блок версии 2, то есть, с возможностью "мультиэкземплярного режима", абсолютный адрес (относительно адресного регистра AR2) переменной экземпляра может быть загружен (load), если переменная относится к статическим локальным данным или является параметром блока. Если необходимо вычислить абсолютный адрес переменной в экземплярном блоке данных, Вы должны сложить внутризонный указатель (area-internal pointer) (только адрес) из AR2 с загруженным адресом переменной.

Таблица 26.1 Разрешенные области для размещения адресов переменных

Инструкция	Тип данных переменной *name*	Блок			
		OB	FC	FB V1	FB V2
L P#*name*	Временные локальные данные	x	x	x	x
	Статические локальные данные	-	-	x	x [1)]
	Параметр блока	-	x	x	x [1)]
LAR*n* P#*name*	Временные локальные данные	x	x	x	x
	Статические локальные данные	-	-	x	x [1)]
	Параметр блока	-	-	x	x [1)]

[1)] Адрес переменной, связанный с адресным регистром AR2

Пример 1:

Загрузить адрес переменной в адресный регистр AR1.

```
TAR2  ;
AD    DW#16#00FF_FFFF;
LAR1  P#name;
+AR1  ;
```

Здесь адрес в AR2 загружается в аккумулятор и добавляется к содержимому AR1 с помощью оператора +AR1. В результате чего адрес переменной *#name* загружается в адресный регистр AR1.

Пример 2:

Загрузить адрес переменной в аккумулятор Accumulator 1.

```
TAR2  ;
AD    DW#16#00FF_FFFF;
L     P#name;
+D    ;
```

В данном примере в результате выполнения данного фрагмента программы в аккумуляторе Accumulator 1 оказывается адрес переменной *#name*.

Сложение с внутризонным указателем (area-internal pointer) может быть пропущено, если указатель имеет значение P#0.0. Это для случая, если Вы не используете функциональный блок как локальный экземпляр.

Необходимо отметить, что при выполнении инструкции "LAR2 P#name" перезаписывается содержимое адресного регистра AR2, который применяется при использовании функциональных блоков с возможностью "мультиэкземплярного режима" как "базовый адресный регистр" при адресации экземплярных данных!

С помощью этих (LAR*n*) операторов загрузки (load) Вы можете организовать доступ только к одной переменной целиком, но не можете организовать доступ к отдельным компонентам массивов, структур или локальных экземпляров. Вы не можете с помощью этих операторов загрузки получить доступ к переменным в блоках глобальных данных или в адресных областях входов, выходов, периферийных входов/выходов и в адресных областях меркеров.

В таблице 26.2 показано, как рассчитываются адреса переменных типов INT и STRING в статических локальных данных и как использовать эти адреса. Если Вы используете образец программы в функциональном блоке, который Вы вызываете как локальный экземпляр, Вы должны прибавить базовый адрес к адресу переменной, как показано выше.

Таблица 26.2 Загрузка адреса переменной (примеры)

```
//Объявление переменных (функциональный блок не является локальным
//экземпляром!)
//Назначение переменных начинается с адреса P#0.0
VAR
  Field     : ARRAY[1..22] OF BYTE;  //переменная ARRAY занимает 22 байта
  Number    : INT := 123;            //переменная INT занимает 2 байта
  FirstName : STRING := 'Joane';     //переменная STRING занимает 5 байтов
END_VAR
```

`LAR1  P#Number;`	Загрузка начального адреса *Number* в AR1 Регистр AR1 теперь содержит P#DIX22.0
`L     W[AR1,P#0.0];`	Соответствует выражению L DIW 22 *или* L *Number*
`LAR1  P#FirstName;`	Загрузка начального адреса *FirstName* в AR1 Регистр AR1 теперь содержит P#DIX24.0
`L     B[AR1,P#0.0];`	Загрузка 1-го байта (максимальная длина строки символов) в аккумулятор Accu 1
`L     B[AR1,P#2.0];`	Загрузка 3-го байта (1-го байта данных) в аккумулятор Accu 1
`L     'John'` `T     D[AR1,P#2.0];`	Запись имени 'John' в строку символов
`L     4;` `T     B[AR1,P#1.0];`	Корректировка текущего размера строки символов (=4) Переменная *FirstName* теперь содержит 'John'

26.2 Хранение переменных

26.2.1 Хранение переменных в блоках глобальных данных

Редактор охраняет отдельные переменные в блоках данных в порядке их объявления. При этом должны соблюдаться следующие правила:

- Первая "битовая" переменной (с длиной в 1 бит) из непрерывной последовательности объявленных переменных размещается в бите 0 следующего байта; за первой битовой переменной размещаются следующие битовые переменные.
- "Байтовая" переменная (с длиной в 1 байт) из непрерывной последовательности объявленных переменных размещается в следующем байте.
- Переменные, имеющие размер одного "машинного" слова и имеющие размер двойного слова, всегда размещаются, начиная с границы слова, то есть в байте с четным адресом.
- Переменные типов DT и STRING также всегда размещаются, начиная с границы слова.
- Переменные типа ARRAY также всегда размещаются, начиная с границы слова, и заполняют область до следующей границы слова. Это касается также битовых и байтовых массивов. Элементы массивов простых типов сохраняются, как описано выше. Элементы массивов сложных типов размещаются, начиная с границы слова. Каждая размерность массива размещается с выравниванием как автономный массив.
- Переменные типа STRUCT также всегда размещаются, начиная с границы слова, и заполняют область до следующей границы слова. Это касается также "чисто" битовых и байтовых структур. Компоненты структур простых типов сохраняются, как описано выше. Элементы структур сложных типов размещаются, начиная с границы слова.

Группируя отдельные битовые переменные и компонуя байтовые переменные, Вы можете оптимизировать размещение данных в блоках данных.

На рисунке 26.1 Вы можете видеть один пример неоптимального и один пример оптимального хранения данных.

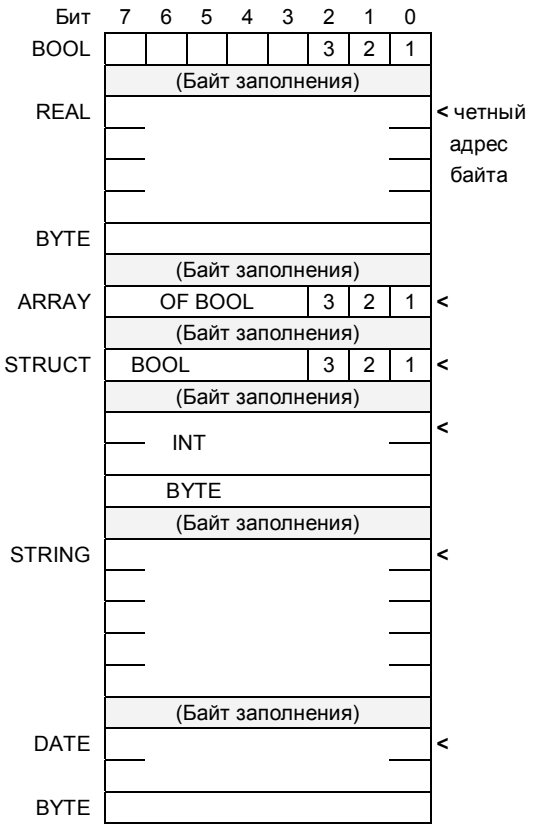
Необходимо отметить, что редактор всегда "заполняет" переменные типа ARRAY и типа STRUCT до следующей границы слова, что означает, что ни битовые, ни байтовые переменные не могут быть размещены для хранения в остающихся незанятыми областях памяти. Тем не менее, Вы можете оптимизировать размещение переменных внутри структуры.

Неоптимальное размещение данных

```
DATA_BLOCK StorageNonOptimized
STRUCT
  Bit1      : BOOL;
  Bit2      : BOOL;
  Bit3      : BOOL;
  Real1     : REAL;
  Byte1     : BYTE;
  Bit_field : ARRAY [1..3] OF BOOL;
  Structure : STRUCT
    S_Bit1  : BOOL;
    S_Bit2  : BOOL;
    S_Bit3  : BOOL;
    S_Int1  : INT;
    S_Byte  : BYTE;
  END_STRUCT;
  Character : STRING[3];
  Date1     : DATE;
  Byte2     : BYTE;
END_STRUCT
BEGIN
END_DATA_BLOCK
```

Оптимальное размещение данных

```
DATA_BLOCK StorageNonOptimized
STRUCT
  Bit1      : BOOL;
  Bit2      : BOOL;
  Bit3      : BOOL;
  Byte1     : BYTE;
  Real1     : REAL;
  Bit_field : ARRAY [1..3] OF BOOL;
  Structure : STRUCT
    S_Bit1  : BOOL;
    S_Bit2  : BOOL;
    S_Bit3  : BOOL;
    S_Byte  : BYTE;
    S_Int1  : INT;
  END_STRUCT;
  Character : STRING[3];
  Byte2     : BYTE;
  Date1     : DATE;
END_STRUCT
BEGIN
END_DATA_BLOCK
```



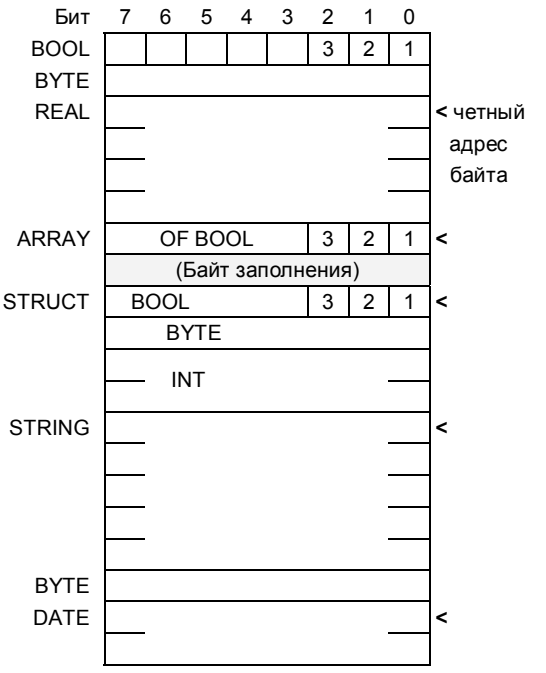


Рис. 26.1 Примеры размещения данных в блоке данных

26.2.2 Хранение переменных в блоках экземплярных данных

Редактор сохраняет отдельные переменные в экземплярах DB, размещая их в следующем порядке:

- Входные параметры;
- Выходные параметры;
- Входные/Выходные параметры;
- Локальные переменные (в том числе и переменные локальных экземпляров).

Порядок расположения переменных в соответствующей области памяти соответствует порядку их объявления. Каждая объявленная область (для одинаково объявленных переменных) начинается на границе "машинного" слова, то есть с байта с четным адресом. Внутри таких областей отдельные переменные располагаются, как описано в предыдущем разделе (как в глобальном блоке данных).

На рисунке 26.2 показан пример размещения переменных в экземплярном блоке данных.

26.2.3 Хранение переменных в области временных локальных данных

Размещение переменных в области временных локальных данных (в L-стеке) соответствует размещению в блоке глобальных данных. Размещение данных всегда начинается с байта 0 (относительный адрес).

Необходимо отметить, что в организационном блоке первые 20 байтов должны быть заняты стартовой информацией. Даже если Вы не используете стартовую информацию, первые 20 байтов должны быть объявлены (например, массивом, имеющим размер 20 байтов.).

Сам редактор также использует локальные данные, например, при передаче параметров в вызываемый блок. Редактор применяет объявленные с символьными именами временные локальные данные и временные локальные данные, которые он использует для выполнения своих функций, в порядке их объявления или в порядке, определяющем их использование. Временные локальные данные с абсолютной адресацией не используются здесь, так как при этом может произойти "перекрытие" данных (замена значений), если Вы не контролируете, какие именно данные использует редактор. Если Вы хотите или должны организовать доступ к локальным данным, используя абсолютную адресацию, то Вы, например, можете оставить свободным массив (field) в первой позиции списка объявленных локальных данных для задания требуемого числа байтов (слов, двойных слов). Затем Вы сможете обращаться, используя абсолютную адресацию, к данной области. Что касается организационных блоков, то здесь Вы должны будете разместить указанный массив после первых 20 байтов, которые должны быть отведены под стартовую информацию блока.

```
FUNCTION_BLOCK StorageExample
VAR_INPUT
  E_Bit1      : BOOL;
  E_Bit2      : BOOL;
  E_Bit3      : BOOL;
  E_Real1     : REAL;
END_VAR
VAR_OUTPUT
  A_BYTE 1    : BYTE;
  A_BYTE 2    : BYTE;
  A_BYTE 3    : BYTE;
END_VAR
VAR_IN_OUT
  D_BYTE 1    : BYTE;
  D_Bit1      : BOOL;
  D_Bit2      : BOOL;
  D_Bit3      : BOOL;
END_VAR
VAR
  Date1     : DATE;
  Character : STRING[3];
  Bit_field : ARRAY [1..3] OF BYTE;
END_VAR
BEGIN
END_FUNCTION_BLOCK
```

```
ORGANIZATION_BLOCK Cycle
VAR_TEMP
  SInfo   : ARRAY [1..20] OF BYTE;
  LData   : ARRAY [1..16] OF BYTE;
  Temp1   : STRING [36];
  Temp2   : BOOL;
  Temp3   : BOOL;
  Temp4   : BOOL;
  Temp5   : BYTE;
  Temp6   : INT;
END_VAR
BEGIN
//Доступ по абсолютным адресам
  ...
  T    LW 20;
  ...
  =    L 22.2;
//Доступ по символам
  T    Temp6;
  =    Temp3;
  T    LData[16];
//Загрузка адреса переменной
  L    P#Temp1;
  LAR1 P#Temp2;      ...
END_ORGANIZATION_BLOCK
```

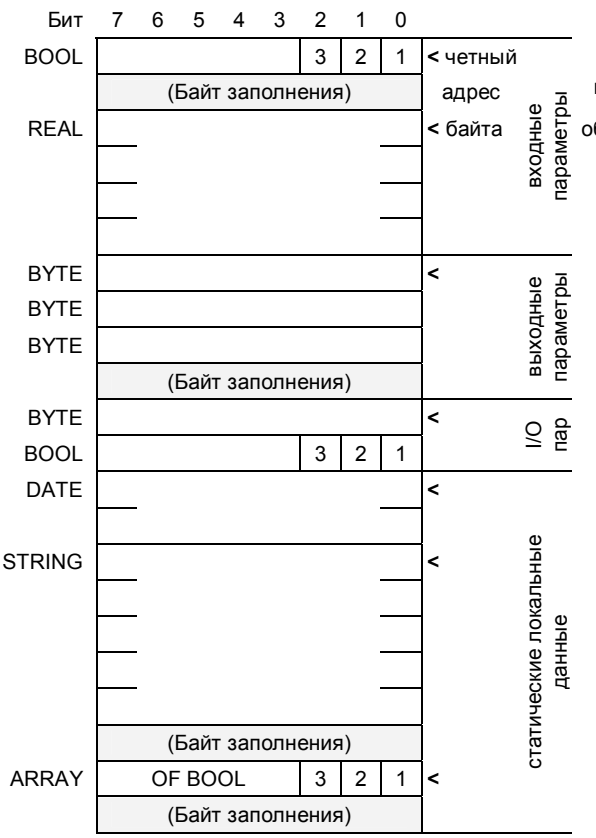


Рис. 26.2 Пример размещения данных в экземплярном DB

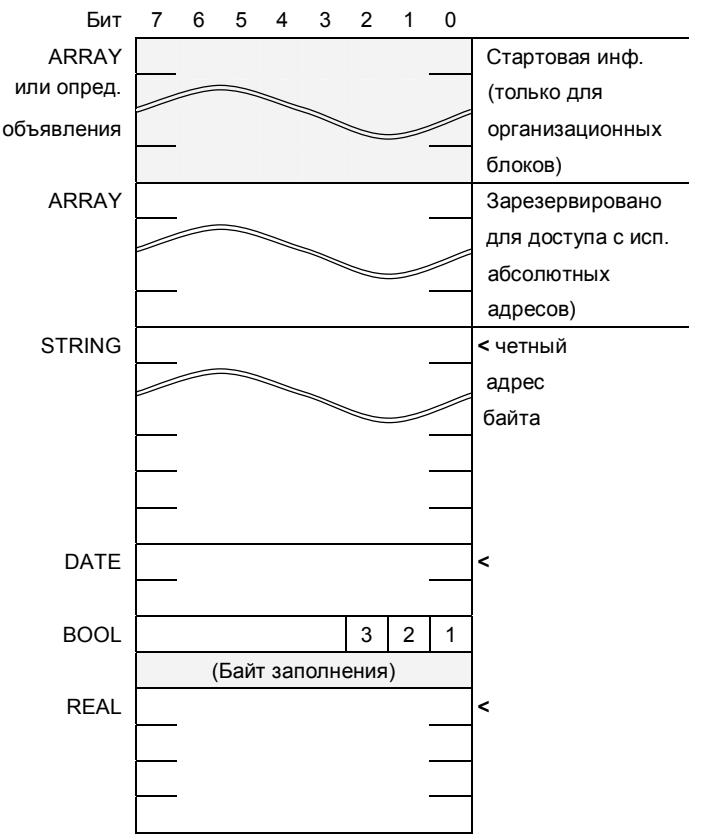


Рис. 26.3 Пример размещения данных L-стека в организационных блоках

На рисунке 26.3 показан пример размещения временных локальных данных в организационном блоке.

Массив "LData" располагается сразу после стартовой информации с байта LB 20 и простирается, например, вплоть до байта LB 35. Редактор не занимает эту область своими собственными временными данными, поэтому Вы можете использовать данную область для абсолютной адресации.

В функциях и функциональных блоках стартовая информация опускается. Если Вам необходимы временные локальные данные для абсолютной адресации, используйте массив как первую переменную в данных блоках; при этом начало массива будет располагаться в байте LB 0.

26.3 Сохранение данных при передаче параметров

Параметры блоков хранятся по-разному в функциях и функциональных блоках. Вам как пользователю нет необходимости вникать в это; Вы программируете параметры для блоков обоих типов одинаково. Тем не менее, эта разница чрезвычайно важна с точки зрения вопроса прямого доступа к параметрам блока.

26.3.1 Доступ к параметрам в функциях

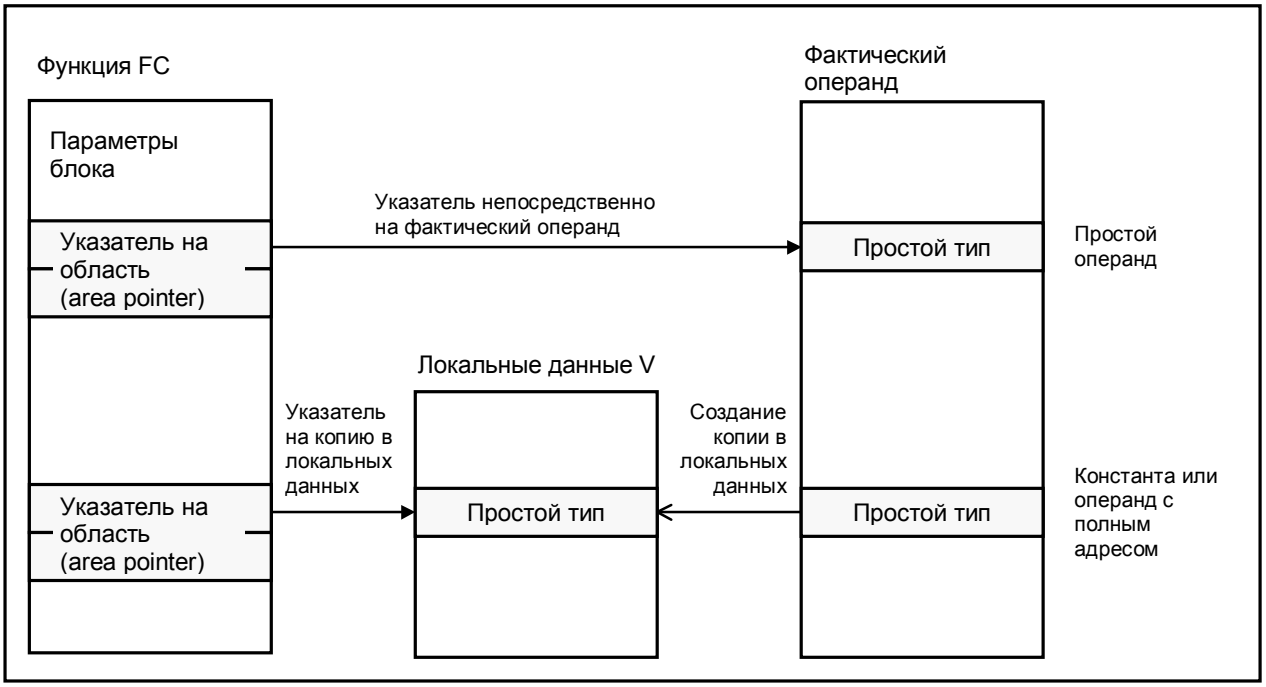
Редактор сохраняет данные о параметрах блоков для функции (имеются в виду параметры функции, относящиеся к типу "параметр блока"; далее по тексту - "параметры функции") в виде межзонных указателей области (area pointer) в коде блока в соответствующем операторе вызова; таким образом, каждый параметр функции требует для хранения одно двойное слово памяти. В зависимости от типа данных и от объявленного типа указатель указывает на собственно фактический параметр, на копию фактического параметра во временных локальных данных вызывающего блока (созданную редактором) или на указатель во временных локальных данных вызывающего блока, который в свою очередь указывает на собственно фактический параметр (см. табл. 26.3). Исключение: для следующих типов параметров: TIMER, COUNTER и BLOCK_xx указателем является 16-разрядное число, расположенное в левом слове параметра блока (функции).

Таблица 26.3 Доступ к параметрам функции

Тип данных	INPUT (входной)	IN_OUT (входной-выходной)	OUTPUT (выходной)
	Параметр является указателем на область (area pointer) для:		
Простой тип	значения	значения	значения
Сложный тип	DB-указателя	DB-указателя	DB-указателя
TIMER, COUNTER и BLOCK	номера	недопустимый	недопустимый
POINTER	DB-указателя	DB-указателя	DB-указателя
ANY	ANY-указателя	ANY-указателя	ANY-указателя

В случае простых типов данных параметр блока указывает непосредственно на фактический параметр (см. 26.4).

Указатель на фактический операнд или его значение



Указатель на другой указатель

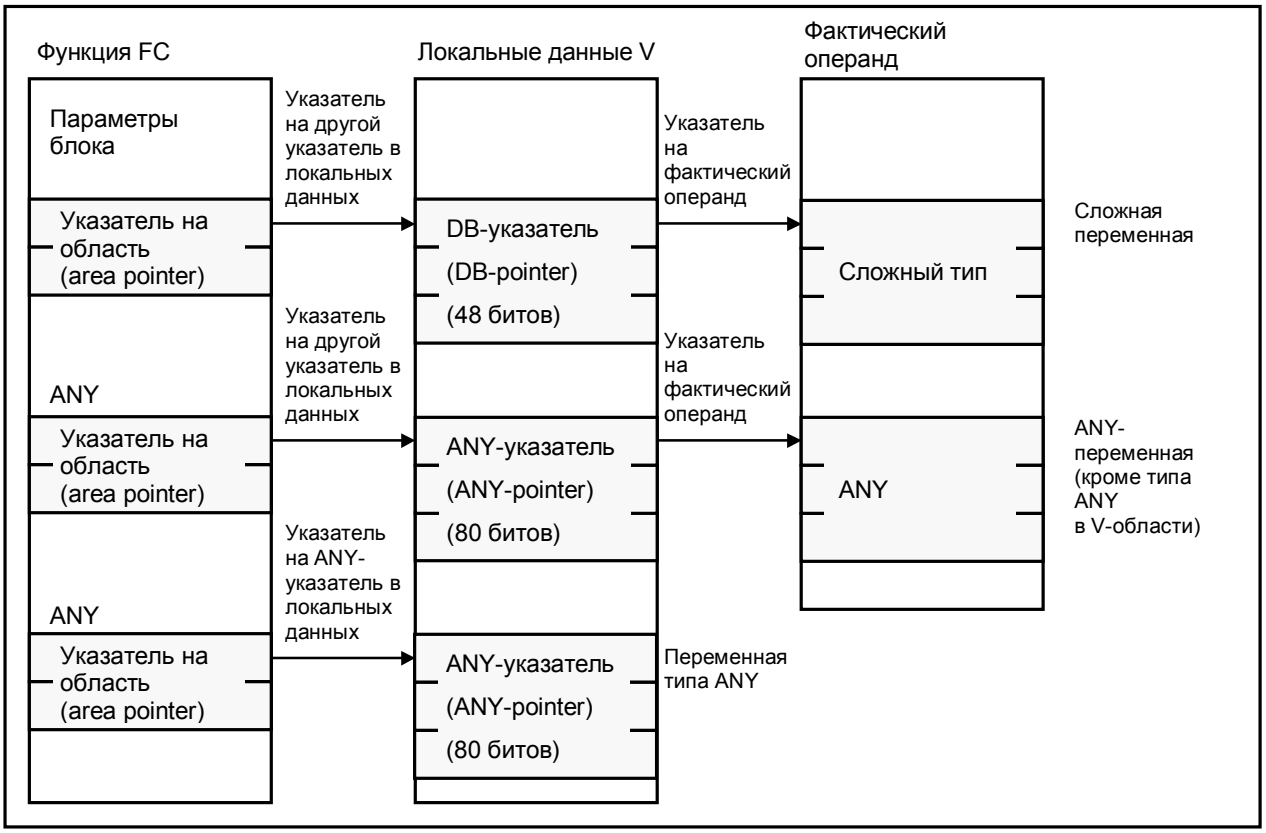


Рис. 26.4 Передача параметров в функции

Тем не менее, с помощью указателя на область (area-pointer) как параметра блока Вы не можете получить доступ ни к каким константам или к операндам, расположенным в блоках данных. По этой причине еще на этапе компилирования редактор копирует константы или фактические параметры, размещенные в блоке данных (операнд с полным адресом), в область временных локальных данных вызывающего блока и "направляет" указатель (area-pointer) на эти копии данных. Подобная область параметров называется "V" (временные локальные данные "предыдущего" блока или V-область).

Перед текущим вызовом функции FC имеет место копирование в V-область данных, полученных при вызове функции: входных параметров и входных-выходных параметров, а после вызова функции имеет место копирование возвращенных входных-выходных параметров, выходных параметров, а также значения функции. Поэтому здесь действует правило, согласно которому Вы можете только проверить входные параметры и сделать запись в выходные параметры. Если Вы передаете значение во входной параметр, используемый в операнде с полным адресом, то это значение будет сохранено во временных локальных данных вызывающего блока и будет в дальнейшем "забыто" (утеряно), так как в дальнейшем не будет никакого "обратного" копирования в "фактическую" ("actual") переменную в блоке данных. Похожая история и с загрузкой соответствующего выходного параметра: так как никакого копирования не происходит из фактической переменной из блока данных в V-область, Вы сможете загрузить (load) лишь "неопределенное" значение из V-области в данном случае.

Из-за операции копирования выходной параметр, как и значение функции, **должен** быть перезаписан определенным значением при простом типе данных в блоке, если операнд с полным адресом рассматривается или может рассматриваться как фактический параметр. Если Вы не назначаете значения для выходного параметра (например, из-за преждевременного выхода из блока или из-за обхода в программе соответствующей инструкции), значение во временных локальных данных также не будет инициализировано. Оно будет иметь значение, которое имело место перед вызовом блока. В дальнейшем выходной параметр будет перезаписан этим "неопределенным" значением.

В случае сложных типов данных, таких как DT, STRING, ARRAY, STRUCT, а также UDT, фактические параметры размещаются или в блоке данных, или в V-области. Так как указатель на область (area-pointer) не может обеспечить доступ к фактическим параметрам в блоке данных, то редактор создает DB-указатель в V-области во время процесса компиляции. Этот DB-указатель указывает на фактический параметр в блоке данных (если номер DB не равен 0), или указывает на V-область (если номер DB равен 0). DB-указатели для всех объявленных типов в V-области создаются до фактического вызова функции FC.

В случае типов параметра, таких как TIMER, COUNTER и BLOCK_xx, параметр блока содержит номер (16-разрядное число, располагающееся с выравниванием влево в 32-разрядном параметре) вместо указателя на область (area-pointer).

В случае, если тип параметра POINTER, то параметр обрабатывается точно также, как данные сложных типов.

В случае, если тип параметра ANY, то редактор создает 10-байтовый ANY-указатель в V-области, который указывает на любую (ANY)

переменную. Такой же подход распространяется на сложные типы данных.

Редактор делает исключение в случае, если Вы используете для параметра блока типа ANY фактический параметр, который размещается в области временных локальных данных и относится к типу данных ANY. В этом случае редактор не создает никаких ANY-указателей, а вместо этого направляет "указатель на область" (area-pointer) (параметр блока) прямо на фактический параметр (в этом случае ANY-указатель может быть модифицирован в процессе выполнения программы, см. раздел 26.3.3 "Переменная" ANY-указатель).

26.3.2 Хранение параметров в функциональных блоках

Редактор сохраняет параметры для функционального блока в его экземплярном блоке данных. При вызове функционального блока редактор создает код, который обеспечивает копирование значений фактических параметров в экземплярный блок данных перед "фактическим" вызовом функционального блока, а затем копирует значения этих параметров назад из экземплярного блока в фактические параметры после вызова блока. При просмотре скомпилированного блока Вы не можете видеть этот код, создаваемый редактором, но Вы можете заметить его присутствие по косвенным признакам - по загрузке памяти.

В экземплярном блоке данных параметры блока хранятся либо как значения (в виде 16-разрядных чисел), либо как указатель, указывающий на фактический параметр (см. табл. 26.4). Если параметр хранится как значение, то требуемая для этого память зависит от типа данных параметра блока. Так, число занимает два байта, указатели занимают 6 байтов (DB-указатели) или 10 байтов (ANY-указатели).

Таблица 26.4 Доступ к параметрам функционального блока

Тип данных	INPUT (входной)	IN_OUT (входной-выходной)	OUTPUT (выходной)
Простой тип	значение	значение	значение
Сложный тип	значение	DB-указатель	значение
TIMER, COUNTER и BLOCK	номер	недопустимый	недопустимый
POINTER	DB-указатель	DB-указатель	недопустимый
ANY	ANY-указатель	ANY-указатель	недопустимый

Связь между параметрами блоков, назначениями экземплярных данных и фактическими параметрами показана на рис. 26.5. При копировании значений фактических параметров сложных типов в экземплярный блок данных (входные параметры) и при копировании значений параметров из экземплярного блока в фактические параметры (выходные параметры) редактор использует системную функцию SFC 20 BLKMOV, параметры которой он встраивает в область временных локальных данных вызывающего блока.

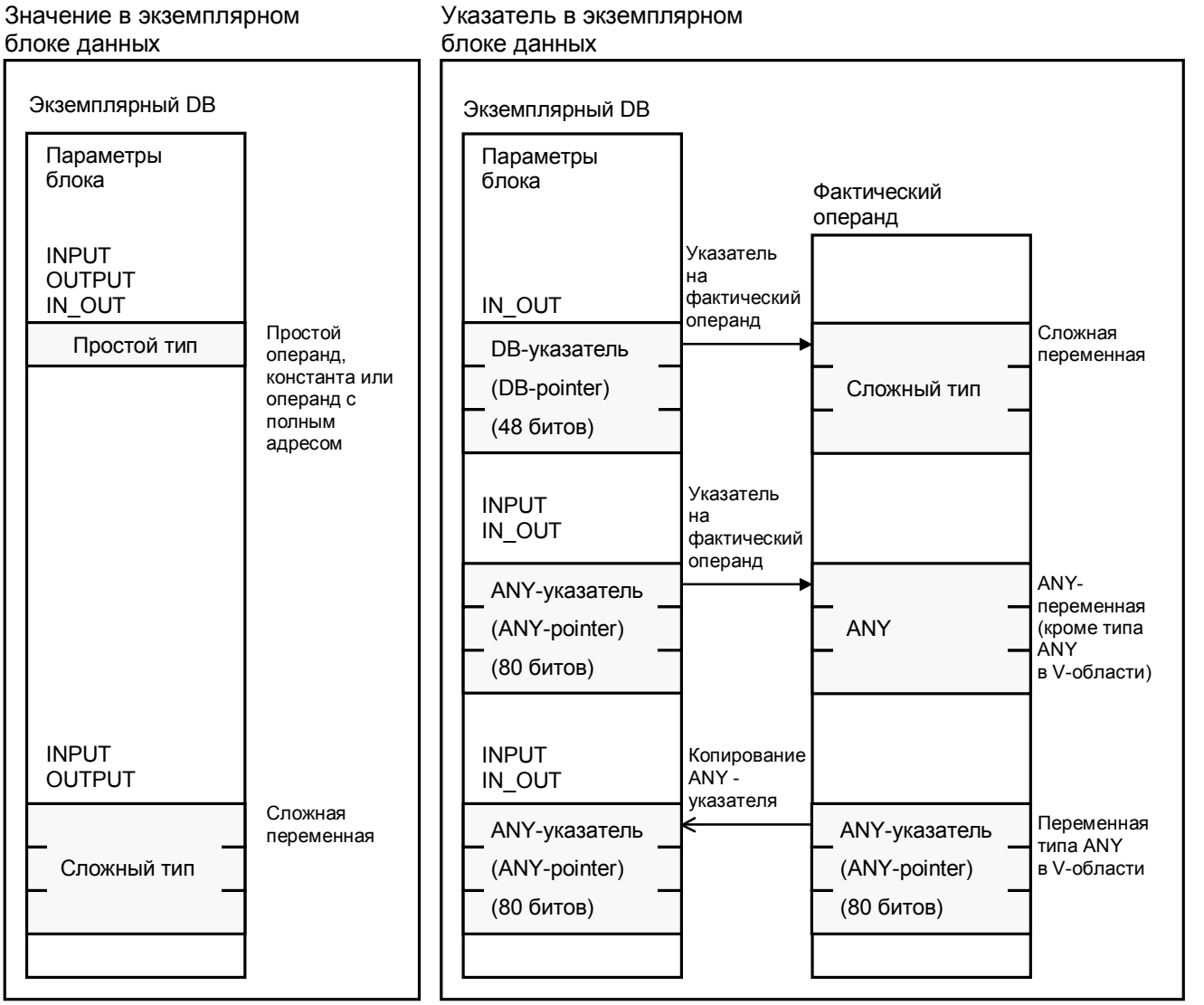


Рис. 26.5 Передача параметров при использовании функциональных блоков

Перед текущим вызовом функционального блока и после такого вызова производится копирование и передача параметров блока в экземплярный блок данных с использованием последовательности операторов кода, вносимого редактором при компиляции. Перед текущим вызовом функционального блока производится копирование входных и входных-выходных параметров блока, а после вызова функционального блока производится копирование входных/выходных и выходных параметров блока. Поэтому здесь действует правило, согласно которому Вы можете только проверить входные параметры и сделать запись в выходные параметры. Если Вы, к примеру, передаете (новое) значение во входной параметр, то текущее значение фактического параметра будет потеряно. Если Вы загружаете (load) выходной параметр, это значит, что Вы загружаете (старое) значение из экземплярного блока данных, а не значение фактического параметра.

Так как параметры блока хранятся в экземплярном блоке данных, то нет

необходимости инициализировать их всякий раз при вызове функционального блока. Если не была проведена инициализация параметров блока, то в программе будут использоваться "старые" значения входных и входных-выходных параметров блока, а что касается выходных параметров блока, то Вы сможете считать их значения, относящиеся к другой точке выполняемой программы. Вне функционального блока Вы можете получить доступ к переменным, которые хранятся в экземплярном блоке данных, таким же способом, как и к переменным в блоке глобальных данных (с использованием символьного имени блока данных и имени параметра блока). Это касается также и статических локальных данных.

Если Вы используете временную локальную переменную типа ANY в ANY-параметре, то редактор копирует содержание этой переменной в ANY-указатель (в параметр блока) в экземплярном блоке данных.

26.3.3 "Переменная" ANY-указатель (ANY-pointer)

ANY-параметры могут быть параметризированы только такими областями данных или переменными, которые уже определены на этапе компиляции.

Пример:

Копирование переменной в область данных с помощью системной функции SFC 20 BLKMOV:

```
CALL SFC 20 (
   SRCBLK  := "ReceiveMailbox".Data,
   DSTBLK  := P#DB63.DBW0.0 BYTE 8,
   RET_VAL := SFC20Error);
```

Во время работы программы можно модифицировать или переопределить переменную или область данных, так как редактор использует фиксированный ANY-указатель на фактический параметр во временных локальных данных (см. далее в данном разделе).

Но есть здесь одно исключение, которое касается случая, когда фактический параметр располагается во временных локальных данных и имеет тип данных ANY. Тогда редактор не создает никакого ANY-указателя, вместо этого он интерпретирует эту ANY-переменную как ANY-указатель на фактический параметр. Это означает, что ANY-переменная должна иметь такую же структуру как ANY-указатель.

Во время работы программы Вы можете модифицировать такие ANY-переменные во временных локальных данных и тем самым специфицировать другие фактические параметры, соответствующие ANY-параметрам.

Для применения таких "переменных" ANY-указателей Вы должны выполнить следующее:

- Применение временной локальной переменной типа ANY (имя ANY-переменной может быть выбрано произвольным в допустимых рамках

для локальных переменных блока):

```
VAR_TEMP
   ANY_POINTER  : ANY;
END_VAR
```

- Инициализация ANY-переменной значениями:

Адрес, известный во время выполнения программы	Адрес, неизвестный во время выполнения программы
	`LAR1 P#ANY_POINTER;`
`L W#16#1002;`	`L W#16#1002;`
`T LW 0;`	`T LW[AR1,P#0.0];`
`L 16;`	`L 16;`
`T LW 2;`	`T LW[AR1,P#2.0];`
`L 63;`	`L 63;`
`T LW 4;`	`T LW[AR1,P#4.0];`
`L P#DBB0.0;`	`L P#DBB0.0;`
`T LD 6;`	`T LD[AR1,P#6.0];`

- Инициализация ANY-параметра, например, в блоке SFC 20:

```
CALL SFC 20 (
     SRCBLK  := "ReceiveMailbox".Data,
     DSTBLK  := P#DB63.DBW0.0 BYTE 8,
     RET_VAL := SFC20Error);
```

Данная процедура не ограничивается только функциональным блоком SFC 20 BLKMOV; Вы можете применять ее в отношение всех ANY-параметров любых блоков.

Пример:

Пусть необходимо создать блок, копирующий одну область данных в другую. Исходная область и область назначения должны быть параметризируемыми. Мы используем функцию SFC 20 BLKMOV для копирования.

Блок - функция FC - имеет следующие параметры:

```
VAR_INPUT
     QDB   : INT; //Исходный блок данных
     SSTA  : INT; //Начальный адрес исходных данных
     NUMB  : INT; //Число байтов
     DDB   : INT; //Блок данных назначения
     DSTA  : INT; //Начальный адрес области назначения
END_VAR
```

Возвращаемое значение функции должно содержать сообщение об ошибках функции SFC 20. Кроме того бит слова состояния BR устанавливается в состояние "0" в случае ошибки.

Для работы с локальными данными блока достаточно использовать две ANY-переменные: одну - как указатель для исходной области, а вторую - как указатель для области назначения:

```
VAR_TEMP
     SANY  : ANY; //ANY-указатель исходной области
     DANY  : ANY; //ANY-указатель области назначения
END_VAR
```

Так как мы знаем адреса ANY-указателей во временных локальных данных, мы можем запрограммировать их, используя абсолютную адресацию, например, при подготовке ANY-указателя области назначения:

```
L    W#16#1002;    // тип BYTE
T    LW 0;
L    NUMB;         // число байтов
T    LW 2;
L    QDB;          // исходный DB
T    LW 4;
L    SSTA;         // начало исходных данных
SLD  3;
OD   DW#16#8400_0000;
T    LD 6;
```

Указатель области назначения с начальным адресом LB 10 программируется аналогичным образом.

Остается только инициализировать блок SFC 20:

```
CALL SFC 20 (
  SRCBLK  := SANY,
  DSTBLK  := DANY,
  RET_VAL := RET_VAL);
```

Значение функции RET_VAL для SFC 20 инициализируется значением RET_VAL нашей функции FC.

Полностью данный маленький пример Вы можете найти на дискете, прилагаемой к данной книге (функция FC 47 в программе "General Examples" ("Общие примеры")).

Таким образом, ANY-указателю может быть назначено любое значение; например, Вы можете менять тип в слове 2 или указатель на область (area pointer); так что в принципе Вы можете адресовать любые переменные и области данных, например, область меркеров.

Примечание:

Если ANY-указатель, размещенный во временных локальных данных, указывает на переменную, которая также размещена во временных локальных данных вызывающего блока, то V-область должна быть

введена как область-операнд ("operand area"), так как с точки зрения вызванного блока данная переменная размещена во временных локальных данных предшествующего ("predecessor") блока.

26.4 Краткое описание примера "Message Frame Example" (Пример фрейма сообщения)

Описываемая далее программа-пример углубит Ваше понимание процесса работы со сложными переменными. Программа состоит из различных блоков, каждый из которых иллюстрирует определенный аспект данной темы. Технологические функции, заложенные в примерах, таких, например, как "Generate_Frame" ("Создание фрейма") и "Checksum" ("Контрольная сумма"), предназначены только для того, чтобы сделать яснее способы решения проблемы, и, где это необходимо, изложены лишь вкратце.

Вы можете найти эту программу на дискете, приложенной к данной книге, под заголовком "Message Frame Example" ("Пример фрейма сообщения"). В книге примеры представлены пояснительными текстами и рисунками.

Программа-пример состоит из следующих разделов:

- Message frame data (UDT 51, UDT 52, DB 61, DB 62, DB 63) (раздел "Фрейм сообщения" иллюстрирует вопросы обработки определяемых пользователем структур данных);
- Clock Check (FC 61)
  (раздел "Контроль времени" иллюстрирует вопросы использования системных и стандартных блоков);
- Checksum (FC 62)
  (раздел "Контрольная сумма" иллюстрирует вопросы использования прямого доступа к переменным);
- Generate frame (FB 51)
  (раздел "Создание фрейма" иллюстрирует вопросы использования системной функции SFC 20 BLKMOV с фиксированными адресами);
- Store frame (FB 52)
  (раздел "Хранение фрейма сообщения" иллюстрирует вопросы использования "переменной" ANY-указатель);
- Date conversion (FC 63)
  (раздел "Преобразование даты" иллюстрирует вопросы обработки переменных сложных типов данных).

Пример "Message frame data"
("Фрейм сообщения")

Пример "Message frame data" ("Фрейм сообщения") показывает, как Вы можете определять часто встречающиеся структуры данных, таких как Ваша собственная структура, то есть, относящаяся к UDT-типу, и как использовать данный тип данных при объявлении переменных и параметров.

Программа-пример организована как база данных для входящих и исходящих сообщений: "почтовый ящик" исходящих сообщений ("send mailbox") со структурой фрейма сообщения, "почтовый ящик" входящих сообщений ("receive mailbox") с такой же структурой и входной ("receive") кольцевой буфер для промежуточного хранения входящих фреймов сообщений (см. рис. 26.6).

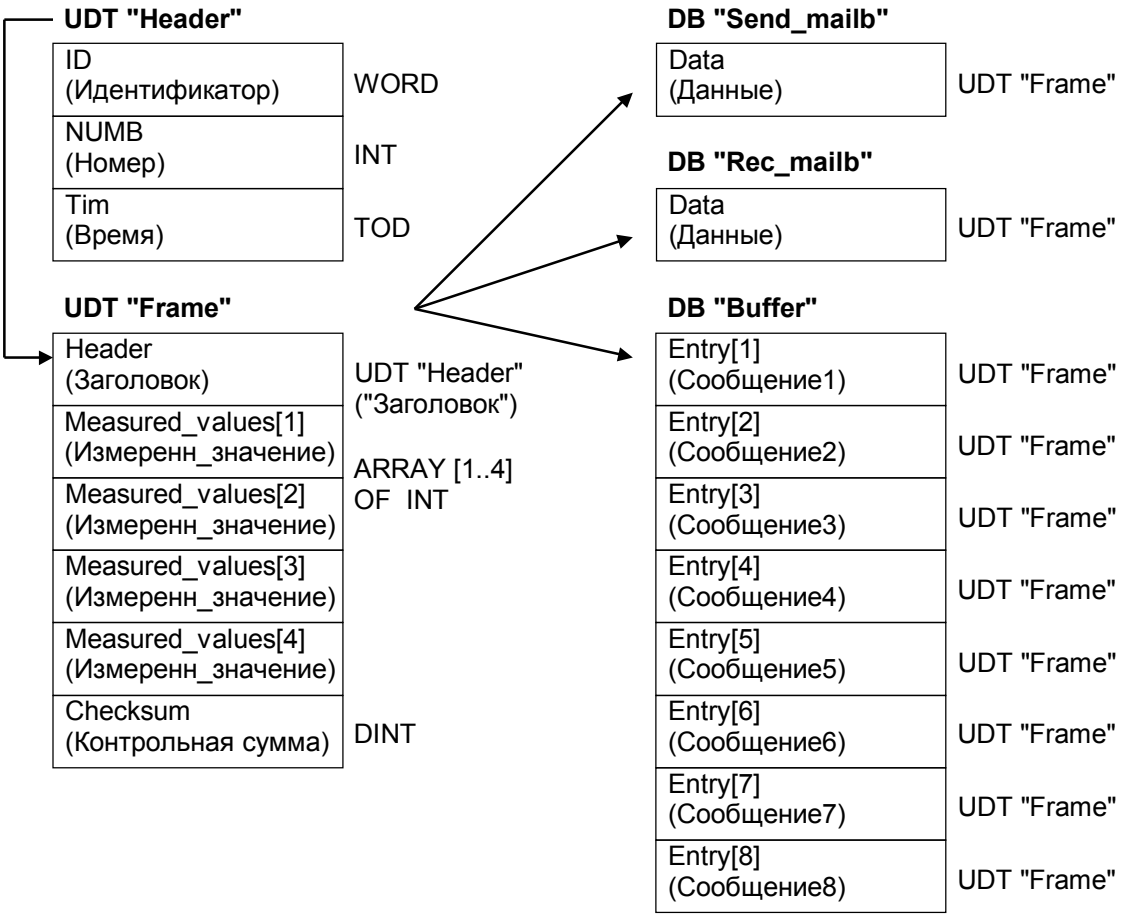


Рис. 26.6 Структура данных для примера "Message frame data" ("Фрейм сообщения")

Так как структура данных сообщения встречается часто, то мы можем определить пользовательский тип данных (UDT) с именем *Frame*. Фрейм содержит заголовок (frame header); мы можем также дать имя структуре заголовка фрейма. "Почтовый ящик" исходящих сообщений ("send mailbox") и "почтовый ящик" входящих сообщений ("receive mailbox") должны представлять собой блоки данных, каждый из которых будет содержать переменные со структурой *Frame.* Наконец, кольцевой буфер для промежуточного хранения входящих фреймов сообщений должен представлять из себя блок данных с массивом из 8 элементов, которые также имеют структуру данных *Frame*.

Сначала мы определяем структуру UDT *Header* , затем - структуру UDT *Frame*.

Структура UDT *Frame* содержит структуру UDT *Header*, массив *Measured_values* из 4 элементов и переменную *Checksum*. Все компоненты при инициализации получают значение 0.

И в блоке данных "Send_mailb", и в блоке данных "Rec_mailb" определена переменная *Data* со структурой *Frame*. Переменные теперь могут быть отдельно инициализированы в разделах инициализации блоков данных.

В нашем примере компонент *ID* в каждом случае получает свое значение, отличающееся от полученного при инициализации пользовательского типа UDT. Блок данных "Buffer" содержит переменную *Entry* в виде массива, состоящего из 8 элементов, каждый из которых имеет структуру *Frame*. В этом случае также отдельные компоненты структуры могут быть инициализированы различными значениями в разделе инициализации (например: Entry[1].Header.Numb := 1).

Данный пример, в свою очередь используемый в последующих примерах, содержит следующие объекты:

UDT 51 - пользовательский тип данных Header,

UDT 52 - пользовательский тип данных Frame,

DB 61 - блок данных "Send_mailb" ("почтовый ящик" исходящих сообщений),

DB 62 - блок данных "Rec_mailb" ("почтовый ящик" входящих сообщений),

DB 63 - блок данных "Buffer".

Пример "Clock Check"
("Контроль времени")

Пример "Контроль времени" показывает, как использовать системные и стандартные блоки (для операций проверки наличия ошибок, для операций копирования из библиотеки, для переименования).

Функция "Clock_check" используется для вывода значения суточного времени, считываемого из встроенных в CPU часов реального времени, в виде значения функции. Для этих целей нам потребуется системная функция SFC 1 READ_CLK, которая считывает дату и время суток из встроенных часов реального времени, в формате даты DATE_AND_TIME или DT. Так как нам нужно только значение времени суток, то нам потребуется еще IEC-функция FC 8 DT_TOD. Данная функция позволяет выбрать значение суточного времени в формате TIME_OF_DAY или TOD из данных формата DT (см. рис. 26.7).

Спецификация времени для часов реального времени хранится в блоке данных "Data66". Данная информация нам еще потребуется для использования в примере "Date conversion" ("Преобразование даты"). Если бы в дальнейшем нам не потребовалась данная информация, мы могли бы также объявить временную локальную переменную вместо переменной *CPU_Tim*.

Проверка на наличие ошибок (Error evaluation)

Системные функции сообщают об ошибках посредством бита

двоичного результата BR и с помощью возвращаемого значения функции RET_VAL. О том что произошла ошибка, свидетельствует значение BR = "0"; также при этом значение функции RET_VAL становится отрицательным - устанавливается бит 15. Стандартные IEC-функции сообщают об ошибках только посредством двоичного результата BR. В примере представлены два способа проверки наличия ошибок. В функции "Clock_check" двоичный результат BR сначала установлен в "1". В случае появления ошибки двоичный результат BR сбрасывается в "0". При этом на выход поступает неверное значение суточного времени. После вызова функции "Clock_check" Вы также можете проверить наличие ошибок, используя проверку бита двоичного результата BR.

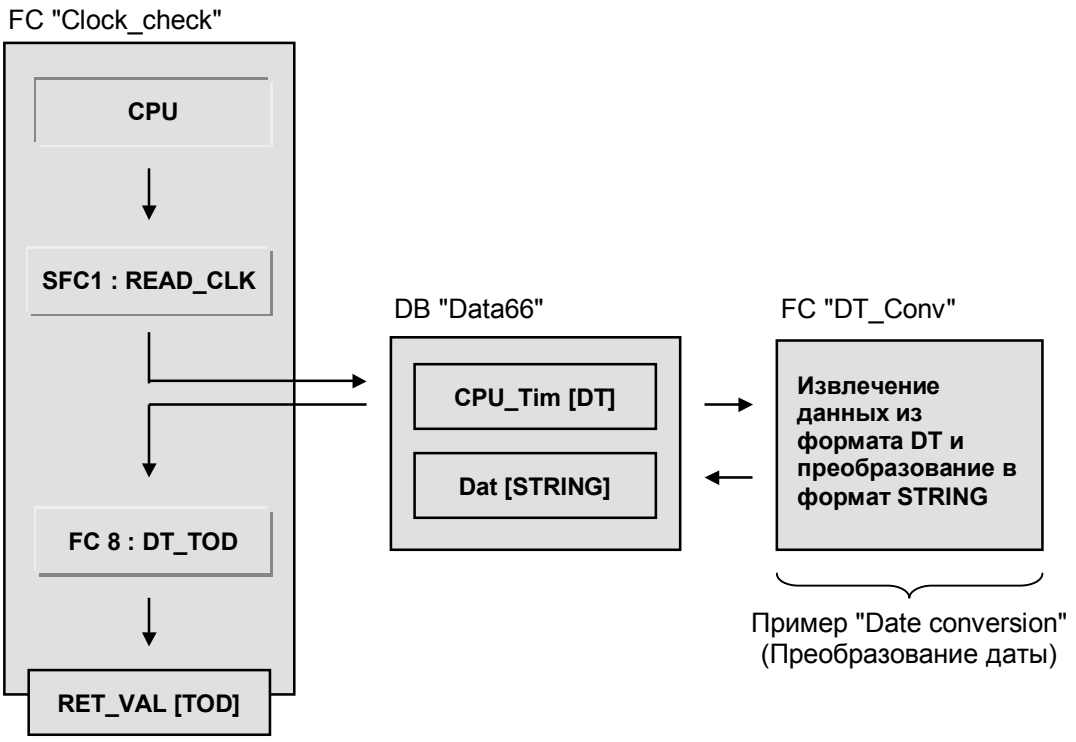


Рис. 26.7 Пример "Clock check" ("Проверка времени")

Программирование системных функций в автономном (offline) режиме

Перед компилированием программы или перед вызовом ее для корректировки в инкрементном режиме "автономная" программа пользователя должна включать в себя системную функцию SFC 1 и стандартную функцию FC 8. Обе функции входят в комплект поставки системы STEP 7. Вы можете найти эти функции в библиотеке блоков из поставляемого программного обеспечения. (Для системных функций, встроенных в CPU, библиотека блоков (block library) содержит описание интерфейса вместо собственно программных кодов этих функций. Эти системные функции могут быть вызваны в автономном (offline) режиме с помощью упомянутого описания интерфейса; описание интерфейса не пересылается в CPU. Загружаемые функции, такие, как IEC-функции хранятся в библиотеке как исполняемые программы.)

Выбрав следующие опции: *File -> Open (Файл -> Открыть)* в Simatic Manager, Вы можете выбрать "стандартную библиотеку" *Standard Library* и открыть библиотеку *System Function Blocks (Системные функциональные блоки)*. В разделе блоки Вы можете найти все описания интерфейсов для системных функций. Если Вы еще не открыли окно Вашего проекта, Вы можете вывести оба окна одно под другим, выбрав следующие опции: *Window -> Arrange -> Vertically (Окно -> Настроить -> Вертикально),* и с помощью манипулятора "мышь" "перетащить" выбранные системные функции в Вашу программу, используя метод "Drag-n-Drop" (Отметьте требуемую функцию SFC с помощью мыши, удерживайте нажатой левую кнопку мыши, "перетащите" захваченный объект на раздел *Blocks* или на его открытое окно и отпустите кнопку). Вы можете скопировать стандартную функцию FC 8 таким же образом. Эту функцию Вы можете найти в библиотеке *IEC Function Blocks (Функциональные блоки стандарта IEC).* Функция FC 8 является загружаемой функцией, следовательно, она занимает часть пользовательской памяти, в отличие от SFC 1.

Если стандартный блок вызывается из каталога программных элементов редактора (Editor's ProgramElement Catalog) из раздела "Libraries" во время инкрементного программирования, то этот блок автоматически копируется в раздел блоков *Blocks* и вводится в таблицу символов.

Переименование стандартных функций

Пользователь имеет возможность переименовывать стандартные функции. Для этого Вы можете выделить стандартную функцию (например, FC 8) в окне проекта и снова щелкнуть кнопкой мыши на имени выбранного объекта. Вокруг имени появится рамка, и в ней Вы можете задать новое имя (например, FC 98). Если теперь Вы нажмете F1, пока выделена стандартная функция (переименованная в FC 98), то Вы получите контекстную справку (функция Help), касающуюся изначальной стандартной функции FC 8.

Если при выполнении операции копирования окажется, что существуют два одноименных блока, то появится диалоговое окно, в котором следует выбрать один из двух вариантов выполнения копирования: копирование с переименованием блока или копирование с перезаписью существующего блока.

Символьная адресация функций

В таблице символов пользователь может назначить символьные имена для системных и стандартных функций, так что эти функции могут быть адресованы в дальнейшем посредством символьных имен. Пользователь свободен в выборе имен в допустимых пределах, регламентированных для имен блоков. В примере символьные имена были заданы для каждого блока (для лучшей идентификации).

Пример "Checksum"
("Контрольная сумма")

Данный пример разъясняет использование прямого доступа к параметру блока типа ANY с вычислением адреса переменной и с использованием косвенной адресации (см. 26.8).

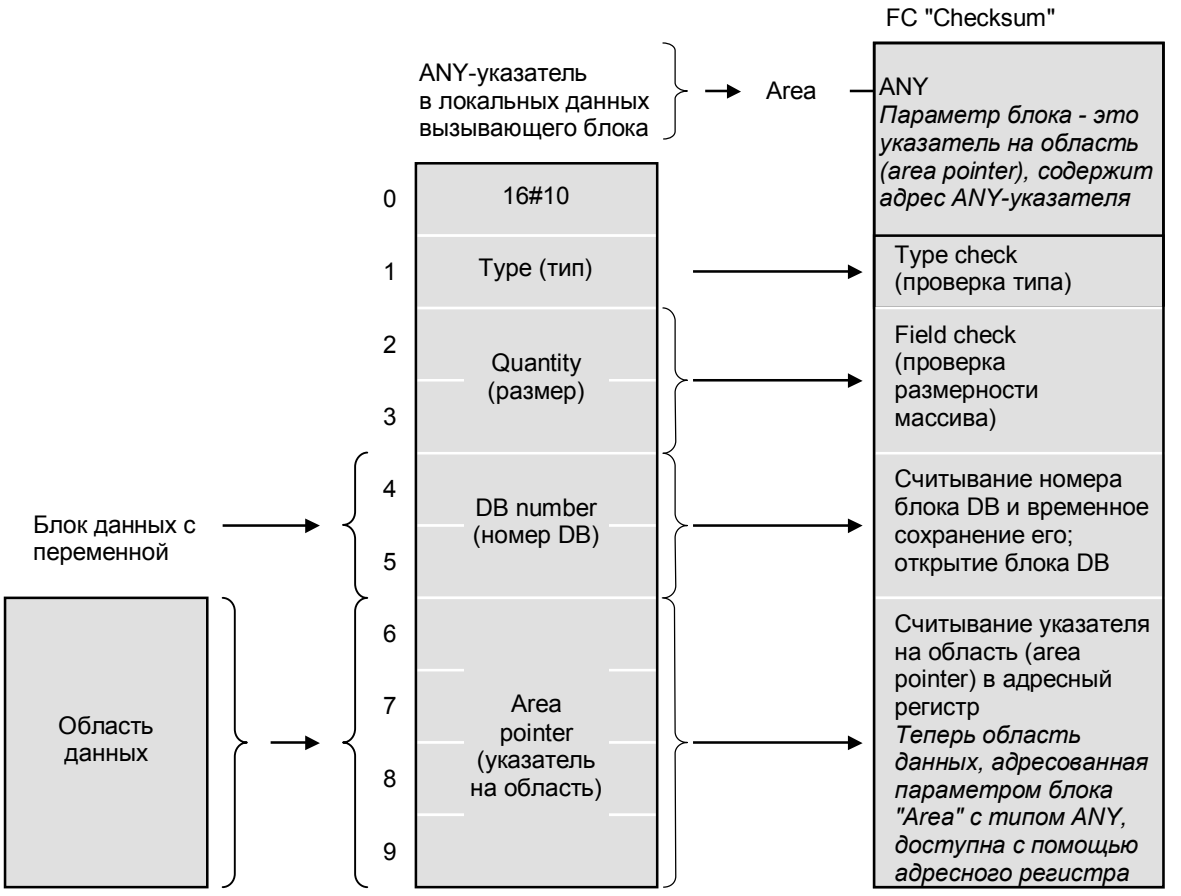


Рис. 26.8 Пример "Checksum" ("Контрольная сумма")

Контрольная сумма должна быть сгенерирована исходя из структуры данных простым сложением всех байтов без учета корректности выполнения (переполнение, нарушение диапазона значений для DINT).

Все структуры данных (STRUCT и UDT) обрабатываются редактором как массивы с байтовыми элементами (размер элемента равен 1 байту), если применяются в параметре блока с типом ANY. Следовательно, с помощью этой программы Вы можете сгенерировать контрольную сумму не только для массивов с байтовыми компонентами (ARRAY OF BYTE), но также и для переменных типа структура (STRUCT). Если Вы хотите использовать программу с переменными другого типа, то Вы должны изменить соответствующую проверку (идентификатор ID типа данных в ANY-указателе).

Функция генерации контрольной суммы использует прямой доступ для получения абсолютного адреса параметра блока (или, более точно - адреса, по которому редактор сохранил ANY-указатель). Сначала выполняется проверка идентификатора ID типа данных (на соответствие типу BYTE) и вводится множитель повторения (>1). В случае ошибки двоичный результат устанавливается в "0" и выполнение функции прерывается с возвращаемым значением функции, равным 0.

Начальный адрес фактического параметра (в режиме выполнения программы) находится в ANY-указателе. Он загружается в адресный

регистр AR1. Если переменная размещена в блоке данных, то этот блок также открывается.

Следующий сегмент добавляет значения всех байтов, составляющих фактический параметр. Циклическое выполнение программы будет продолжаться до тех пор, пока переменная *Quantity* не будет иметь нулевое значение (при каждом проходе цикла значение переменной декрементируется). После этого сумма передается в возвращаемое значение функции.

Пример "Generate frame"
("Создание фрейма")

Пример "Generate frame" ("Создание фрейма") показывает использование системной функции SFC 20 BLKMOV для копирования сложных переменных.

Блок данных "Send_mailb" должен быть заполнен данными фрейма сообщения. Здесь используется функциональный блок с идентификатором ID и порядковым номером в его экземплярном блоке. Собственно данные расположены в глобальном блоке данных; они копируются в блок "почтовый ящик для исходящих сообщений" ("send mailbox") с помощью системной функции BLKMOV.

Значение времени суток поступает от часов реального времени, встроенных в CPU, с помощью функции "Clock_check" (см. выше), затем генерируется контрольная сумма методом простого сложения всех байтов заголовка фрейма сообщения и данных (см. пример "Checksum" ["Контрольная сумма"]). На рис. 26.9 показаны структура программы и структура данных.

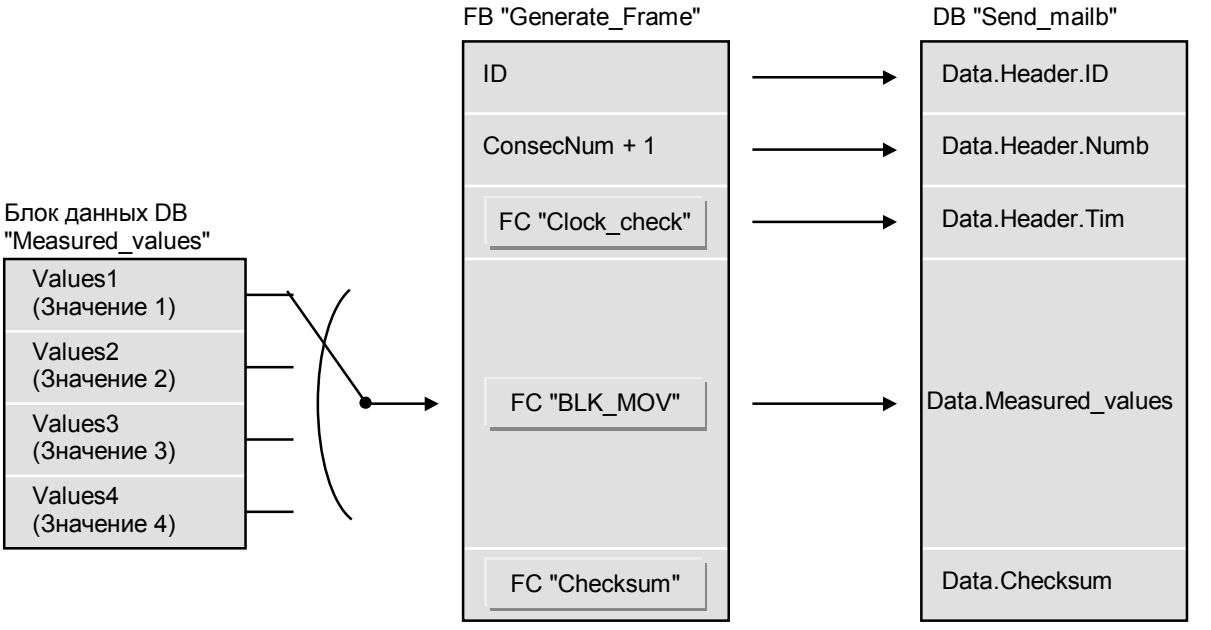


Рис. 26.9 Пример "Generate frame" ("Создание фрейма")

Первый сегмент в функциональном блоке FB "Generate_frame" передает идентификатор ID, сохраненный в экземплярном блоке данных в заголовок фрейма. Порядковый номер ConsecNum инкрементируется (+1) и также поступает в заголовок фрейма.

Второй сегмент содержит вызов функции "Clock_check", которая считывает суточное время из часов реального времени и в формате TIME_OF_DAY передает эту информацию в заголовок фрейма.

В третьем сегменте Вы можете видеть принципы использования системной функции SFC 20 BLKMOV для копирования переменных, выбранных в процессе работы программы, без использования косвенной адресации. Следовательно, нет необходимости знать абсолютный адрес и структуру переменной.

Принцип предельно прост: требуемая функция копирования выбирается с помощью распределителя переходов. Здесь при выборе перехода допускаются номера от 1 до 4. Пример "Buffer entry" ("Входной буфер") демонстрирует такую же функциональность, но на этот раз, с набором "переменных назначения" и с вычисляемым указателем в процессе работы программы (см. 26.10).

В следующем сегменте программы генерируется контрольная сумма с учетом заголовка фрейма и данных фрейма. Так как функция "Checksum" генерирует контрольную сумму по единой области данных, то сначала заголовок фрейма и данные фрейма объединяются во временную переменную *Block*. После этого содержимое переменной *Block* побайтно суммируется, и результат сохраняется как контрольная сумма в исходящем фрейме.

Функциональный блок FB "Generate_Frame" запрограммирован таким образом, что он может вызываться для генерации фрейма по фронту сигнала.

Пример "Store frame"
("Хранение фрейма сообщения")

В примере "Store frame" ("Хранение фрейма сообщения") показано, как используется "переменная" ANY-указатель).

Фрейм в блоке данных "Rec_mailb" должен быть внесен в следующую позицию в блоке данных "Buffer". Локальная переменная блока Entry определяет позицию в кольцевом буфере; адрес кольцевого буфера высчитывается, исходя из значения в данной позиции (см. рис. 26.10).

Если номер фрейма блоке "почтовый ящик для входящих сообщений" ("receive mailbox") изменился, то входящий фрейм должен быть записан в буфере в следующей позиции. Буфер должен представлять собой блок данных, который может накапливать до 8 фреймов. После прихода восьмого фрейма следующий, девятый фрейм, должен быть внесен вновь в первую позицию.

Функциональный блок "Store_Frame" сравнивает номер пришедшего фрейма с сохраненным номером в блоке данных "Rec_mailb". Если номера фреймов различаются, то номер, который сохранен, корректируется, и фрейм, находящийся в блоке "почтовый ящик для входящих сообщений" ("receive mailbox"), копируется в блок данных "Buffer" в следующую позицию. Системная функция SFC 20 BLKMOV управляет копированием.

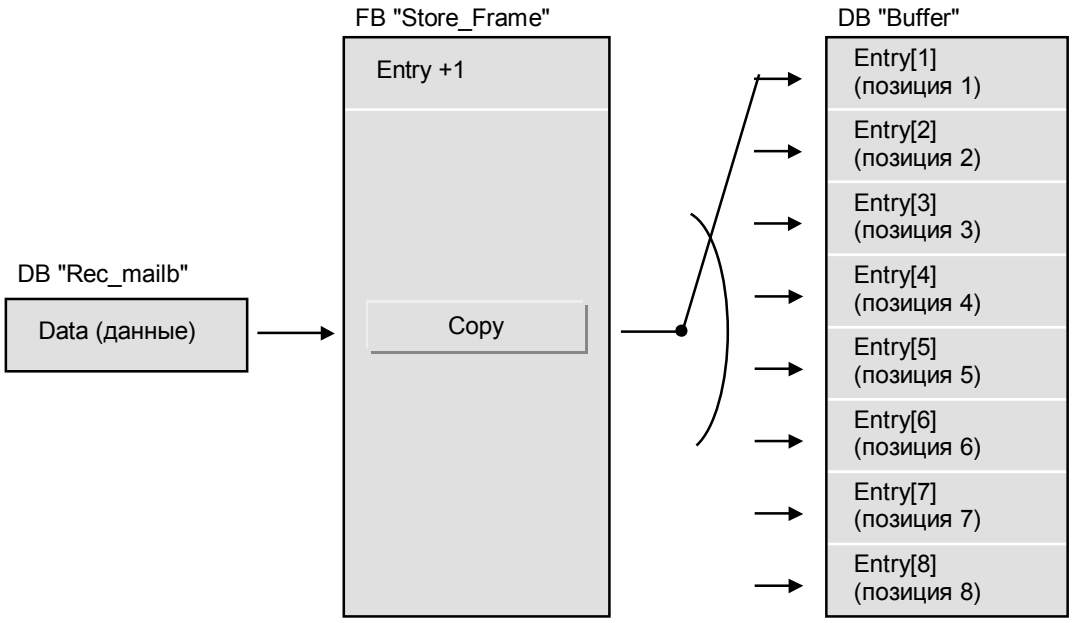


Рис. 26.10 Пример "Store frame" ("Хранение фрейма сообщения")

Область назначения может быть различной, в зависимости от состояния переменной *Entry*. Производится расчет абсолютного адреса области назначения, создается ANY-указатель в переменной *ANY_pointer* и передается в SFC в параметр DSTBLK.

Необходимо отметить, что для косвенной адресации временных локальных переменных Вы можете использовать только внутризонную адресацию.

Структура данных *Frame* имеет размер, равный 20 байтов (заголовок: 8 байтов, собственно данные: 8 байтов и контрольная сумма: 4 байта). Переменная *Receive* в блоке данных "Rec_mailb", следовательно, также имеет длину 20 байтов, точно также как и каждый компонент массива *Entry* в блоке данных "Buffer" имеет длину 20 байтов. Следовательно, отдельные элементы *Entry[n]* начинаются с байтового адреса $n \times 20$, где n соответствует номеру элемента переменной *Entry*.

Пример "Date conversion"
("Преобразование даты")

Пример "Date conversion" ("Преобразование даты") иллюстрирует вопросы обработки переменных сложных типов данных с использованием прямого доступа к переменным и косвенной адресации посредством обоих адресных регистров.

Глобальный блок данных "Data66" содержит переменные *CPU_Tim* (тип данных *DATE_AND_TIME*) и *Dat* (тип данных *STRING*). Дата должна считываться из переменной *CPU_Tim* и сохраняться в виде строки символов в формате "*YYMMTT"* в переменной *Dat*.

Следующая программа в функции "DT_Conv" использует адресный регистр A1 и DB-регистр для указателя на входной параметр Tim.

Адресный регистр A2 и DI-регистр в программе используются для указателя на значение функции (в соответствии с переменной *Dat,* относящейся к типу STRING, в блоке данных "Data66"). Данная программа размещена в функции, так что и DB-регистр, и DI-регистр, а также оба адресных регистра предоставляют пользователю неограниченный доступ.

В первом сегменте программы рассчитывается адрес фактического параметра для параметра блока *Tim,* остающийся действующим ("валидным") во время выполнения программы, после чего этот адрес сохраняется в DB-регистре в AR1. Фактический параметр сложного типа может быть размещен только в блоке данных (глобальных данных или экземплярных данных) или во временных локальных данных вызывающего блока (в V-области). Если фактический параметр находится в блоке данных, то номер этого блока данных будет загружен в DB-регистр, и указатель на область (area pointer) в адресном регистре AR1 будет содержать адресную область блока данных DB. Если фактический параметр находится в V-области, то в DB-регистр будет загружен нуль, и указатель на область (area pointer) в адресном регистре AR1 будет содержать адресную область V.

Во втором сегменте программы содержится "эквивалентная" программа для значения функции, адрес которого размещается в адресном регистре AR2 и в DI-регистре. Чтобы можно было использовать косвенную адресацию также посредством DI-регистра, адресная область DI должна быть введена в адресный регистр AR2. Тем не менее, в зависимости от занимаемой фактическим параметром области памяти здесь определяется или DB для блока данных, или V - для V-области. Установкой 24-го бита в адресном регистре AR2 в состояние "1" мы можем изменить адресную область с DB на DI, но мы никак не сможем изменить адресную область V.

Подготовленное таким путем значение максимальной длины, зафиксированное для фактического параметра в значении функции, может быть использовано в следующем сегменте. Эта длина должна состоять по крайней мере из 6 символов. Если длина короче 6 символов, то в двоичный результат BR записывается значение "0" (в противоположном случае BR = "1"), после чего завершается обработка блока. Таким образом, Вы можете контролировать наличие ошибок обработки с помощью проверки двоичного результата после вызова функции "DT_Conv".

В следующем сегменте программы считываются год и месяц из переменной *Tim* (в формате двоично-десятичного [BCD] числа), после чего эти значения преобразуются в символы ASCII (с предшествующим символом "3") и записываются в качестве возвращаемого значения функции. Так же выполняется обработка данных, которые определяют день.

Обработка программы заканчивается корректировкой длины в значении функции.

Структурированный язык управления SCL

Структурированный язык управления SCL (Structured Control Language) является языком программирования высокого уровня для SIMATIC S7. Язык базируется на стандарте DIN EN 61131-3 (часть "Structured Text" ["Структурированные тексты"]) и имеет сертификат совместимости с PLC Base Level [Базовый уровень] версии V4.01 при использовании интернациональных мнемоник (в данной книге изначально использованы мнемоники, принятые в Германии). Язык SCL оптимизирован для программирования программируемых контроллеров (PLC). SCL содержит в себе элементы языка Паскаль (Paskal) наряду с типичными для PLC элементами, такими, например, как "вход" ("input") и "выход" ("output").

SCL особенно подходит для программирования сложных алгоритмов или для задач, относящихся к области управления данными. Язык SCL поддерживает характерную для STEP 7 блочную структуру, а также позволяет создавать S7-программы, включающие в себя фрагменты на базовых языках программирования STL, LAD и FBD.

Программное обеспечение S7-SCL является опционным (то есть, поставляемым по отдельному заказу) программным продуктом. ПО S7-SCL может быть поставлено вместе с базовым пакетом STEP 7 Basic Package. Описание S7-SCL в данной книге базируется на версии языка SCL V.5.1

При инсталляции ПО S7-SCL данный язык программирования полностью интегрируется с утилитой SIMATIC Manager и может после этого использоваться наряду с базовыми языками программирования (например, STL). Используя редактор SCL-программ, пользователь может создавать в S7-проекте исходные программы, которые он должен затем скомпилировать с помощью SCL-компилятора. Программа пользователя содержит скомпилированные SCL-блоки; эта программа, кроме того, может содержать скомпилированные блоки, написанные на других языках программирования. Пользователю также предоставляется возможность протестировать блоки, созданные с использованием языка SCL, в интерактивном (online) режиме в CPU с использованием отладчика SCL Debugger.

Элементы языка SCL в синтаксисе инструкций отличаются от элементов других (базовых) языков программирования (имеются в виду операторы, выражения, присвоение значений). Однако, все они совместно используют типы данных, адресные области, символьные имена и блочную структуру.

Используя **управляющие операторы (control statements)**, пользователь может организовывать ветвление программы, выполнять программные циклы. С помощью использования операций переходов можно прерывать

последовательное выполнение программы, а затем продолжать ее выполнение, начиная с другой точки блока.

С использованием SCL пользователь может запрограммировать **блоки** и затем - организовать их вызов в программе (как говорится, вставить блоки в программу), что может быть выполнено как с использованием языка SCL, так и с использованием другого языка программирования для S7. Используя язык SCL, Вы можете получить доступ к любой системной функции.

Стандартные функции, такие как функции преобразования, доступны в виде SCL-функций; кроме того, пользователь, используя языки SCL или STL, может запрограммировать свои собственные функции. И, наконец, для обработки переменных сложных типов пользователь в своей программе может использовать IEC-функции из базового пакета программного обеспечения STEP 7 Basic Package.

27 Введение, элементы языка

Интеграция с системой SIMATIC; адресация; операторы; выражения; присвоение значений.

28 Операторы управления

IF, CASE, FOR, WHILE, REPEAT, CONTINUE, EXIT, GOTO, RETURN.

29 SCL-блоки

Вызовы блоков; передача параметров; переменная OK; механизм EN/ENO.

30 SCL-функции

Функции таймеров; функции счетчиков; функции преобразования; математические функции; функции сдвига и циклического сдвига; программирование собственных функций пользователя с использованием языков программирования SCL и STL.

31 IEC-функции

Функции преобразования; функции сравнения; STRING-функции (функции для работы с переменными типа STRING); функции обработки даты и времени; функции для работы с числами.

27 Введение. Элементы языка

В данной главе рассматриваются требования, которые должны выполняться при программировании на языке SCL. В главе 2 "Программное обеспечение STEP 7" и в главе 3 "SIMATIC S7-программа" содержится детальное описание данного вопроса, поэтому Вы найдете здесь ссылки на эти главы.

В главе 2 "Программное обеспечение STEP 7" предоставлено введение в вопросы, касающиеся средств программирования, таких как, редактор символов (symbol editor), редактор SCL-программ (SCL program editor), компилятор и отладчик. В главе также говорится о программно-аппаратной среде программирования на языке SCL.

В главе 3 "SIMATIC S7-программа" представлена структура программы пользователя. В главе описываются различные варианты выполнения программы, структура блоков, а также перечислены все требуемые для программирования блоков ключевые слова. Вы найдете там также введение в темы "адресация переменных" и "типы данных, поддерживаемые STEP 7".

Примеры, рассматриваемые в данной главе, Вы можете найти на прилагаемой дискете в библиотеке STL_Book library в разделе "27 Language Elements" ("Элементы языка").

27.1 Интеграция с SIMATIC

27.1.1 Инсталляция (установка)

Инсталляция средств программирования на языке SCL требует наличия утилиты SIMATIC Manager соответствующей версии. SCL устанавливается посредством программы установки SETUP; при инсталляции всех поддерживаемых языков и примеров программирования для установки требуется около 8 Мб на жестком диске. Кроме того, Вам потребуется также выполнить авторизацию SCL (подтвердить права на использование), для чего служит специальная дискета.

27.1.2 Создание проекта

Утилита SIMATIC Manager является ключевым средством также и для работы с языком программирования SCL. Для того, чтобы создать программу на SCL, пользователь должен сначала запустить SIMATIC Manager и создать проект также, как и при создании программ на одном из стандартных языков программирования (см. раздел 2.1 "Базовый пакет STEP 7 (STEP 7 Basic Package)"). При этом пользователь может создать проект "вручную" или воспользоваться для его создания специальной программой Wizard -"мастером создания проектов".

При конфигурировании станции достаточно назначить CPU, чтобы SIMATIC Manager создал разделы для связанной с проектом S7-программы. Можно также создать S7-программу непосредственно в проекте и назначить CPU позднее.

Кроме того, можно также использовать уже готовый проект. Необходимо, чтобы были созданы разделы: *S7 Program [S7 программа], Blocks [Блоки],* а также *Symbols [Символы]* для таблицы символов. Если какой-либо объект не существует, необходимо создать его. Для этого нужно выделить раздел более низкого уровня и выбрать опцию меню *Insert (Вставка)*.

Если используется уже существующий проект, то в этом случае в нем уже могут присутствовать исходные программы на STL или скомпилированные блоки, созданные, скажем, с помощью FBD. Это не нарушает работы SCL-редактора. Пользователь может даже вызывать в своей SCL-программе ранее созданные и скомпилированные блоки, независимо от языка, который использовался для их создания.

27.1.3 Редактирование SCL-программы

Выделите раздел *Sources [Исходные программы]* и затем выберите опции меню: *Insert -> SCL Source (Вставка -> Исходная программа SCL)* (такая опция меню будет доступна для использования, только в том случае, если Вы установили SCL в своей системе). Теперь Вы можете переименовать вставленный объект *SCL Source(1).* Дважды щелкнув кнопкой мыши на этом объекте, вызовите редактор SCL-программ, который отобразит при открытии на экране монитора "пустой" исходный SCL-файл. Теперь Вы можете вводить SCL-программу.

Как использовать для ввода программ SCL-редактор, описывается в разделе 2.5.4 "Редактор SCL-программ (SCL-Program Editor)". Написание программы начинается с ввода (редактирования) блока. Структура блока и необходимый набор ключевых слов описываются в разделе 3.5 "Программирование кодовых блоков на SCL".

Здесь для начала мы рассмотрим простой пример: мы запрограммируем функцию ограничителя "Delimiter", которая должна ограничивать входные величины, адаптируя их к заданному диапазону значений (между верхним и нижним пределами); кроме того, в примере мы запрограммируем вызов этой функции в организационном блоке (см. рис. 27.1).

```
FUNCTION Delimiter : INT
VAR_INPUT
   MAX : INT;                              //максимальное значение
   IN  : INT;                              //входное значение
   MIN : INT;                              //минимальное значение
END_VAR
BEGIN
IF IN > MAX THEN Delimiter := MAX;      //адаптация к верхнему значению
   ELSIF IN < MIN THEN Delimiter := MIN;//адаптация к нижнему значению
   ELSE Delimiter := IN;                //допустимое входное значение
END_IF;
END_FUNCTION

ORGANIZATION_BLOCK Mainjprogram
VAR_TEMP
   SINFO : ARRAY [1..20] OF BYTE;
END_VAR
BEGIN
Result := Delimiter (MAX := Maximum, IN := INPUT_VALUE, MIN := Minimum);
END_ORGANIZATION_BLOCK
```

Рис. 27.1 Пример программы функции ограничителя "Delimiter"

Пример программы начинается с объявления типа блока ограничителя "Delimiter" (функция FC) и типа для значения функции (INT). Далее следует описание параметров блока: параметры MAX (максимальное значение, верхняя граница), IN (величина входного сигнала) и MAX (максимальное значение, нижняя граница) объявляются входными параметрами (INPUT), относящимися к типу данных INT. Сама программа следует за разделом объявления переменных. В соответствии с программой, если входная величина IN больше, чем заданная максимальная величина, то функция принимает значение, равное максимальной величине. Если это не так, и входная величина IN меньше, чем заданная минимальная величина, то функция принимает значение, равное минимальной величине. Если оба рассмотренных случая не дали положительного результата, то функция принимает значение, равное входной величине IN.

Теперь мы запрограммируем вызов этой функции в организационном блоке "Main Program". Программируя на языке SCL, Вы должны также зарезервировать 20 байтов в области временных локальных данных для стартовой информации организационного блока, даже если Вы не планируете их использовать.

В отличие от стандартных языков программирования, в языке SCL функция FC с возвращаемым значением функции является

"реальной" функцией, которая вставляется в выражение вместо адреса (идентификатора), и при этом обеспечивается совместимость типов данных. При вызове в организационном блоке "Main Program" функции ограничителя "Delimiter" ее значение присваивается глобальной переменной "Result" ("Результат"); теперь эта переменная будет содержать в себе значение "Input_value", ограниченное предельными значениями "Maximum" и "Minimum".

Исходная SCL-программа может содержать, как минимум, один блок. Вы можете также создать несколько исходных SCL-программ, которые затем должны скомпилировать в определенном порядке с использованием управляющего файла компилятора.

После написания исходной SCL-программы сохраните ее, используя опции меню: *File -> Save (Файл -> Сохранить)*. Так как в примере мы использовали символьную адресацию, то мы должны заполнить таблицу символов (Symbol Table) перед компилированием исходной программы.

27.1.4 Заполнение таблицы символов (Symbol Table)

Заполнение таблицы символов (Symbol Table) в SCL производится так же как и в стандартных языках программирования (см. раздел 2.5.2 "Таблица символов (Symbol Table)"). Вы также можете вводить информацию в уже существующую и частично заполненную таблицу символов. При этом в любой данной S7-программе может присутствовать только одна единственная таблица символов. Таблица символов в разделе *S7 Program [S7 программа]* представляется объектом *Symbols [Символы].*

Вы можете вызвать редактор символов (symbol editor), выбрав опции меню: *Options -> Symbol Table (Опции -> Таблица символов)* в редакторе SCL-программ или дважды щелкнув кнопкой манипулятора "мышь" на объекте *Symbols [Символы]* в окне утилиты SIMATIC Manager. После того, как откроется таблица (пустая, новая или ранее частично заполненная), Вы можете ввести символьные имена Ваших параметров (см. табл. 27.1). После заполнения таблицы, ее необходимо сохранить.

Таблица 27.1 Таблица символов (Symbol Table) для примера функции ограничителя "Delimiter"

Symbol (символ)	Address (адрес)	Data Type (тип данных)	Comment (комментарий)
Main program	OB1	OB1	Исполняемый блок циклически выполняемой программы
Delimiter	FC271	FC271	Функция для ограничения переменной типа INT
Input_value	MW10	INT	Величина входного сигнала
Maximum	MW12	INT	Верхний предел
Minimum	MW14	INT	Нижний предел
Result	MW16	INT	Сигнал на выходе из функции ограничения

27.1.5 Компилирование SCL-программы

Для того, чтобы выполнить компилирование исходной SCL-программы откройте раздел *Sources [Исходные программы]* (если он еще не открыт). Компилятор Вы можете найти, используя опции меню: *Options -> Customize* (*Опции -> Установки пользователя*) на вкладке "Compiler" ["Компилятор"]. (При необходимости создания блоков выберите опцию "Blocks generated" ("Создание блоков")).

Компилирование исходной SCL-программы может быть выполнено с помощью опций: *File -> Compile (Файл -> Компиляция)*; скомпилированные блоки сохраняются в разделе *Blocks [Блоки].* Более подробная информация о компиляции блоков представлена в разделе 2.5.4 "Редактор SCL-программ (SCL-Program Editor)".

Вы можете организовать пакетный режим компиляции сразу нескольких исходных программ; использование специального управляющего файла позволяет проводить компиляцию этих исходных файлов в любом порядке.

Необходимо отметить, что вызываемые блоки или функции должны быть доступны в момент компилирования или как уже скомпилированные блоки, хранящиеся в разделе *Blocks [Блоки]*, или как (заведомо не имеющие ошибок) исходные программы, хранящиеся в разделе *Sources [Исходные программы],* или как стандартные функции, хранящиеся в библиотеке стандартов Standard Library.

27.1.6 Загрузка SCL-блоков

Если программатор (программирующее устройство PLC) подключен к CPU, то с помощью опций меню: *PLC -> Load (PLC -> Загрузка)* Вы можете загрузить скомпилированные блоки в пользовательскую память CPU. Сам CPU должен быть при этом в режиме STOP (СТОП), потому что порядок следования загружаемых блоков может отличаться от порядка следования вызовов блоков.

Более подробная информация об этом и других аспектах, на которые Вы должны обратить Ваше внимание, представлена в разделе 2.6 "Интерактивный режим (Online mode)".

Вы можете также обрабатывать блоки в "интерактивном" ("online") и в "автономном" ("offline") окнах утилиты SIMATIC Manager.

27.1.7 Тестирование SCL-блоков

SCL-отладчик позволяет проводить тестирование отдельных блоков с помощью функции "Program Status" ("Состояние программы") в режиме последовательного мониторинга выбранных регистров и переменных или в пошаговом режиме. С помощью функции "Program Status" ("Состояние программы") Вы можете наблюдать, как меняются значения переменных в

процессе выполнения программы. В пошаговом режиме Вы можете останавливать выполнение программы в точке прерывания и выполнять программу оператор за оператором, отслеживая состояние переменных (см. раздел 2.7 "Тестирование программы").

Таблица переменных (VAT) также может использоваться для тестирования SCL-программы. С помощью этой таблицы Вы можете устанавливать значения переменных и затем наблюдать при выполнении программы результаты таких назначений.

27.1.8 Адреса и типы данных

Адресные области

Адреса и переменные, применяемые при программировании на языке SCL, соответствуют адресам и переменным, применяемым при написании программ на стандартных языках программирования (см. раздел 1.5 "Адресные области"):

- входы I, выходы Q, меркеры M;
- периферийные входы PI;
- периферийные выходы PQ;
- адреса глобальных данных D;
- временные и статические локальные данные (только символьная адресация);
- организационные блоки OB, функциональные FB, функции FC как с возвращаемым значением, так и без него; блоки данных DB.

Функции таймеров T и функции счетчиков C обрабатываются в SCL-программах как "стандартные функции" (см. раздел 30.1 "Функции таймеров T" и раздел 30.2 "Функции счетчиков C").

Примечание:

Адреса глобальных данных имеют отличающиеся адресные идентификаторы по сравнению со стандартными языками программирования. Более подробное изложение вопроса, посвященного адресным идентификаторам в SCL, представлено в разделе 27.2.1 "Абсолютная адресация".

В SCL функции, которые возвращают "значение функции", могут использоваться в выражениях как адреса.

Типы данных

Назначение типа данных определяет:

- тип и значение (компонентов) данных (например, integer [целая], character string [строка символов]);
- разрешенные диапазоны (например, числовой диапазон, длина строки символов);
- разрешенные операции, допустимые для обработки данных определенного типа;

- способ написания констант.

Типы данных, применяемые при программировании на языке SCL, такие же как те, что применяются при написании программ на стандартных языках программирования (в разделе 3.7 "Переменные и константы" представлен соответствующий обзор в табличной форме, а в главе 24 "Типы данных" Вы найдете детальное описание вопроса).

Численные значения могут быть представлены как десятичные числа, как шестнадцатеричные числа, как восьмеричные числа (8#17 соответствует 16#F или 15dec) и как двоичные числа.

Классы типов данных

В связи с возможностью группирования значений, в SCL определяют классы типов данных, которые представляют одинаковое поведение внутри одного класса:

- класс ANY_INT включает в себя данные типов INT и DINT;
- класс ANY_NUM включает в себя данные типов INT, DINT и REAL;
- класс ANY_BIT включает в себя данные типов BOOL, BYTE, WORD и DWORD.

Указанные классы типов данных были введены, чтобы сделать яснее описание операторов; переменные не могут быть описаны с помощью данных классов типов данных.

Запись констант

Константы - это фиксированные значения, которые в общем случае не изменяются при выполнении программы. Константы используются для предопределения начальных значений переменных при описании последних или для объединения (комбинирования) их в программе с другими переменными (например, при применении в качестве граничных значений).

В языке SCL константа не определяет "свой" тип данных, пока она не будет обработана в арифметической операции. Например, константа 1234 может относиться к типу данных INT или к типу данных REAL, в зависимости от применения:

```
int1    := int2 + 1234;   //константа INT
real1   := real2 + 1234;  //константа REAL
```

В языке SCL Вы можете назначать тип данных для константы (так называемая запись константы со спецификацией типа - "type-defined"). Используя соответствующий префикс, Вы можете, например, предопределить переменную WORD в разделе объявлений с помощью десятичного, шестнадцатеричного, восьмеричного или двоичного числа. Ниже представлен пример, в котором переменная, имеющая в каждом из случаев одинаковое значение, имеет различное представление:

```
W1   :  WORD  :=  W#1234 ;                  //десятичное
W2   :  WORD  :=  W#16#04D2 ;               //шестнадцатеричное
W3   :  WORD  :=  W#8#2322 ;                //восьмеричное
W4   :  WORD  :=  W#2#0000_0100_1101_0010; //двоичное
```

Тип данных при абсолютной адресации

Абсолютный адрес всегда принадлежит к классу типов данных ANY_BIT (например, двойное слово меркеров MD10 имеет тип данных DWORD). Операнд может иметь тип данных (отличный от ANY_BIT) только в случае, если имеет символьное имя ("когда он обращен к переменной"), или после преобразования типа данных.

```
MW14    := SHL(IN := MW12, N := 2);
real1   := real2 + DWORD_TO_REAL(MD10);
```

Тип данных STRING

Строка символов должна вводиться в одинарных кавычках. С данным типом могут также использоваться непечатаемые управляющие символы; они должны вводиться в формате $hh (здесь hh означает значение ASCII символа в шестнадцатеричной форме).

```
string1  := '$0A$0D';   //новая строка
```

Для продолжения строки символов на следующей строке или для вставки комментария в разрыв строки символов (если не нужно отображать этот комментарий при печати и при выводе на дисплей) предназначены две специальные комбинации символов '$>' и '<$':

```
string2  := 'ABCDEFGHIJKLMNOP$>'//продолжение следует
            '<$QRSTUVWXYZ';
```

27.1.9 Виды типа данных (Data Type Veiws)

В SCL Вы можете назначать дополнительные типы данных для уже объявленных переменных, или, более точно, Вы можете назначить дополнительные виды типа данных (data type veiws). При этом становится возможным обращаться к содержимому переменной в целом и по частям как к данным, относящимся к разным типам.

Пример:

Пусть, Вы объявляете входной параметр с именем *Station* и типом STRING. Вы должны переслать данную переменную *Station* в вызываемый блок для того, чтобы выполнить обработку, например, путем добавления численного значения. Кроме того, Вам необходимо высчитать текущую длину *Station*. Для этой цели Вы применяете дополнительный вид типа данных, например, в форме структуры из двух байтов. Первый байт этой структуры содержит максимальную длину строки, второй байт этой структуры содержит текущую длину строки. Дополнительный вид типа данных принимает имя *Len,* а компоненты структуры соответственно называются *max* и *cur.*

```
VAR_INPUT
   Station : STRING24 := ' ';
```

```
Len AT Station : STRUCT
   max : BYTE;     //Максимальная длина
   cur : BYTE;     //Текущая длина
   END_STRUCT;
END_VAR
...
IF WORD_TO_INT(Len.cur) > 12
       THEN ...
END_IF;
...
```

В примере сначала Вы объявляете переменную с "исходным" типом данных и с любым предопределением ее значения. Затем Вы назначаете дополнительный вид типа данных (data type veiws) с ключевым словом AT:

```
View AT Variable : Data_type;   //комментарий
```

Вы можете применить несколько видов типа данных (data type veiws) для переменной, давая при этом им различные имена. При этом предопределение их фиксированными значениями (инициализация) не допускается.

Требуемый объем памяти для вида типа данных (data type veiw) не должен превышать объема памяти для самой переменной, для которой этот вида типа данных назначается (новый вид типа данных должен соответствовать переменной).

Вы можете использовать виды типа данных (data type veiws) для переменной как любые другие переменные, но только локально в блоке. В вышеприведенном примере в вызывающем блоке входной параметр *Station* инициализируется строкой символов: вид типа данных недоступен ему как байтовая структура.

Вид типа данных (data type veiw) может применяться посредством параметров блока и временных и статических локальных данных. Вид типа данных (data type veiw) должен быть объявлен в том же самом разделе объявления, где объявляются переменные.

В таблице 27.2 показано, какие виды типа данных (data type veiws) Вы можете применять с переменными разных типов данных. Если, к примеру, переменная размещается в области временных локальных данных функции FC, и если она относится к сложному типу данных, то виды типа данных (data type veiws), которые могут применяться с такой переменной могут относиться к одному из следующих типов данных: простой тип, сложный тип, тип данных POINTER или ANY.

Переменные типов TIMER, COUNTER или BLOCK_xx не допускают применения к себе видов типа данных (data type veiws).

В представленной ниже таблице 27.2 приняты следующие обозначения: типы данных для видов типов данных:

E - (elementary) - простой тип (BOOL, CHAR, BYTE, WORD, INT, DINT, REAL, S5TIME, TIME, DATE, TIME_OF_DAY);
C - (complex) - сложный тип (DATE_AND_TIME, STRING, ARRAY, STRUCT и UDT);
P - (POINTER) - указатель;
A - (ANY) - ANY-тип.

Таблица 27.2 Разрешенные виды типа данных (data type veiws)

Блок	Объявление переменной в блоке	Типы данных переменных			
		Простые (E) (elementary)	Сложные (C) (complex)	POINTER (P)	ANY (A)
FC	VAR_INPUT	E	C		
	VAR_OUTPUT	E	C		
	VAR_IN_OUT	E	C		
	VAR [1)]	E C	E C A		C
	VAR_TEMP	E C	E C A		C
FB	VAR_INPUT	E C	E C P A	C	C
	VAR_OUTPUT	E C	E C		
	VAR_IN_OUT	E	C		
	VAR	E C	E C		
	VAR_TEMP	E C	E C A		C

[1)] для временных локальных данных

27.2 Адресация

27.2.1 Абсолютная адресация

При абсолютной адресации адреса назначаются в соответствии с началом адресной области; например, I 1.0 (0-ой бит входного 1-го байта). Абсолютная адресация в SCL соответствует абсолютной адресации в стандартных языках программирования (см. раздел 3.3 "Адресация переменных") за исключением способа идентификации адресов глобальных данных, в котором есть отличие (см. табл. 27.3).

Таблица 27.3 Идентификация адресов при абсолютной адресации

Адресная область	Бит	Байт	Слово	Двойное слово
Входы	I y.x	IBy	IWy	IDy
Выходы	Qy.x	QBy	QWy	QDy
Периферийные входы	-	PIBy	PIWy	PIDy
Периферийные выходы	-	PQBy	PQWy	PQDy
Меркеры	My.x	MBy	MWy	MDy
Адреса глобальных данных	DBz.DXy.x	DBz.DBy	DBz.DWy	DBz.DDy
	DBz.Dy.x			

x = адрес бита, y = адрес байта, z = номер блока данных

В языке программирования SCL доступ к адресам глобальных данных возможен только способом полной адресации. Блок данных может быть переменной типа BLOCK_DB (см. также раздел 27.2.3 "Косвенная адресация в SCL").

Примечание:

В SCL не допускается присутствие разделителей (типа "пробел" или "табуляция") между адресом и идентификатором адреса.

Различия по сравнению со стандартными языками программирования: в SCL не применяется абсолютная адресация временных и статических локальных данных; вызов блока данных с частичным адресом невозможен; вычисление номера и размера глобального экземплярного блока данных невозможно.

27.2.2 Символьная адресация

При символьной адресации символьные имена должны быть назначены абсолютным адресам и переменным. Для глобальных данных имена назначаются в таблице символов; для локальных данных имена назначаются в разделе объявления переменных блока.

Символьная адресация в SCL соответствует символьной адресации в стандартных языках программирования (см. раздел 3.3 "Адресация переменных"). Может иметь место также использование смешанных абсолютно-символьных (mixed absolute/symbolic) идентификаторов, например таких:

```
DB10.Setpoint
"Motor1Data".DW12
```

Данные идентификаторы допустимы для доступа к глобальным данным с использованием полного адреса.

В SCL Вы можете назначать имена константам в разделе объявления блока и использовать в программе эти имена как символы.

27.2.3 Косвенная адресация в SCL

Косвенное назначение глобальных адресов

Косвенное использование глобальных адресов основывается на абсолютной адресации. При этом для указания расположения данных в памяти в квадратных скобках указывается переменная INT (две переменные INT в случае адресации бита):

- I[*byteindex.bitindex*];
- MB[*byteindex*],

где *byteindex* и *bitindex* являются константами или переменными, которые могут быть изменены в процессе обработки программы, или выражениями типа INT.

Таким способом Вы можете адресовать следующие области:

- периферийные входы PI и периферийные выходы PQ
  (в обоих этих случаях адрес бита не указывается);
- входы I, выходы Q, меркеры M;
- адреса глобальных данных D
  (блок данных и адрес данных);
- временные и статические локальные данные
  (только символьная адресация);
- Функции таймеров T и функции счетчиков C
  (для обоих этих функций адрес бита не указывается).

Косвенное назначение адресов глобальных данных

Косвенное использование адресов глобальных данных основывается на абсолютной адресации, но при этом адреса данных также как адреса блоков данных могут быть изменены в процессе обработки программы.

Вы можете использовать или абсолютный адрес или символьный адрес для блока данных:

- DB10.DX[*byteindex.bitindex*];
- MotorData.DW[*byteindex*],

где *byteindex* и *bitindex* являются константами или переменными, которые могут быть изменены в процессе обработки программы, или выражениями типа INT.

Применяя функцию преобразования WORD_TO_BLOCK, Вы можете назначить косвенный адрес блоку данных. Номер блока данных DB определяется либо как переменная, либо как выражение с типом данных WORD (см. пример на рис. 27.2).

- WORD_TO_BLOCK_DB[*dbindex*].DW0,

где *dbindex* является переменной, которая может быть изменена в процессе обработки программы, или выражением с типом данных WORD.

Если блок данных адресован косвенным способом, то для доступа к адресу данных не может использоваться символьное имя.

Обращение к адресам данных с помощью параметра блока BLOCK_DB

Если к блоку данных возможен доступ через параметр блока BLOCK_DB, то обращение к адресам данных в блоке может быть организовано и абсолютным, и косвенным способом (см. рис. 27.2). Пусть входной параметр *Data* имеет тип BLOCK_DB:

- Data.DW0;
- Data.DX2.0;
- Data.DW[*byteindex*];
- Data.DX[*byteindex.bitindex*],

где *byteindex* и *bitindex* являются константами или переменными, которые могут быть изменены в процессе обработки программы, или выражениями типа INT.

Если блок данных адресован через параметр блока BLOCK_DB, то для доступа к адресу данных не может использоваться символьное имя.

```
//***********************************************************************
//Пример косвенного использования глобальных адресов
k := 120; FOR i := 48 TO 62 BY 2 DO
MW[k] := PIW[i]; k := k + 2; END_FOR;

//***********************************************************************
//Косвенная адресация блоков данных
//Индекс DB имеет тип данных WORD
M0.0 := WORD_TO_BLOCK_DB(dbindex_w).DX0.0;
M0.0 := WORD_TO_BLOCK_DB(dbindex_w).DX[byteindex,bitindex];

//Индекс DB имеет тип данных INT
M0.0 := WORD_TO_BLOCK_DB(INT_TO_WORD(dbindex_i)).DX0.0;
M0.0 := WORD_TO_BLOCK_DB(INT_TO_WORD(dbindex_i)).DX[byteindex,bitindex];

//***********************************************************************
//Косвенная адресация посредством параметра блока
// Для имени "Data" и параметра с типом BLOCK_DB
M0.0 := Data.DX0.0;                  //абсолютная адресация
M0.0 := Data.DX[byteindex,bitindex]; //косвенная адресация
//***********************************************************************
```

Рис. 27.2 Пример косвенного использования глобальных адресов

Адресация массивов

В SCL в качестве индекса массива Вы можете использовать или константу, или переменную, или выражение типа INT, поэтому индекс может быть изменен в процессе выполнения программы. Вы можете также организовать доступ к части массива как к переменной (см. раздел 27.5.4 "Присвоение значений массивам").

При предопределении массивов отдельным размерностям массивов могут назначаться множители повторения.

27.3 Операторы

Выражения обеспечивают получение определенных значений. Выражение может включать в себя один адрес (идентификатор) данных (одну переменную) или несколько адресов (идентификаторов) данных (несколько переменных), которые объединяются с помощью операторов.

Пример:

```
a + b;
```

здесь *a* и *b* - это адреса (идентификаторы данных = переменные),

"+" - это оператор.

Порядок выполнения операторов в выражении определяется их взаимным расположением и приоритетом. Этот порядок выполнения операторов можно регулировать с помощью скобок. С точки зрения операторов выражения могут быть смешанными в соответствии с правилами, которые регламентируют в SCL комбинирование данных (как исходных, так и результатов вычислений), относящихся к разным типам данных.

Язык программирования SCL поддерживает операторы, список которых приводится в таблице 27.4.

Операторы, относящиеся к одному приоритетному классу, выполняются последовательно слева направо.

Таблица 27.4 Список операторов, поддерживаемые языком программирования SCL

Комбинирование	Наименование	Оператор	Приоритет
Скобки	(*Выражение*)	(,)	1
Арифметические	Степень	**	2
	Унарный плюс, унарный минус (знак)	+, -	3
	Умножение, деление	*, /, DIV, MOD	4
	Сложение, вычитание	+, -	5
Сравнения	Меньше, меньше или равно, больше, больше или равно,	<, <=, >, >=	6
	Равно, не равно	=, <>	7
Двоичные	Логическая операция отрицания (инвертирование)	NOT	3
	Логическая операция "И"	AND, &	8
	Логическая операция "Исключающее ИЛИ"	XOR	9
	Логическая операция "ИЛИ"	OR	10
Присвоение	Операция присваивания	:=	11

термин "унарный" означает, что оператор относится к одному адресу (переменной)

27.4 Выражения

Выражение - это формула, которая обеспечивает получение определенного значения. Выражение содержит адреса (идентификаторы) данных (переменные) и операторы. В простейшем случае выражение может включать в себя один адрес (идентификатор) данных, одну переменную или константу, а также может содержать знак или оператор

инвертирования.

Выражение может содержать адреса (идентификаторы) данных (переменные), объединенные в группы с помощью операторов. Сами выражения могут объединяться в группы с помощью операторов; при этом такие комбинированные выражения могут иметь очень сложную структуру. Порядок выполнения операторов в выражении обычно регулируется с помощью скобок.

Результат, полученный при выполнении выражения, может быть присвоен переменной или параметру блока, или этот результат может быть использован для проверки критерия условия в инструкции управления (control instruction).

Выражения могут быть разделены в соответствии со способом объединения (комбинирования) данных на арифметические выражения, логические выражения и выражения сравнения.

27.4.1 Арифметические выражения

Арифметическое выражение или состоит из численного значения или оно объединяет два значения или выражения с помощью арифметических операторов.

Пример:

```
Voltage * Current
```

В таблице 27.5 представлен перечень допустимых типов данных для арифметических выражений, а также для результатов арифметических выражений.

Спецификация класса типов данных ANY_NUM означает то, что тип данных первого и второго операнда может относиться к типу данных INT, DINT или REAL. Если Вы объединяете с помощью оператора операнд типа INT с операндом типа DINT, то результат должен быть типа DINT; если Вы объединяете с помощью оператора операнд типа INT или DINT с операндом типа REAL, то результат должен быть типа REAL. Перед тем как сформировать исполняемый блок, редактор выполняет (незаметно для пользователя) преобразование данных, то есть приводит данные из выражения к требуемому типу (см. также таблицу 30.4 "Функции неявного преобразования (Implicit Conversion Functions)").

В случае операции деления, второй операнд (делитель) не должен быть равным нулю.

На рис. 27.3 представлен пример арифметических выражений в сочетании с выражением присвоения значений.

27.4.2 Выражения сравнения

Выражение сравнения сравнивает значения двух операндов и выдает результат в виде булева значения; если условие сравнения выполняется,

то результат равен TRUE (ИСТИНА), иначе - результат равен FALSE (ЛОЖЬ).

Пример:

```
Voltage1 > Voltage2
```

Сравниваемые в выражении сравнения операнды должны относиться к одному типу данных или к одному классу типов данных (ANY_INT, ANY_NUM или ANY_BIT).

В таблице 27.5 представлен перечень допустимых типов данных для выражений сравнения, а также для результатов этих выражений.

Таблица 27.5 Допустимые типы данных операндов и результатов в SCL-выражениях

Операция	Оператор	1 операнд	2 операнд	Результат
Арифметические выражения				
Возведение в степень	**	ANY_NUM	INT	REAL
Умножение	*	ANY_NUM TIME	ANY_NUM ANY_INT	ANY_NUM TIME
Деление	/	ANY_NUM	ANY_NUM	ANY_NUM
Деление нацело	DIV	ANY_INT TIME	ANY_INT ANY_INT	ANY_INT TIME
Деление по модулю	MOD	ANY_INT	ANY_INT	ANY_INT
Сложение	+	ANY_NUM TIME TOD DT	ANY_NUM TIME TIME TIME	ANY_NUM TIME TOD TOD
Вычитание	-	ANY_NUM TIME TOD DATE TOD DT	ANY_NUM TIME TIME DATE TOD TIME	ANY_NUM TIME TOD TIME TIME DT
Выражения сравнения				
Меньше, меньше или равно, больше, больше или равно,	<, <=, >, >=	ANY_NUM CHAR или STRING TIME DATE TOD	ANY_NUM CHAR или STRING TIME DATE TOD	BOOL BOOL BOOL BOOL BOOL
Равно, не равно	=, <>	ANY_BIT	ANY_BIT	BOOL
Логические выражения				
Логическая операция отрицания	NOT	ANY_BIT	-	ANY_BIT
Логическая операция "И"	AND, &	ANY_BIT	ANY_BIT	ANY_BIT
Логическая операция "Исключающее ИЛИ"	XOR	ANY_BIT	ANY_BIT	ANY_BIT
Логическая операция "ИЛИ"	OR	ANY_BIT	ANY_BIT	ANY_BIT

Для улучшения ясности в выражениях сравнения рекомендуется использовать скобки.

Выражения сравнения могут быть объединены с логическими операторами, например, следующим образом:

```
(Value1 > 40)  AND NOT  (Value2 = 20)
```

Сравнение переменных типов CHAR выполняется в соответствии с кодом символов ASCII. Пользователю доступны IEC-функции для выполнения операций сравнения переменных, относящихся к типу данных STRING и DT. IEC-функции представляют собой загружаемые FC блоки, которые находятся в библиотеке стандартов *Standard Library* в разделе *IEC Function Blocks.*

На рис. 27.3 представлены несколько примеров выражений сравнения в сочетании с выражениями присвоения значений.

```
(********************* Операции присвоения ************************

Automatic := TRUE;                    //Присвоение значения константы
Setpoint  := StartSetpoint;           //Присвоение переменной
Deviation := ActualValue - Setpoint; //Присвоение выражения
Display   := INT_TO_WORD(Deviation); //Присвоение значения функции

(******************* Арифметические выражения **********************

Power     := Voltage * Current;
Volume    := 4/3 * PI * Radius**3;
Solutionl := -P/2 + SQRT(SQR(P/2)-Q);
MeanValue := (Motor[1].Power + Motor[2].Power)/2;

(********************* Выражения сравнения ************************

TooLarge  := Voltage_Act > Voltage_Set;
Warning   := (Voltage * Current) >= 20_000;
M101.0    := Setpoint = ActualValue;
IF Deviation > 2_000 THEN Display := 16#F002; END_IF;

(********************Логические выражения ************************

Q4.0    := II.0 & II.1;
ON      := (Manual_on OR Auto_pn) AND NOT Fault;
MW30    := MW32 AND Mask;
Pulses  := (Edge_mem_bits XOR ID16) AND ID16; Edge_mem_bits := ID16;

(*****************************************************************
```

Рис. 27.3 Примеры операций присвоения, арифметических, логических выражений и выражений сравнения

27.4.3 Логические выражения

Логическое выражение объединяет два операнда или выражения, относящихся к классу типов данных ANY_BIT, в соответствии с логикой AND (И), OR (ИЛИ) или XOR (Исключающее ИЛИ).

Пример:

```
Automatic  AND NOT  Manual_on
```

Логическое выражение также включает в себя (булево) инвертирование; эта операция аналогична изменению знака значения.

Логическое выражение выдает значение, относящееся к классу типов данных ANY_BIT. Результат логического выражения относится к типу BOOL, если оба операнда также имеют тип BOOL. Если один или оба операнда имеют тип BYTE, WORD или DWORD, то результат будет иметь тип данных более "требовательного к памяти" операнда.

На рис. 27.3 представлены несколько примеров логических выражений в сочетании с выражениями присвоения значений.

27.5 Присвоение значений

С помощью операции присвоения значения одна переменная получает значение другой переменной или значение выражения. Слева от оператора присваивания ":=" находится переменная, которая принимает значение другой переменной или выражения, которые в свою очередь находятся справа от оператора присваивания.

Тип данных, находящихся с двух сторон от знака присваивания, должен быть идентичен. Исключение составляет только случай присваивания в функции "неявного изменения типа данных" ("Implicit data type conversion"): если тип данных переменной характеризуется по крайней мере таким же размером в битах или имеет больший размер в битах, чем тип данных выражения, то тип данных выражения "неявно" конвертируется (значение выражения автоматически конвертируется в требуемый тип данных и присваивается переменной). Другими словами, "неявное изменение типа данных" (с помощью функции преобразования типа данных) является необходимым.

27.5.1 Присвоение значений в случае простых типов данных

С помощью операции присвоения значение константы, переменной, адреса или выражение может быть присвоено переменной или адресу (см. рис. 27.3).

Абсолютные адреса (например, MW 10) имеют тип данных ANY_BIT; они имеют тип данных, определяемый размером занимаемой области (типы BOOL, BYTE, WORD, DWORD). Если Вы хотите назначить значение отличающегося типа данных абсолютному адресу, то используйте преобразование типа данных, или назначьте этому адресу имя и требуемый тип данных в таблице символов.

27.5.2 Присвоение значений переменным типов DT и STRING

Каждой DT-переменной может быть назначено значение другой DT-

переменной или DT-константы.

Каждой STRING-переменной может быть назначено значение другой STRING-переменной или строки символов. Если назначаемая строка символов длиннее, чем переменная, стоящая слева от оператора присваивания, то на этапе компиляции пользователь получит предупреждающее сообщение.

Нельзя выполнять предопределение в разделе объявления в области временных локальных данных. Если Вы используете функции обработки STRING-операндов, например, IEC-функции, с помощью которых STRING-переменная проверяется (так, как выходной параметр), то Вы должны запрограммировать предопределенное выходное значение.

27.5.3 Присвоение значений структурам

С помощью операции присвоения одной STRUCT-переменной может быть назначено значение другой STRUCT-переменной только в том случае, если:

- структуры этих данных согласованы;
- компоненты структур согласованы с точки зрения типов данных;
- компоненты структур согласованы с точки зрения имен.

Отдельные компоненты структур могут быть обработаны как переменные одного типа данных; например, значение компонента структуры *Motor1.Setpoint* типа INT может быть присвоено другой INT-переменной, или какое-либо целое (INT) значение может быть присвоено данному компоненту структуры.

27.5.4 Присвоение значений массивам

С помощью операции присвоения одной ARRAY-переменной может быть назначено значение другой ARRAY-переменной только в том случае, если согласованы типы данных элементов этих массивов, а также граничные значения индексов (наименьшее и наибольшее значения) в каждой размерности, а также число размерностей для этих массивов совпадают.

Отдельные компоненты массивов могут быть обработаны так же как переменные соответствующего типа данных.

В случае использования массивов с несколькими размерностями, Вы можете работать с частями массивов как с массивами соответствующей размерности (ARRAY-переменными соответствующей размерности): если отбрасывать те или иные индексы массива, то можно получать массивы с меньшей размерностью по сравнению с исходным массивом данных.

Пример: пусть исходный массив задан следующим образом:

```
Field1 : ARRAY [1..8,1..16] OF INT;
```

Таким образом, массив `Field1` представляет собой двумерный массив.

Исходя из условий данного примера, Вы можете:

- обращаться к массиву в целом, используя идентификатор `Field1`;
- обращаться к части массива, используя идентификатор `Field1[i]` (что соответствует строке матрицы);
- обращаться к элементу массива, используя идентификатор `Field1[i,j]`.

Вы можете также присваивать часть массива `Field1[i]` другому массиву (то есть, другой ARRAY-переменной), имеющему соответствующую размерность, например:

```
Field12 := Field1[i],
```

где:

i может принимать значения от 1 до 8;

массив `Field2` объявлен следующим образом:

```
Field2  :  ARRAY [1..16] OF INT;
```

28 Операторы управления (Control Statements)

С помощью операторов управления (control statements) пользователь может организовать ветвление программы, циклическое выполнение отдельных фрагментов программы и осуществлять переход в программе блока для выполнения другой ее части. Язык программирования SCL поддерживает следующие операторы управления:

- IF (оператор для выполнения ветвления в программе по условию, проверяемому в отношении булевой переменной (или параметра типа BOOL));
- CASE (оператор для выполнения ветвления в программе по условию, проверяемому в отношении целой переменной (или параметра типа INT));
- FOR (оператор для организации в программе циклов с переменной - счетчиком циклов);
- WHILE (оператор для организации в программе циклов, инициируемых при выполнении определенного условия);
- REPEAT (оператор для организации в программе циклов с завершением по условию);
- CONTINUE (оператор для завершения текущего прохода цикла в программе);
- EXIT (оператор для выхода из цикла в программе);
- GOTO (оператор для продолжения выполнения программы, начиная с метки перехода);
- RETURN (оператор для выхода из программы блока).

28.1 Оператор IF

Оператор IF управляет выполнением той или иной части программы в зависимости от состояния булевой переменной. С помощью оператора IF пользователь может запрограммировать выполнение различных, определяемых условиями, ветвей программы.

```
IF  condition
  THEN  statements;
END_IF;
```

Здесь `condition` - это адрес или выражение с типом BOOL. Если `condition` имеет значение TRUE (ИСТИНА), то выполняются операторы после ключевого слова THEN. Если `condition` имеет значение FALSE (ЛОЖЬ), то выполняются операторы после ключевого слова END_IF. Ключевое слово END_IF завершает оператор IF.

```
IF  condition
   THEN  statements1;
   ELSE  statements0;
END_IF;
```

В данном примере, как и в предыдущем, `condition` имеет значение TRUE (ИСТИНА) или FALSE (ЛОЖЬ). Если `condition` имеет значение TRUE (ИСТИНА), то выполняются операторы после ключевого слова THEN. Если `condition` имеет значение FALSE (ЛОЖЬ), то выполняются операторы после ключевого слова ELSE.

```
IF  condition1
   THEN statements1;
   ELSEIF  condition2
     THEN  statements2;
   ELSE statements0;
END_IF;
```

Оператор `IF` может быть вложенным. В данном примере, как и в предыдущем, если `condition1` имеет значение TRUE (ИСТИНА), то выполняются операторы `statements1` после первого ключевого слова THEN, и далее программа будет выполняться, начиная с операторов после ключевого слова END_IF. Если `condition1` имеет значение FALSE (ЛОЖЬ), то выполняется проверка условия `condition2` после ключевого слова ELSEIF. Если `condition2` имеет значение TRUE (ИСТИНА), то выполняются операторы `statements2` после второго ключевого слова THEN, и далее программа будет выполняться, начиная с операторов после ключевого слова END_IF. Если `condition2` имеет значение FALSE (ЛОЖЬ), то выполняются операторы `statements0` после ключевого слова ELSE.

Пользователь может использовать любое количество комбинаций ключевых слов ELSEIF ... THEN ... между ключевыми словами IF ... THEN ... и ELSE. Ключевое слово ELSE и последующие операторы не являются обязательными.

Пример:

Если переменная *Actual_value* больше, чем переменная *Setpoint*, то выполняются операторы, идущие после ключевого слова THEN. Если, наоборот, переменная *Actual_value* меньше, чем переменная *Setpoint*, то выполняются операторы, идущие после ключевого слова ELSEIF. Если оба выражения сравнения не выполняются, то выполняются операторы после ключевого слова ELSE.

```
IF  Actual_value > Setpoint
   THEN greater_than := TRUE;
        less_than    := FALSE;
        equal_to     := FALSE;
```

```
   ELSEIF  Actual_value < Setpoint
       THEN  greater_than := FALSE;
             less_than    := TRUE;
             equal_to     := FALSE;
       ELSE  greater_than := FALSE;
             less_than    := FALSE;
             equal_to     := TRUE;
END_IF;
```

28.2 Оператор CASE

Оператор CASE позволяет выбрать для выполнения нужную последовательность операторов в программе в зависимости от значения целой переменной (параметра типа INT).

Общая структура программы с оператором CASE может иметь следующую форму:

```
CASE Selection OF
  CONST1 :    statements1;
  CONST2 :    statements2;
  ...
  CONSTx :    statementsx;
  ELSE   :    statements0;
END_CASE;
```

Здесь `Selection` - это адрес или выражение с типом INT. Если `Selection` имеет значение `CONST1`, то выполняются операторы `statements1`. Если `Selection` имеет значение `CONST2`, то выполняется цепочка операторов `statements2` и так далее.

Если `Selection` имеет значение, находящееся за рамками списка значений, указанных для проверки, то выполняется цепочка операторов после ключевого слова `ELSE`. При этом цепочка операторов, начинающаяся с ключевого слова `ELSE`, не является обязательной.

Список значений CONST1, CONST2 и т.д. состоит из целых (INT) констант. Для этих констант могут использоваться несколько вариантов форматов записи при использовании оператора CASE, то есть в качестве константы-"переключателя" CONSTx могут быть указаны:

- одиночное целое (INT) число;
- диапазон целых (INT) чисел (например: 15..20);
- смесь разделенных запятыми отдельных целых (INT) чисел и диапазонов целых чисел (например: 21,25,31..33).

При этом каждое значение константы-"переключателя" CONSTx должно быть уникально, оно может появится в списке одного оператора CASE только один раз.

Оператор CASE может быть вложенным.

Вместо отдельной цепочки операторов в списке оператора CASE может стоять другой оператор CASE.

Пример:

Значение, присваиваемое переменной *Error_number*, зависит от переменной *ID*.

```
CASE ID OF
   0      :    Error_number := 0;
   1,3,5  :    Error_number := ID + 128;
   ...
   6..10  :    Error_number := ID;
   ELSE   :    Error_number := 16#7F;
END_CASE;
```

28.3 Оператор FOR

Оператор FOR для организации в программе циклов с переменной - счетчиком циклов. Выполнение внесенного в цикл фрагмента программы будет повторяться столь долго, пока переменная "счетчик циклов" будет оставаться в указанном диапазоне значений.

Общая структура программы с оператором FOR может иметь следующую форму:

```
FOR i := limit1
      TO limit2
      BY step
      DO statements;
END_FOR;
```

В начале обработки данного оператора начальное значение `limit1` присваивается счетчику циклов `i`. Вы можете сами определять переменную-счетчик циклов; это должна быть переменная с типом INT или с типом DINT. Начальное значение `limit1` может быть выражением с типом INT или DINT, также как и конечное значение `limit2` и шаг изменения переменной-счетчика цикла `step`.

В начале выполнения цикла переменная-счетчик цикла получает указанное после оператора присваивания начальное значение. В то же самое время рассчитываются и запоминаются конечное значение `limit2` и шаг изменения переменной-счетчика цикла `step` (изменение этих значений во время выполнения цикла не будет иметь никаких последствий). После этого проверяется условие завершения выполнения цикла, и если условие не выполняется, то выполняется программа внутри цикла.

После каждого прохода программы цикла счетчик цикла увеличивается на величину шага приращения `step` (если шаг указан как положительное число) или уменьшается на величину шага приращения `step` (если шаг указан как отрицательное число).

При программировании цикла строка `BY step` (то есть, начиная с ключевого слова BY) не является обязательной. Если такое условие для шага изменения переменной-счетчика цикла отсутствует, то шаг (по умолчанию) принимается равным +1. Если величина переменной-счетчика цикла выходит за пределы диапазона значений между начальным значением и конечным значением `(limit1...limit2)`, то программа продолжает выполняться сразу после завершающего ключевого слова END_FOR.

Последний проход цикла выполняется при значении переменной-счетчика цикла, равном конечному значению, или (если шаг приращения таков, что конечное значение `limit2` не может быть достигнуто точно) при значении переменной-счетчика, равном разности конечного значения и шага приращения `(limit2 - step)`. После полного окончания обработки цикла значение переменной-счетчика цикла равно ее значению при последнем проходе цикла, суммированному с величиной шага приращения.

Оператор FOR может быть вложенным: внутри цикла с использованием оператора FOR могут быть запрограммированы другие циклы с использованием оператора FOR с другими переменными-счетчиками цикла.

Внутри цикла с использованием оператора FOR может быть запрограммирован переход к началу цикла (с использованием оператора управления CONTINUE) или полный выход из цикла для продолжения выполнения программы, начиная сразу же после ключевого слова END_FOR, (с использованием оператора управления EXIT).

Пример:

Пусть, необходимо считать слова с PIW 128 по PIW 142 из области периферии в область меркеров в слова с MW 128 по MW 142.

```
FOR i := 128 TO 142 BY 2 DO
    MW[i] := PIW[i];
END_FOR;
```

28.4 Оператор WHILE

Оператор WHILE служит для организации в программе циклов, выполнение которых продолжается все время, пока выполняется определенное условие.

Общая структура программы с оператором WHILE может иметь следующую форму:

```
WHILE  Condition  DO
    Statements;
END_WHILE;
```

Здесь `Condition` - это адрес или выражение с типом BOOL. Пока выполняется определенное условие (то есть, пока `Condition = TRUE(ИСТИНА)`), будут циклически выполняться выражения `Statements`.

Перед каждым проходом выполняется проверка условия `Condition`. Если условие не выполняется (`Condition = FALSE(ЛОЖЬ)`), то программа продолжает выполняться, начиная сразу же после ключевого слова END_WHILE. Такой вариант событий возможен даже, если не было ни одного прохода программы цикла (то есть до первого прохода цикла), что означает, что операторы `Statements` при этом ни разу не будут выполнены.

Оператор WHILE может быть вложенным; при этом внутри одного цикла с оператором WHILE могут размещаться другие циклы с оператором WHILE.

Внутри цикла с использованием оператора WHILE может быть запрограммирован переход к началу цикла (с использованием оператора управления CONTINUE) или полный выход из цикла для продолжения выполнения программы, начиная сразу же после ключевого слова END_WHILE, (с использованием оператора управления EXIT).

Пример:

Пусть, необходимо найти в блоке данных DB10 битовое значение 16#FFFF. Слово данных DW0 содержит или 16#FFFF, или интервал перед следующим словом, которое снова содержит или 16#FFFF, или интервал перед следующим словом.

```
i := 0;
WHILE DB10.DB[i] := 16#FFFF DO
    i := i + WORD_TO_INT(DB10.DB[i]);
END_WHILE;
```

28.5 Оператор REPEAT

Оператор REPEAT служит для организации в программе циклов, выполнение которых продолжается все время, пока не встретится условие завершения обработки цикла.

Общая структура программы с оператором REPEAT может иметь следующую форму:

```
REPEAT
    Statements;
    UNTIL Condition
END_REPEAT;
```

Здесь `Condition` - это адрес или выражение с типом BOOL. Пока не выполняется определенное условие (то есть, пока `Condition = FALSE(ЛОЖЬ)`), будут циклически выполняться выражения `Statements`.

После каждого прохода цикла выполняется проверка условия `Condition`. Если условие выполняется (`Condition = TRUE(ИСТИНА)`), то цикл далее не обрабатывается и выполнение программы будет продолжено сразу же после ключевого слова END_REPEAT.

Таким образом, программа цикла будет обработана по крайней мере один раз, даже если при первом проходе цикла выполняется условие завершения его обработки, что означает, что операторы `Statements` будут выполнены по крайней мере один раз.

Оператор REPEAT может быть вложенным; при этом внутри одного цикла с оператором REPEAT могут размещаться другие циклы с оператором REPEAT.

Внутри цикла с использованием оператора REPEAT может быть запрограммирован переход к началу цикла (с использованием оператора управления CONTINUE) или полный выход из цикла для продолжения выполнения программы, начиная сразу же после ключевого слова END_REPEAT, (с использованием оператора управления EXIT).

Пример:

Пусть, необходимо вызывать системную функцию SFC 25 COMPRESS в программе перезапуска, пока не будет завершено "сжатие" памяти пользователя.

```
REPEAT
    SFC_ERROR := COMPRESS(
         BUSY := busy,
         DONE := done);
    UNTIL done
END_REPEAT;
```

28.6 Оператор CONTINUE

Оператор CONTINUE служит для завершения текущего прохода цикла в программе, организованного с помощью операторов FOR, WHILE или REPEAT.

После выполнения оператора CONTINUE проверяются условия для выполнения следующего прохода программы цикла (в случае циклов, организованных с помощью операторов REPEAT или WHILE) или (в случае циклов, организованных с помощью оператора FOR) переменная-счетчик цикла изменяется на величину шага приращения, и далее следует проверка, находится ли эта переменная в допустимых для нее пределах?

Если проверяемое условие соблюдается, то после оператора CONTINUE выполняется следующий проход программы цикла с самого начала.

Пример:

Пусть, необходимо установить меркеры с помощью двух вложенных циклов с использованием оператора FOR; если байтовый адрес (i) равен 0, а битовый адрес (k) меньше 2, то операторы тела внутреннего цикла FOR не выполняются (установка битов должна начинаться с меркера M0.3).

```
FOR i := 0 TO 2 DO
  FOR k := 0 TO 7 DO
     IF (k<2 & i=0)
     THEN CONTINUE;
     END_IF;
     M[i,k] := TRUE;
  END_FOR;
END_FOR;
```

28.7 Оператор EXIT

Оператор EXIT служит для полного завершения обработки цикла (с выходом из него), организованного с помощью операторов FOR, WHILE или REPEAT. При этом выход из цикла с оператором EXIT не зависит от выполнения условий, проверяемых в цикле, и может производиться из любой точки цикла. При выходе из цикла с оператором EXIT программа продолжает выполняться сразу же после ключевых слов END_FOR, END_WHILE или END_REPEAT.

Выход из цикла с оператором EXIT происходит немедленно из точки внутри программы цикла, где этот оператор встретился.

Пример:

Пусть, необходимо установить меркеры с помощью двух вложенных циклов с использованием оператора FOR; если байтовый адрес (i) равен 2, а битовый адрес (k) больше 5, то выполнение внутреннего цикла FOR прерывается (установка битов должна заканчиваться на меркере M2.5).

```
FOR i := 0 TO 2 DO
  FOR k := 0 TO 7 DO
     IF (k=2 & i>5)
     THEN EXIT;
     END_IF;
     M[i,k] := TRUE;
  END_FOR;
END_FOR;
```

В данном примере выполнение цикла FOR прекращается при определенном условии для счетчика цикла *k* с помощью оператора EXIT. Выполнение внешнего цикла FOR с счетчиком *i* не связан с выходом из цикла. Тем не менее, пример составлен таким образом, что выход из цикла FOR производится на последнем проходе цикла с счетчиком *i*.

28.8 Оператор RETURN

Оператор RETURN служит для безусловного выхода из текущего блока.

Выполнение программы при этом будет продолжено либо в вызывающем блоке, либо в операционной системе (если выход с оператором RETURN происходит в организационном блоке).

Оператор RETURN не является обязательным оператором в конце блока.

Оператор RETURN пересылает состояние сигнала переменной OK на выход ENO завершаемого (с RETURN) блока.

Пример:

Пусть, необходимо организовать выход из блока по условию.

```
IF Error <> 0 THEN RETURN;
END_IF;
```

28.9 Оператор GOTO

Оператор GOTO служит для продолжения выполнения программы начиная с другой точки.

Пример:

Пусть, необходимо организовать выход из блока по условию.

```
        GOTO M1;
        ...;        //Пропущенные операторы
        ...;        //Пропущенные операторы
M1:     ...;        //Точка - цель перехода
```

Связь между оператором GOTO и точкой перехода обеспечивается меткой перехода. Вы должны объявить метки перехода в разделе объявлений блока между ключевыми словами: LABEL и END_LABEL. Имя меток перехода имеет такую же структуру, как и имя локальной переменной блока.

Метка перехода должна быть уникальной (не должна повторяться). Вы можете организовать в программе несколько переходов с помощью нескольких операторов перехода GOTO к одной и той же метке перехода.

После выполнения перехода к некоторой метке программа продолжает выполняться со строки, отмеченной данной меткой. Метка перехода и выражение в этой же строке должны быть разделены двоеточием.

За меткой перехода всегда следует оператор (выражение). Допустим также "пустой" оператор:

```
Label1: ;
```

Итак, если Вы используете оператор GOTO, то помните следующие правила:

- ель перехода должна быть внутри того же блока программы, что и оператор GOTO;
- цель перехода должна быть однозначно определена;
- Вы не можете перейти "извне" внутрь цикла, но из цикла переход возможен.

Пример:

```
LABEL
      M1, M2, M3, End;
END_LABEL
...
CASE Selection OF
1 : GOTO M1;
2 : GOTO M2;
3 : GOTO M3;
ELSE GOTO End;
END_CASE;
M1: ... statement1 ...;
GOTO End;
M2: ... statement2 ...;
GOTO End;
M3: ... statement3 ...;
END:;
```

Примечание:

Оператор GOTO не определен как стандартный. Язык SCL поддерживает все операторы и функции, необходимые для структурного программирования, поэтому без оператора GOTO можно обойтись.

29 SCL-блоки

29.1 SCL-блоки: общая информация

В языке программирования SCL используется точно такая же структура блока, как и в стандартных языках программирования. Вы можете запрограммировать отдельные блоки с использованием языка программирования SCL, а затем вызывать их, скажем, в FBD-блоке, и Вы также можете из SCL-блоков вызывать блоки, созданные с использованием языка STL.

Для того, чтобы обрабатывать в пользовательской программе блоки, созданные с использованием различных языков программирования, интерфейс блока должен иметь стандартную структуру. Это особенно касается инициализации входа EN и выхода ENO (см. раздел 29.4 "Механизм EN/ENO").

Примеры программ, рассматриваемые в данной главе, Вы можете найти на прилагаемой дискете в библиотеке SCL_Book library в программе "29 Block Calls" ("Вызов блоков").

Структура программы пользователя

Организационный блок представляет собой интерфейс между операционной системой и пользовательской программой. Организационные блоки вызываются операционной системой CPU при свершении определенных событий, таких, например, как прерывания. Для управления программируемым контроллером пользовательская "обычная" программа выполняется в циклическом режиме. Эта программа располагается в организационном блоке OB 1, который используется практически всеми пользовательскими программами. Начало пользовательской программы соответствует первому сегменту в блоке OB 1 (см. раздел 3.1 "Обработка программы").

В соответствии со своим пониманием функциональной структуры пользовательской программы Вы можете разбить ее в OB1 на отдельные подпрограммы (или "блоки" - "blocks"). Пользовательская программа размещается в кодовых блоках (Code Block) и в блоках данных (Data Block), где хранятся данные пользователя. Кодовые блоки - это и есть собственно подпрограммы, которые Вы должны вызывать для выполнения (см. раздел 20.1 "Организация программы").

Блоки

Система STEP 7 поддерживает функции FC и функциональные блоки FB как кодовые блоки. Функциональные блоки FB вызываются вместе с блоками данных, в которых хранятся локальные переменные блока (как

бы "память" блока). Такой блок данных, назначаемый вызову FB, называется "экземплярным блоком данных" ("*instance data block*"); он может быть или собственно блоком данных, или может быть частью блока данных более высокого уровня ("higher-level" data block). Функции FC не имеют блоков данных, но они могут иметь "функциональное значение" ("*function value*"). Это функциональное значение позволяет, например, в одном арифметическом выражении использовать в качестве операнда функцию FC (точнее, ее функциональное значение) наряду с другими переменными (см. раздел 3.2 "Блоки").

Оба упомянутых блока могут иметь параметры блока. Параметры блока позволяют параметризировать "правила" его обработки (функцию блока).

При программировании блока Вы можете объявить параметр блока как "входной параметр" ("*input parameter*") (VAR_INPUT), если необходимо только считывать (сканировать) значение параметра, или как "выходной параметр" ("*output parameter*") (VAR_OUTPUT), если необходимо только записывать значение параметра, или как "входной/выходной параметр" ("*in-out parameter*") (VAR_IN_OUT), если в программе необходимо как считывать, так и записывать значение параметра.

Обращение к параметру блока в программном блоке выполняется с "использованием формального параметра" ("*formal parameter*") по его имени (имени параметра блока). Формальный параметр служит своего рода макетом для "фактического параметра" ("*actual parametr*"), используемого CPU во время выполнения программы. При вызове блока фактические параметры назначаются параметрам блока; они представляют собой значения, которые должны поступить в вызываемый блок, и с которыми этот блок обрабатывается.

29.2 Программирование SCL-блоков

Средства программирования SCL-блоков описаны в главе 2 "Программное обеспечение STEP 7"; соответствующий список ключевых слов Вы можете найти в разделе 3.5 "Программирование кодовых блоков на SCL". Блоки данных и данные пользовательского типа UDT в основном программируются точно так же как и в языке программирования STL (см. раздел 3.6 "Программирование блоков данных" и раздел 24.3 "Пользовательский тип данных").

Для того, чтобы обозначить разницу в программировании различных кодовых блоков, а точнее - разницу при использовании различных языков программирования, рассмотрим реализацию функции ограничителя "Delimiter" из главы 27 "Введение. Элементы языка" в соответствии со следующим планом реализации программы:

- в виде функции FC 291 без возвращаемого значения функции;
- в виде функции FC 292 с возвращаемым значением функции;
- в виде функционального блока FB 291 со своим собственным блоком данных DB 291;
- в виде функционального блока FB 291 как локального экземпляра в функциональном блоке FB 290.

После этого мы организуем вызов всех блоков в функциональном блоке (FB 290 с DB 290 в качестве экземплярного блока данных). Во всех случаях программа всегда будет одна и та же. Изменяются только объявление и инициализация параметров.

Примечание:

Так как программа функции ограничителя "Delimiter" не сохраняет локальные данные и возвращает свое значение, то вариант программной реализации в виде функции FC с функциональным значением является оптимальным типом блока.

29.2.1 Функции FC без возвращаемого значения функции

Функция FC без возвращаемого значения функции имеет тип данных VOID. В нашем примере функция FC 291 имеет входные параметры MAX, IN, MIN и выходной параметр OUT.

```
FUNCTION FC291 : VOID
VAR_INPUT
   MAX : INT;
   IN  : INT;
   MIN : INT;
END_VAR
VAR_OUTPUT
   OUT : INT;
END_VAR;
BEGIN
IF IN > MAX THEN OUT := MAX;
   ELSIF IN < MIN THEN OUT := MIN;
   ELSE OUT := IN;
END_IF;
END_FUNCTION
```

Все выходные параметры простых типов в функции должны быть соответствующим образом определены заданным алгоритмом и должны вычисляться в процессе обработки программы. Входные параметры функции могут быть только считаны, а выходные параметры могут быть только записаны.

29.2.2 Функции FC с возвращаемым значением функции

Функция FC с возвращаемым значением функции имеет тип данных, соответствующий функциональному значению (возвращаемому значению функции). В нашем примере функция FC 292 имеет входные параметры MAX, IN, MIN и функциональное значение (возвращаемое значение

функции), которое имеет адрес (имя) функции или в абсолютной или в символьной форме. Тип данных функционального значения должен быть определен после имени блока, отделенный от него двоеточием.

```
FUNCTION FC292 : INT
VAR_INPUT
   MAX : INT;
   IN  : INT;
   MIN : INT;
END_VAR
BEGIN
IF IN > MAX THEN FC292 := MAX;
   ELSIF IN < MIN THEN FC292 := MIN;
   ELSE FC292 := IN;
END_IF;
END_FUNCTION
```

В качестве типа данных для функционального значения Вы можете использовать любые простые типы, а также следующие типы данных: DATE_AND_TIME, STRING и пользовательские типы UDT. Типы данных ARRAY, STRUCT, POINTER и ANY в данном случае не допускаются.

Если функциональное значение относится к типу STRING, то для него будет зарезервирована длина, определенная в установках компилятора (но не максимальная длина, заданная в квадратных скобках в разделе объявления параметров).

Все выходные параметры простых типов в функции должны быть соответствующим образом определены заданным алгоритмом и должны вычисляться в процессе обработки программы. Входные параметры функции могут быть только считаны, а выходные параметры могут быть только записаны.

В соответствии с программой функции FC функциональное значение должно быть определено в выражении, относящемся к тому же типу данных. Определение этого значения и присвоение его возвращаемому значению функции должно быть выполнено также в процессе обработки программы.

29.2.3 Функциональный блок FB

Функциональному блоку соответствует экземплярный блок, в котором хранятся переменные функционального блока (функциональный блок или вызывается со своим собственным блоком данных, или он использует блок данных вызывающего функционального блока).

В нашем примере мы хотим использовать это и объявим предельные значения для функции ограничителя статическими локальными переменными. Только входная переменная IN и выходная переменная OUT остаются параметрами блока.

```
FUNCTION_BLOCK FB291
VAR_INPUT
   IN  : INT;
END_VAR
VAR_OUTPUT
   OUT : INT;
END_VAR;
VAR
   MAX : INT := 10_000;
   MIN : INT := -5_000;
END_VAR
BEGIN
IF IN > MAX THEN OUT := MAX;
   ELSIF IN < MIN THEN OUT := MIN;
   ELSE OUT := IN;
END_IF;
END_FUNCTION_BLOCK
```

Входные параметры блока могут быть только считаны, а выходные параметры могут быть только записаны.

Различают два варианта вызовов: вызов с собственным блоком данных или вызов в режиме локального экземпляра. Вариант последовательного вызова блока не должен рассматриваться при программировании функционального блока. Тем не менее, необходимо обеспечить, чтобы при использовании локального экземпляра по крайней мере один параметр блока или одна статическая локальная переменная были доступны: длина экземпляра не должна быть равна нулю.

Примечание:

Входные и выходные параметры сложных типов сохраняются в экземплярном блоке данных в виде значений, а входные-выходные параметры сохраняются в виде указателей на фактические параметры (см. раздел 26.3.2 "Хранение параметров в функциональных блоках").

29.2.4 Временные локальные данные

Все кодовые блоки имеют область для временных локальных данных, которую Вы можете использовать для промежуточного хранения данных в блоке. При программировании на языке SCL Вы можете использовать временные локальные данные точно также, как при программировании на стандартных языках. Для получения более подробной информации Вы можете обратиться к разделу 18.1.5 "Временные локальные данные".

Вы должны объявить временные локальные данные в разделе объявлений блока после ключевого слова VAR_TEMP. Здесь допускается применять все простые, сложные, пользовательские (UDT) типы данных, а также типы данных POINTER и ANY. Для типа данных ANY существуют специальные правила использования (см. далее).

Временные локальные данные не могут быть предопределены значениями на этапе объявления. Вот почему, при назначении L-стека редактор резервирует для переменных типа STRING область с размером, который вводится на вкладке "Compiller" ("Компилятор"), выбираемой с использованием опций меню: *Option -> Customize (Опции -> Установки пользователя)*.

Если временные локальные данные должны получить конкретные значения, они должны быть сначала записаны. Это также касается временных переменных типа STRING, полученных в качестве выходного параметра, например, при использовании IEC-функций. При записи значения IEC-функция проверяет корректность введенной информации о размере строковой переменной. Вы можете обратиться к ней, назначая значение (любое значение) переменной в программе перед ее использованием.

При программировании на языке SCL Вы можете объявлять переменные, относящиеся к одному типу данных, списком:

```
VAR_TEMP
   VALUE1, VALUE2, VALUE3 : INT;
   ...
END_VAR
```

Необходимо отметить, что при программировании на языке SCL временные локальные данные могут быть адресованы только символьным способом.

Тип данных ANY

Временные локальные данные типа ANY могут хранить адреса инструкций, а также глобальных переменных или локальных переменных блока:

```
ANY_VAR := MW10;
ANY_VAR := Setpoint;
ANY_VAR := DB10.Field;
```

Временные локальные данные типа ANY могут быть предопределены посредством ключевого слова NIL, указывающим на "нуль":

```
ANY_VAR := NIL;
```

Пример:

Пусть, различные записи данных должны копироваться в "почтовый ящик" для пересылки ("send mailbox") с помощью системной функции SFC 20 BLKMOV в зависимости от идентификатора:

```
VAR_TEMP
   Address := ANY;
END_VAR
...
CASE Identifier OF
1: Address := DataRecord1;
2: Address := DataRecord2;
...
```

```
ELSE Address := NIL;
END_CASE;
SFC_ERROR := BLKMOV(
   SRCBLK := Address,
   DSTBLK := SendMailbox);
```

Вы можете редактировать отдельные компоненты ANY-указателя, такие как номер DB или адрес, непосредственно с помощью видов типа данных (см. раздел 27.1.9 "Виды типа данных (Data Type Veiws)").

29.2.5 Статические локальные данные

Статические локальные данные - это "память" функционального блока. Эти данные располагаются в экземплярном блоке данных. При этом значения данных сохраняются до тех пор, пока не будут изменены из программы точно также, как изменяются значения переменных в глобальных блоках данных.

В статических локальных данных Вы также можете объявлять локальные экземпляры функциональных блоков и системных функциональных блоков. Для получения более подробной информации Вы можете обратиться к разделу 18.1.6 "Статические локальные данные".

Статические локальные данные объявляются с помощью ключевых слов VAR и END_VAR. В статических локальных данных допускается применять все простые, сложные, пользовательские (UDT) типы данных, а также типы данных POINTER и ANY.

При программировании на языке SCL Вы можете списком объявлять переменные, относящиеся к одному типу данных. Переменные, объявленные таким способом, не могут получить предопределение (инициализирующее значение).

Пример:

```
VAR
   VALUE1, VALUE2, VALUE3 : INT;
   VALUE4                 : INT:=3;
   ...
END_VAR
```

Необходимо отметить, что при программировании на языке SCL статические локальные данные в функциональном блоке могут быть адресованы только символьным способом.

Так как статические локальные данные располагаются в блоке данных, то доступ к ним может быть организован таким же образом, как к глобальным данным. Доступ к этим данным обеспечивается с использованием полного адреса, с определением блока данных и адреса данных.

29.2.6 Параметры блока

Параметры блока обеспечивают связь между вызывающим и вызываемым блоками. Эти параметры могут быть объявлены как входные (Input), входные-выходные (In/Out) и выходные (Output) параметры (см. раздел 19.1.3 "Объявление параметров блока").

Входные (Input) параметры могут только быть считаны, выходные (Output) параметры могут только быть записаны. Поэтому, если Вам требуются такие параметры, которые могут быть считаны, потом изменены и, наконец, снова записаны, то Вам необходимо использовать входные-выходные (In/Out) параметры.

В случае использования функций FC, параметры блока являются указателями на фактические параметры или на другие указатели. В случае использования функциональных блоков FB, параметры блока сохраняются в экземплярных блоках данных (см. раздел 26.3 "Сохранение данных при передаче параметров").

При программировании на языке SCL Вы можете списком объявлять переменные, относящиеся к одному типу данных. Переменные, объявленные таким способом, не могут получить предопределение (инициализирующее значение).

Пример:

```
VAR_INPUT
   VALUE1, VALUE2, VALUE3 : INT;
   ...
END_VAR
```

Так как параметры блока размещаются в блоке данных, то доступ к ним может быть организован таким же образом, как к глобальным данным. Доступ к этим данным обеспечивается с использованием полного адреса, с определением блока данных и адреса данных.

```
Result := DB279.DW20;
Result := DB279.Total;
Result := Totalizer.Total;
Result := Totalizer.DW20;
```

Фактически дальнейшая обработка возможна только для значений выходных (Output) параметров (см. вопросы, касающиеся вызовов блоков в разделе 29.3.3 "Функциональный блок со своим собственным блоком данных", а также раздел 29.3.4 "Функциональный блок как локальный экземпляр").

Предварительная инициализация параметров блоков

Предварительная инициализация параметров блоков не обязательна и допускается только в отношении таких функциональных блоков, параметры которых сохраняются как значение. Это распространяется на любые параметры блоков с простыми типами данных, а также на входные (Input) и выходные (Output) параметры сложных типов данных.

Если инициализация параметров блоков не производится, то редактор

в качестве начальных значений параметров будет использовать нулевое значение, наименьшее значение или пробел, в зависимости от типа данных. Для параметров типа BLOCK_DB в качестве значения по умолчанию принимается DB1 (DB0 не допускается, так как он не существует).

Если Вы не задаете размер для переменных типа STRING, то компилятор сам установит 254 байтов в качестве максимальной длины и 0 байтов - в качестве текущего значения размера данных, или компилятор использует установки, взятые с вкладки "Compiler" ("Компилятор"), которая открывается посредством выбора опций меню: *Options -> Customize (Опции -> Установки пользователя).*

29.2.7 Формальные параметры

Формальные параметры используются для адресации параметров в программе блока. Формальные параметры имеют такие же имена, как и параметры блока, и используются в выражения программы в качестве операндов.

Формальные параметры простых типов данных

Вы можете использовать формальные параметры простых типов данных в любых выражениях в качестве операндов, имеющих такой же тип данных; Вы можете также передавать их значения в параметры вызываемых блоков.

Вы можете назначать несколько видов типа данных для параметров блока простых типов и таким путем организовать доступ к ним для различных формальных параметров.

Формальные параметры сложных типов данных и типов данных пользователя (UDT)

Вы можете использовать формальные параметры сложных типов данных и пользовательских типов данных в выражениях присвоения (assignment) в качестве операндов, имеющих такой же тип данных; Вы можете передавать их значения в параметры вызываемых блоков. Вы можете также работать с отдельными компонентами сложных типов данных ARRAY, STRUCT и UDT.

Вы можете назначать несколько видов типа данных для параметров блока сложных типов и таким путем организовать доступ к ним для различных формальных параметров. Такая возможность может быть особенно полезной для типов данных DT и STRING, обработать отдельные байты которых Вам не удастся другим способом.

Формальные параметры параметрических типов TIMER и COUNTER

Формальные параметры параметрических типов TIMER и COUNTER могут быть обработаны с использованием SIMATIC-функций таймеров и SIMATIC-функций счетчиков (см. раздел 30.1 "Функции таймеров" и раздел 30.2 "Функции счетчиков"). Значения формальных параметров таких типов могут быть переданы в параметры вызываемых блоков.

Формальные параметры параметрического типа BLOCK_xx

С помощью формальных параметров типа BLOCK_xx Вы можете получить доступ к адресам данных в блоке данных (см. раздел 27.2.3 "Косвенная адресация на SCL").

Формальные параметры параметрических типов BLOCK_FB и BLOCK_FC при использовании языка программирования SCL могут только передаваться в вызываемые блоки (нет команд для обработки формальных параметров такого типа в блоке).

Формальные параметры типов данных POINTER и ANY

Формальные параметры типов данных POINTER и ANY могут быть переданы целиком в вызываемые блоки при использовании языка программирования SCL. Исключение составляют фактические параметры, размещенные во временных локальных данных, передача которых не допускается.

Вы можете назначать несколько видов типа данных для параметров блока типов данных POINTER и ANY и таким путем организовать доступ к ним для различных формальных параметров. Такая возможность может быть особенно полезной для типов данных ANY для того, чтобы модифицировать ANY-указатель в процессе выполнения программы.

29.3 Вызов SCL-блоков

При программировании блоков на языке программировании SCL различают блоки с функциональным значением (function value) и блоки без функционального значения.

Функциональные блоки FB и функции FC без функционального значения являются просто "ветвями" программы (имеют смысл подпрограмм); к этой же группе блоков можно отнести системные функциональные блоки SFB и системные функции SFC без функционального значения.

Функции FC с функциональным значением могут использоваться в выражениях присваивания, а также в других выражениях в качестве операндов. В таблице 29.1 представлен обзор способов вызовов блоков.

Системные функциональные блоки SFB вызываются точно так же, как и функциональные блоки FB, а системные функции SFC вызываются точно так же, как функции FC. Если Вы вызываете системные функциональные блоки SFB с блоком данных, то блок данных размещается в пользовательской программе.

При вызове блока с параметрами параметры блока инициализируются *фактическими параметрами.* Фактические параметры - это такие значения констант, переменных или выражений, с которыми вызванный блок обрабатывается при выполнении программы, и в которых сохраняются результаты этой обработки.

Все параметры блока должны быть инициализированы при вызове функций FC и системных функций SFC.

При вызове функциональных блоков FB и системных функциональных блоков SFB инициализация параметров блока не обязательна. Выходные

параметры при вызове функциональных блоков FB и системных функциональных блоков SFB инициализируются посредством прямого обращения к экземплярным данным вместо использования фактических параметров при вызове.

Таблица 29.1 Вызовы SCL-блоков

Вызов функции	
с функциональным значением	без функционального значения
переменная := FCx(...); переменная := FC_*name*(...);	FCx(...); FC_*name*(...);
Вызов функционального блока	
с блоком данных	как локальный экземпляр
FBx.DBx(...); FB_*name*.DB_*name*(...);	local_*name*(...);

29.3.1 Функции FC без функционального значения

Пример вызова функции FC без функционального значения:

```
FC291(MAX := Maximum,
       IN := InputValue,
      MIN := Minimum,
      OUT := Result);
```

При вызове используется либо абсолютный, либо символьный адрес блока, за которым в скобках следует список параметров.

Все параметры должны быть инициализированы, при этом порядок их следования может быть произвольным. Скобки должны быть указаны, даже если функция FC не имеет параметров.

Если функция имеет единственный входной параметр, то имя параметра при инициализации может быть опущено.

Пример:

Пусть, необходимо запрограммировать преобразование INT-переменной Speed в STRING-переменную Display:

```
Display := I_STRING(Speed);
```

29.3.2 Функции FC с функциональным значением

Пример вызова функции FC с функциональным значением:

```
Result := FC292(
   MAX := Maximum,
    IN := InputValue,
   MIN := Minimum);
```

Функции FC с функциональным значением могут использоваться в любых выражениях в качестве операндов с таким же типом данных, например, в выражениях присваивания. В данном примере глобальной переменной *Result* присваивается функциональное значение функции FC 292.

При вызове используется либо абсолютный, либо символьный адрес блока, за которым в скобках следует список параметров.

Все параметры должны быть инициализированы, при этом порядок их следования может быть произвольным. Скобки должны быть указаны, даже если функция FC не имеет параметров.

Если функция имеет единственный входной параметр, то имя параметра при инициализации может быть опущено.

Если Вы используете при вызове блока входной параметр EN, и если этот вход имеет значение FALSE (ЛОЖЬ), то функциональное значение не будет определено (функциональному значению не будет присвоено никакой величины).

29.3.3 Функциональный блок со своим собственным блоком данных

Экземплярный блок данных специфицируется, когда производится вызов функционального блока. Он может быть либо запрограммирован в исходной программе (после функционального блока и перед его вызовом), либо сгенерирован в среде программирования на SCL, если он еще существует. Экземплярный блок данных может быть запрограммирован на SCL также и инкрементным способом, без исходной программы, (см. раздел 3.6.1 "Инкрементное программирование блоков данных").

Любой свободный блок данных может использоваться как экземплярный блок данных. При этом пользователь свободен в выборе символьного имени в допустимых пределах.

```
DATA_BLOCK DB291
   FB291
BEGIN
END_DATA_BLOCK
```

Пример вызова функционального блока с экземплярным блоком данных:

```
FB291.DB291(IN := InputValue);
Result := DB291.OUT;
```

Вызов записывается как адрес функционального блока с последующим адресом экземплярного блока, отделенные точкой, и далее - список параметров в скобках. В качестве адресов могут использоваться либо абсолютный, либо символьный адрес блока.

Инициализация параметров функционального блока необязательна. Так как входные/выходные параметры сложных типов сохраняются как указатели, они должны быть инициализированы значащими величинами при первом вызове функционального блока. Если параметр блока не инициализирован, то он сохраняет свое последнее определенное значение. Скобки должны присутствовать в записи, даже если не инициализированы никакие параметры.

Все параметры могут быть адресованы также как глобальные данные с указанием имени экземплярного блока данных и имени параметра. В примере граничные значения определены константами. Они также могут быть инициализированы до вызова функционального блока посредством операторов присваивания:

```
DB291.MAX := Maximum;
DB291.MIN := Minimum;
```

Выходные параметры не могут быть инициализированы при вызове функционального блока. Если требуется, их значения считываются непосредственно из экземплярного блока данных и в дальнейшем обрабатываются без промежуточного хранения.

```
IF DB291.OUT > 10000 THEN ... END_IF;
DB291.MIN := Minimum;
```

29.3.4 Функциональный блок как локальный экземпляр

Функциональные блоки могут быть объявлены как "локальные экземпляры" (local instance) и вызваны в другом функциональном блоке. Такие функциональные блоки (вызываемые) сохраняют свои локальные данные в экземплярном блоке данных вызывающего функционального блока.

```
FUNCTION_BLOCK FB290
...
VAR
  Delimiter : FB291;
END_VAR
...
BEGIN
Delimiter (IN := InputValue);
Result := Delimiter.OUT;
...
END_FUNCTION_BLOCK
```

Объявление экземпляров выполняется в статических локальных данных; здесь Вы назначаете имя (например, Delimiter) и назначаете функциональный блок (FB291 или его символьное имя) как данных. К моменту компилирования функциональный блок, который должен вызываться, должен уже существовать или в виде ранее скомпилированного блока в разделе *Blocks (Блоки)*, или в виде (заведомо безошибочной) исходной программы, которая компилируется перед вызовом блока.

Точно такие же действия выполняются при вызове системного функционального блока SFB как локального экземпляра.

Вызов блока в виде локального экземпляра производится как инициализация имени переменной с последующим списком параметров в скобках. Инициализация параметров функционального блока необязательна.

Так как входные/выходные параметры сложных типов сохраняются как указатели, они должны быть инициализированы значащими величинами при первом вызове функционального блока. Если параметр блока не инициализирован, то он сохраняет свое последнее определенное значение. Скобки должны присутствовать в записи, даже если не инициализированы никакие параметры.

Пользователь может создать несколько локальных экземпляров с различными именами для одного и того же функционального блока.

Все параметры локального экземпляра могут быть адресованы также как компоненты структурированной переменной - в записи указываются имя локального экземпляра и имя параметра. В примере граничные значения определены константами. Они также могут быть инициализированы перед вызовом локального экземпляра посредством операторов присваивания:

```
Delimiter.MAX := Maximum;
Delimiter.MIN := Minimum;
```

Выходные параметры не могут быть инициализированы при вызове функционального блока (это также касается локальных экземпляров). Если требуется, их значения считываются как компоненты локального экземпляра:

```
Result := Delimiter.OUT;
```

Вы можете также получить доступ к параметрам локального экземпляра "со стороны" вызывающего функционального блока. Такой доступ организуется как доступ к адресам глобальных данных посредством записи спецификации блока данных (DB290), локального экземпляра (Delimiter) и имени параметра:

```
DB290.Delimiter.MAX := Maximum;
DB290.Delimiter.MIN := Minimum;
Result := DB290.Delimiter.OUT;
```

29.3.5 Фактические параметры

При вызове блока происходит инициализация параметров блока текущими значениями (фактическими параметрами - "actual parameter") с помощью операции присваивания (см. предыдущий раздел). В языках программирования STL и SCL в отношении фактических параметров применяются одинаковые операции (см. раздел 19.3 "Фактические параметры"), за исключением следующих моментов:

- Параметры блока сложных типов данных DT и STRING в SCL могут быть инициализированы значениями констант.
- Параметры блока типов POINTER в SCL не могут быть инициализированы значениями констант или с помощью указателя в формате P#Operand. Исключение: определение значения по умолчанию посредством "нулевого" указателя (NIL) разрешается.
- Параметры блока типов ANY не могут быть инициализированы значениями констант или с помощью ANY-указателя в форме P#(DB.(Операнд Тип Размер)). Исключение: определение значения по умолчанию посредством "нулевого" указателя (NIL) разрешается.

- Вы можете инициализировать параметры блока с помощью выражений, которые используют значение такого же типа данных как параметр блока. Например, функция FC с функциональным значением также может быть фактическим параметром.

Примечание:

Если Вы инициализируете формальный параметр типа POINTER или ANY временной переменной при вызове FB или FC, Вы не можете передавать этот параметр из вызванного блока в другой блок. Адреса временных переменных теряют свои значения при передаче в другой блок.

29.4 Механизм EN/ENO

При программировании на SCL пользователь имеет возможность проверить отдельные выражения (операции) на правильность выполнения, например, имеется возможность проверить, находится ли результат вычисления функции в допустимом численном диапазоне? Результат подобных проверок сохраняется в так называемой "OK-переменной". Пользователь может также связать назначение "OK-переменной" с вызывающим блоком посредством выходного параметра ENO блока. Наконец, имеется возможность выполнять вызов блока посредством EN в зависимости от условий.

Вы можете использовать заранее определенные переменные EN и ENO для всех блоков (FC, SFC, FB, SFB, а также для IEC-функций), для всех стандартных функций (например, функции сдвига и функции преобразования), кроме функций таймеров и функций счетчиков.

В главе 15 "Биты состояния", в разделе 15.4 "Использование двоичного результата" описывается, как используется механизм EN/ENO в стандартных языках программирования.

29.4.1 OK-переменная

В языке программирования SCL существует инициализируемая переменная с именем "OK" и типом данных BOOL. Эта переменная сообщает об ошибках, возникающих при выполнении программы в SCL-блоке, но только в случае, если Вы выбрали опцию "Set OK flag" ("Установить OK-флаг") на вкладке "Compiler" ("Компилятор"), которая открывается при выборе опций меню: *Options -> Customize (Опции -> Установки пользователя)* в редакторе SCL-программ.

Редактор или компилятор не проверяют, установлена данная опция или нет, если Вы используете OK-переменную в программе.

В начале блока OK-переменная имеет значение TRUE (ИСТИНА). При возникновении программной ошибки OK-переменная сбрасывается в состояние FALSE (ЛОЖЬ). Вы можете проверять OK-переменную с помощью соответствующих SCL-операций, а также можете присваивать этой переменной требуемые значения в любое время.

```
SUM := SUM + IN;
IF OK
     THEN (* не произошло ошибок *);
     ELSE (* ошибка в операции сложения *);
END_IF;
```

В данном примере на OK-переменную влияет результат обработки арифметического выражения и некоторых функций преобразования (см. раздел 30.5.2 "Явные функции преобразования"). Если происходит ошибка при выполнении стандартных функций, таких, как математические функции, об этом сообщается посредством выхода ENO (см. далее по тексту).

При выходе их блока значение OK-переменной назначается выходу ENO.

29.4.2 Выход ENO (ENO output)

Вызываемый блок сохраняет значение OK-переменной в выходе ENO ("ENO output" = "Enable output" = "Выход разблокирован"). ENO имеет тип данных BOOL. После вызова блока значение ENO может быть использовано для определения того, правильно ли был обработан блок (в этом случае ENO = TRUE [ИСТИНА]) или произошла ошибка (в этом случае ENO = FALSE [ЛОЖЬ]).

```
FC15 (In1:= ..., In2:= ...);
IF ENO
     THEN (* не произошло ошибок *);
     ELSE (* произошла ошибка *);
END_IF;
```

Если необходимо с помощью ENO после вызова блока передать в вызывающий блок сообщение о групповой ошибке, то установите соответствующим образом OK-переменную:

```
FC15 (In1:= ..., In2:= ...);
OK := ENO;
```

Вы можете также назначить значение для выхода ENO в блоке с помощью соответствующей установки OK-переменной:

```
IF (* произошла ошибка *)
   THEN OK := FALSE; RETURN;
END_IF;
```

ENO является не параметром блока, а последовательностью выражений, генерируемой редактором программ, в случае использования ENO. ENO не объявляется. ENO может быть немедленно проверен после вызова блока.

Если Вы управляете вызовом блока с помощью входа EN (см. следующий раздел), и вход EN имеет значение FALSE (ЛОЖЬ), так что блок не может быть запущен для выполнения, то выход ENO также будет иметь значение FALSE (ЛОЖЬ).

Примечание:

Если в блоке, созданном с использованием стандартных языков программирования, для сообщения об ошибках используется "двоичный результат" BR, то в SCL Вы можете проверять сообщения об ошибках с помощью выхода ENO после вызова блока (см. раздел 15.4 "Использование двоичного результата").

29.4.3 Вход EN (EN input)

Вы можете управлять вызовами блока с помощью входа EN. EN имеет тип данных BOOL. Если вход EN инициализирован значением TRUE (ИСТИНА), то вызываемый блок будет вызван для выполнения. Если EN инициализирован значением FALSE (ЛОЖЬ), то вызываемый блок соответственно не будет вызван для выполнения. При этом будет выполнен переход к следующему оператору после вызова блока.

```
FC15 (EN := I1.0,
      In1:= ...,
      In2:= ...);
(* FC15 выполняется только в случае, когда I1.0 = "1" *)
```

Если Вы не используете EN, то блок будет выполняться всегда.

EN является не параметром блока, а последовательностью выражений, генерируемой редактором программ, в случае использования EN. EN не объявляется. Вы должны использовать EN в списке параметров блока также, как любой другой входной параметр.

Вы можете также назначить значение для входа EN значением ENO; в этом случае вызываемый блок будет обрабатываться только в том случае, если обработка ранее вызванного блока успешно завершена.

Пример:

Если необходимо вызывать блок FC16 только в случае, если обработка FC15 успешно завершена, то фрагмент программы может выглядеть следующим образом:

```
FC15 (EN := I1.0,
      In1:= ...,
      In2:= ...);
FC16 (EN := ENO,
      In1:= ...,
      In2:= ...);
```

Если ранее никакой блок не вызывался на том же уровне вызовов, то ENO будет иметь значение TRUE (ИСТИНА).

Необходимо отметить, что функция FC или системная функция SFC принимает неопределенное значение (произвольное значение функции), если Вы управляете вызовом функции с помощью входа EN, и вход EN имеет значение FALSE (ЛОЖЬ).

30 SCL-функции

30.1 Функции таймеров

Таймеры в системной памяти CPU в языке программирования SCL адресуются так же, как функции с возвращаемым (функциональным) значением. Названия функций, соответствующих различным режимам таймера, представлены ниже:

- S_PULSE ("pulse timer" - "режим управляемого импульса")
- S_PEXT ("extended pulse" - "режим расширенного импульса")
- S_ODT ("ON delay" - "с задержкой включения")
- S_ODTS ("latching ON delay" - "с задержкой включения с памятью")
- S_OFFDT ("OFF delay" - "с задержкой выключения")

Все функции таймера имеют параметры, показанные в таблице 30.1.

Таблица 30.1 Параметры для SIMATIC-функций таймеров

Параметр	Объявление	Тип данных	Значение
T_NO	INPUT	TIMER	Адрес таймера
S	INPUT	BOOL	Параметр запуска таймера
TV	INPUT	S5TIME	Установленное значение таймера
R	INPUT	BOOL	Параметр сброса таймера
Функциональное значение	OUTPUT	S5TIME	Текущее значение времени в двоично-десятичном коде (BCD)
Q	OUTPUT	BOOL	Состояние таймера
BI	OUTPUT	WORD	Текущее значение времени в двоичном коде (binary)

Пример вызова функций таймеров:

```
Time_BCD  := S_PULSE(
     T_NO := Timer_address,
     S    := Start_input,
     TV   := Timer_duration,
     R    := Reset,
     Q    := Timer_status,
     BI   := Binary_time);
```

Диаграммы работы функций таймеров подробно рассмотрены в главе 7 "Функции таймеров (Timer Functions)".

Необходимо отметить, что функция разблокирования таймера (см. гл. 7 "Функции таймеров (Timer Functions)" не поддерживается в SCL.

При инициализации параметров функций таймеров применяются следующие правила:

- параметр T_NO должен быть всегда инициализирован
- пара параметров S и TV могут быть не инициализированы
- параметр Q может быть не инициализирован
- параметр BI может быть не инициализирован

В дополнение к SIMATIC-функциям таймеров в специализированных CPU поддерживаются также IEC-функции таймеров как системные функциональные блоки SFB:

- SFB 3 TP
  Pulse generation - функция генерации импульса
- SFB 4 TON
  ON delay - функция генерации импульса с задержкой включения
- SFB 5 TOF
  OFF delay - функция генерации импульса с задержкой выключения

Данные функции описаны в разделе 7.7 "IEC-функции таймеров (IEC-Timer Functions)".

Функциональные блоки хранятся в "стандартной библиотеке" *Standard library* в разделе *System Function Blocks (Системные функциональные блоки).*

Примеры применения SIMATIC-функций таймеров и IEC-функций таймеров Вы можете найти на дискете, прилагаемой к книге, в библиотеке "SCL_Book" в исходном файле "Timer Functions" ("Функции таймеров") в разделе "30 SCL Functions" ("30 SCL-функции").

30.2 Функции счетчиков

Счетчики в системной памяти CPU в языке программирования SCL адресуются так же, как функции с возвращаемым (функциональным) значением. Названия функций, соответствующих различным режимам счетчика, представлены ниже:

- S_CU ("up counter" - "функция счетчика прямого счета")
- S_CD ("down counter" - "функция счетчика обратного счета")
- S_CUD ("up-down counter" - "функция счетчика прямого и обратного счета")

Параметры для всех SIMATIC-функций счетчиков представлены в таблице 30.2.

Таблица 30.2 Параметры для SIMATIC-функций счетчиков

Параметр	Объявление	Тип данных	Значение
C_NO	INPUT	COUNTER	Адрес счетчика
CU	INPUT	BOOL	Прямой счет
CD	INPUT	BOOL	Обратный счет
S	INPUT	BOOL	Параметр запуска счетчика
PV	INPUT	S5TIME	Установленное значение счетчика
R	INPUT	BOOL	Параметр сброса счетчика
Функциональное значение	OUTPUT	WORD	Текущее значение счетчика в двоично-десятичном коде (BCD)
Q	OUTPUT	BOOL	Состояние счетчика
CV	OUTPUT	WORD	Текущее значение счетчика в двоичном коде (binary)

Пример вызова функций счетчиков:

```
BCD_Count_Value  := S_CU(
     C_NO := Count_address,
     CU   := Count_up,
     S    := Set_input,
     PV   := Count_value,
     R    := Reset,
     Q    := Counter_status,
     BI   := Binary_count_value);
```

Диаграммы работы функций счетчиков подробно рассмотрены в главе 8 "Функции счетчиков (Counter Functions)".

Необходимо отметить, что функция разблокирования счетчиков не поддерживается в SCL.

При инициализации параметров функций счетчиков применяются следующие правила:

- применение параметра CD не допускается вместе с функцией счетчика S_CU
- применение параметра CU не допускается вместе с функцией счетчика S_CD
- параметр C_NO должен быть всегда инициализирован
- в зависимости от выбранного режима счетчика должен быть установлен один из параметров CD или CU
- пара параметров S и PV могут быть не инициализированы
- параметр Q может быть не инициализирован
- параметр CV может быть не инициализирован

В качестве константы для инициализации параметра PV счетчика

может быть применено целое число типа INT с диапазоном значений от 0 до 999 или шестнадцатеричное число с диапазоном значений от 16#000 до 16#3E7.

В дополнение к SIMATIC-функциям счетчиков в специализированных CPU поддерживаются также IEC-функции счетчиков как системные функциональные блоки SFB:

- SFB 0 CTU
  ("up counter" - "функция счетчика прямого счета")
- SFB 1 CTD
  ("down counter" - "функция счетчика обратного счета")
- SFB 2 CTUD
  ("up-down counter" - "функция счетчика прямого и обратного счета")

Данные функции описаны в разделе 8.6 "IEC-функции счетчиков (IEC-Counter Functions)". Функциональные блоки хранятся в "стандартной библиотеке" *Standard library* в разделе *System Function Blocks (Системные функциональные блоки).*

Примеры применения SIMATIC-функций счетчиков и IEC-функций счетчиков Вы можете найти на дискете, прилагаемой к книге, в библиотеке "SCL_Book" в исходном файле "Counter Functions" ("Функции счетчиков") в разделе "30 SCL Functions" ("30 SCL-функции").

30.3 Математические функции

В языке программирования SCL поддерживаются следующие математические функции:

- Тригонометрические функции:

SIN	функция синуса
COS	функция косинуса
TAN	функция тангенса

- Обратные тригонометрические функции (Arc-функции):

ASIN	функция арксинуса
ACOS	функция арккосинуса
ATAN	функция арктангенса

- Логарифмические функции:

EXP	экспоненциальная функция по основанию *e*
EXPD	экспоненциальная функция по основанию 10
LN	натуральный логарифм
LOG	десятичный логарифм

- Тригонометрические функции:

ABS	функция выделения абсолютного значения
SQR	функция нахождения квадрата числа
SQRT	функция взятия квадратного корня

Математические числа обрабатывают числа форматов INT, DINT и REAL.

При использовании в функции числа в формате INT или DINT в качестве входного параметра, оно автоматически преобразуется в число в формате REAL.

Математические функции работают с числами, внутренне представляемыми в формате REAL; результат также получается в формате REAL. Исключение составляет функция ABS, которая выдает результат того же типа, к которому относилось исходное число.

Тригонометрические функции рассматривают входные параметры, как углы, выраженные в радианах, в диапазоне от 0 до 2π (где π = 3,141593e+00), что соответствует диапазону углов от 0° до 360°.

Функция	Допустимый диапазон	Возвращаемое значение
ASIN	-1 ... +1	-π/2 ... +π/2
ACOS	-1 ... +1	0 ... π
ATAN	нет ограничений	-π/2 ... +π/2

Примеры:

Расчет реактивной мощности Reactive_power, которая определяется произведением напряжения Voltage на ток Current и на синус фазового сдвига между ними:

```
Reactive_power  := Voltage * Current * SIN(φ);
```

Расчет объема жидкости Volume, который определяется произведением π числа PI на квадрат радиуса основания сосуда Radius и на уровень заполнения сосуда Level:

```
Volume  := PI * SQR(Radius) * Level;
```

Расчет длины гипотенузы по известным величинам катетов:

```
c  := SQRT(SQR(a) + SQR(b));
```

30.4 Функции сдвига (Shifting) и циклического сдвига (Rotating)

Общий для функций сдвига (Shifting) и циклического сдвига (Rotating) способ вызова имеет вид:

```
Result  := Function(
    IN  := Input_value,
     N  := Shift_number);
```

Функции сдвига и циклического сдвига имеют два входных параметра. Параметр N показывает число битов, на которое необходимо произвести операцию сдвига или циклического сдвига, этот параметр относится к типу INT. Параметр IN определяет переменную, с которой необходимо выполнить операцию сдвига или циклического сдвига, этот параметр относится к классу типов ANY_BIT (то есть, к типам BOOL, BYTE, WORD,

DWORD). При этом значение функции относится к тому же типу, что и входное значение.

Примеры:

```
MW14  := SHL(IN := MW12, N := 2);

res_dword := ROR(
       IN := in_dword,
        N := shift_int);
```

Таблица 30.3 Функции сдвига (Shifting) и циклического сдвига (Rotating)

SHL	Сдвиг влево	Входное значение IN сдвигается влево на N битов; освободившиеся позиции заполняются нулями.
SHR	Сдвиг вправо	Входное значение IN сдвигается вправо на N битов; освободившиеся позиции заполняются нулями.
ROL	Циклический сдвиг влево	Входное значение IN сдвигается влево на N битов; освободившиеся позиции заполняются значениями "вытолкнутых" из аккумулятора битов.
ROR	Циклический сдвиг вправо	Входное значение IN сдвигается вправо на N битов; освободившиеся позиции заполняются значениями "вытолкнутых" из аккумулятора битов.

30.5 Функции преобразования (Conversion Functions)

При совместной обработке переменных с помощью операторов переменные должны (внутренне) относиться к одному типу данных. Это касается и инструкций присваивания, и операций инициализации параметров функций и параметров блоков. Если переменная не может быть обработана, так как принадлежит к другому типу данных, то ее тип должен быть изменен. Для этой цели применяются функции преобразования типов данных (Conversion Functions).

В языке программирования SCL поддерживаются два типа функций преобразования. Первый тип функций преобразования относится к классу A ("Class A"). Эти функции выполняются в SCL автоматически ("неявно"), так как они не связаны с потерей информации (например, преобразование данных типа BYTE в данные типа WORD). Второй тип функций преобразования относится к классу B ("Class B"). Эти функции пользователь должен инициировать самостоятельно, "явно", (например, преобразование данных типа REAL в данные типа INT). Любая потеря информации может быть предупреждена с помощью соответствующего контроля данных или же пользователь может проверять "OK-переменную" для проверки результатов выполнения операций (возможность задействования OK-переменной должна быть инициирована в *Compiler Properties [Свойствах компилятора]*).

Пользователь имеет возможность преобразовывать и обрабатывать переменные типов DATE_AND_TIME и STRING с помощью IEC-функций. Описание стандартной библиотеки *Standard Library* и раздела *IEC*

Function Blocks [IEC функциональные блоки]) Вы можете найти в главе 31 "IEC-функции".

30.5.1 Неявные функции преобразования (Implicit Conversion Functions)

Неявные функции преобразования типов данных выполняются в SCL автоматически ("неявно"). Тем не менее, Вы можете запрограммировать эти функции преобразования типов, например, для того, чтобы улучшить ясность или читаемость Вашей программы.

В таблице 30.4 показаны поддерживаемые в языке программирования SCL неявные функции преобразования типов данных.

Таблица 30.4 Неявные функции преобразования (Implicit Conversion Functions)

Функция	OK	Особенности операции преобразования
BOOL_TO_BYTE	N	Заполнение нулями незаполненных позиций слева
BOOL_TO_WORD	N	
BOOL_TO_DWORD	N	
BYTE_TO_WORD	N	
BYTE_TO_DWORD	N	
WORD_TO_DWORD	N	
INT_TO_DINT	N	Заполнение знаком незаполненных позиций слева
INT_TO_REAL	N	-
DINT_TO_REAL	N	При преобразовании, помимо прочего, теряется точность
CHAR_TO_STRING	Y	Преобразование в строку символов, состоящую из одного символа

При преобразовании данных из символьной переменной в строку символов (функция CHAR_TO_STRING), создается строка STRING, длина которой равна 1 (строка состоит из одного символа), и OK-переменная устанавливается в состояние FALSE (ЛОЖЬ).

Примеры:

```
                    //Преобразование типов данных
MB10       := M7.0;       //BOOL -> BYTE
real_var   := int_var;    //INT  -> REAL
string_var := char_var;   //CHAR -> STRING
```

В рассмотренном примере меркер M10.7 принимает значение меркера M7.0. Остальные биты устанавливаются в состояние "0".

30.5.2 Явные функции преобразования (Explicit Conversion Functions)

Пользователь должен самостоятельно инициировать в программе явные функции преобразования типов данных; тем не менее, при использовании некоторых из этих явных функций собственно преобразование данных не происходит и не выполняется никакой код (см. табл. 30.5 с комментарием "принимается без изменения"). На состояние переменной OK влияют некоторые из функций преобразования.

Примеры:

```
MB10      := CHAR_TO_BYTE(char_var);
int_var   := WORD_TO_INT(MW20);
real_var  := DWORD_TO_REAL(MD30);
```

Необходимо отметить, что в последнем примере преобразование данных не происходит. Битовая структура двойного слова меркеров принимается без изменений в переменную типа REAL.

В случае преобразования вида REAL_TO_xxx переменная OK устанавливается в состояние FALSE (ЛОЖЬ), даже в случае, если корректное число типа REAL недоступно.

30.6 Программирование Ваших собственных функций на SCL

Если Вы не можете найти подходящих функций среди набора стандартных SCL-функций и IEC-функций, язык SCL позволяет Вам создать свои собственные функции, которые Вы можете адаптировать к Вашим требованиям.

Подходящий тип блоков для этих целей - это функции FC с функциональным значением. Вопросы программирования функций FC с функциональным значением и вопросы по организации вызовов этих функций рассматриваются в разделе 29.2.2 "Функции FC с функциональным значением" и в разделе 29.3.2 "Функции FC с функциональным значением" соответственно.

Во многих случаях языковых ресурсов SCL не хватает для программирования требуемых функций. В таких случаях для создания пользовательских функций возможно также использование языка программирования STL (см. раздел 30.7 "Программирование Ваших собственных функций на STL"). Однако, принципиальная возможность использования видов типов данных ("data type views") в SCL позволяет обрабатывать сложные переменные. В разделе 27.1.9 "Виды типа данных (Data Type Views)" рассмотрено, какие виды типа данных для каких переменных Вы можете использовать.

Поразрядная обработка переменных простых видов

Например, Вам необходимо обработать отдельные биты переменной формата двойного слова (doubleword) некоторым образом, например, методом опроса битов или логического комбинирования с последующей

Таблица 30.4 Явные функции преобразования (Implicit Conversion Functions)

Функция	Преобразование	OK	Примечания
BYTE_TO_BOOL	Принимается младший значащий бит	Y	Если бит в не принятой части переменной = "1", то OK= TRUE (ИСТИНА)
WORD_TO_BOOL		Y	
DWORD_TO_BOOL		Y	
WORD_TO_BYTE	Принимается младший значащий байт	Y	
DWORD_TO_BYTE		Y	
DWORD_TO_WORD	Принимается младшее значащее слово	Y	
CHAR_TO_BYTE	При присваивании не изменяется	N	
BYTE_TO_CHAR	При присваивании не изменяется	N	
CHAR_TO_INT	Старший значащий байт заполняется нулями	N	
INT_TO_CHAR	Младший значащий байт принимается без изменения	Y	Если бит в левом байте = "1", то OK = TRUE (ИСТИНА)
STRING_TO_CHAR	Принимается первый символ	Y	Если длина переменной STRING <> "1", то OK = FALSE (ЛОЖЬ)
WORD_TO_INT	Принимается без изменения	N	
DWORD_TO_DINT		N	
INT_TO_WORD		N	
DINT_TO_DWORD		N	
REAL_TO_DWORD		N	Нет преобразования!
DWORD_TO_REAL		N	Нет преобразования!
DINT_TO_INT	Копируются биты для знака	Y	Если численный диапазон нарушен, то OK = FALSE (ЛОЖЬ)
REAL_TO_INT	Округляется до целого INT	Y	
REAL_TO_DINT	Округляется до целого DINT	Y	
ROUND	Преобразование REAL в DINT с округлением	Y	Если численный диапазон нарушен, то OK = FALSE (ЛОЖЬ)
TRUNC	Преобразование REAL в DINT без округления ("усечение" дробной части)	Y	
DINT_TO_TIME	Принимается без изменения	N	
DINT_TO_TOD		Y	Если нарушен диапазон для TOD, то OK = FALSE (ЛОЖЬ)
DINT_TO_DATE		Y	Если левое слово имеет назначение, то OK = FALSE (ЛОЖЬ)
DATE_TO_DINT		N	
TIME_TO_DINT		N	
TOD_TO_DINT		N	
WORD_TO_BLOCK_DB	Принимается без изменения	N	
BLOCK_DB_TO_WORD	Принимается без изменения	N	

записью в другой бит. Для этой цели Вы можете применить виды типа данных к переменной в форме массива битов с последующей адресацией отдельных битов как элементов этого массива.

```
VAR_TEMP
  DW_VAR   :  DWORD;
  Pattern AT DW_VAR : ARRAY[0..31] OF BOOL;
END_VAR
...
Pattern[1] := Pattern[10]&Pattern[11]
...
```

В данном маленьком примере 10-й и 11-й биты переменной DW_VAR комбинируются в логической операции AND (И), и результат записывается в 1-й бит той же переменной.

Обработка переменных типов DT и STRING

Переменные типов DT и STRING в языке программирования SCL обычно обрабатываются "целиком", например, при инициализации параметров функции или при пересылке данных из одного параметра в другой. Вы можете использовать IEC-функции из стандартной библиотеки STEP 7 Standard Library для обработки переменных типов DT или STRING.

При необходимости обработки части переменной типа DT или STRING посредством SCL-инструкций используйте виды типа данных (Data Type View) для переменной, которую нужно обработать. Байтовые массивы (OF BYTE) подходят для представления переменных типов DT и STRING (см. табл. 30.6).

Пример SCL-функции

Рассмотрим пример функции "Hour" ("Час"), которая извлекает информацию о часе из данных в формате DT и передает эту информацию в возвращаемое значение функции Hour.

```
FUNCTION Hour : INT
VAR_INPUT
  DAT : DT;
  TMP AT DAT : ARRAY[1..8] OF BYTE;
END_VAR
BEGIN
Hour :=
  WORD_TO_INT(SHR(IN:=TMP[4],N:=4))*10 +
  WORD_TO_INT(TMP[4] AND 16#0F);
END_FUNCTION
(* Считывание данных о времени из CPU и
вызов функции "Hour" *)
SFC_ERROR := READ_CLK(DATE_TIME);
IF Hour (DATE_TIME) >= 18
  THEN FINISH_WORK := TRUE;
END_IF;
```

Таблица 30.6 Часто используемые виды типа данных (Data Type View)

Тип переменной	Виды типа данных (Data Type View)	Особенности операции преобразования
Простой	Массив элементов типа BOOL	`TEMPVAR : DWORD;` `VEIW AT TEMPVAR : ARRAY[0..31] OF BOOL;`
DT	Массив элементов типа BYTE	`TEMPVAR : DT;` `VEIW AT TEMPVAR : ARRAY[1..8] OF BYTE;`
STRING	Массив элементов типа CHAR	`TEMPVAR : STRING[max];` `VEIW AT TEMPVAR : ARRAY[1..max] OF CHAR;`

ARRAY	STRUCT	`TEMPVAR : ARRAY[0..255] OF BYTE;`
		`VEIW1 AT TEMPVAR : STRUCT`
		`name : data_type;`
		`.... : ...`
		`END_STRUCT;`
		`VEIW2 AT TEMPVAR : STRUCT`
		`name : data_type;`
		`.... : ...`
		`END_STRUCT;`

ANY	STRUCT	`TEMPVAR : ANY;`
		`VEIW1 AT TEMPVAR : STRUCT`
		`ID : BYTE;`
		`TYP : BYTE;`
		`ANZ : INT;`
		`DBN : INT;`
		`PTR : DWORD;`
		`END_STRUCT;`

Различные виды массивов и структур

Переменным типов ARRAY и STRUCT могут быть назначены виды типа данных (Data Type View), сами имеющие соответственно типы ARRAY и STRUCT. Применение таких видов типа данных может найти себе место при создании области "почтового ящика" ("mailbox") для передачи или приема фреймов сообщений.

Например, Вы можете установить максимальную длину области "почтового ящика" ("mailbox"), используя байтовый массив. Для каждого фрейма сообщения, который необходимо обработать в "почтовом ящике" ("mailbox"), Вы можете применить для работы с "почтовым ящиком" вид типа данных, который имеет структуру фрейма сообщения. Такой вид типа данных должен быть специально предназначен и приспособлен для фрейма сообщения, следовательно, он может иметь размер, меньший, чем размер области "почтового ящика" ("mailbox").

Управление указателями ANY

При создании в области временных локальных данных переменной типа

ANY компилятор интерпретирует эту переменную как указатель и передает ее прямо во входной параметр типа ANY, например, параметр вызываемого блока (см. раздел 29.2.4 "Временные локальные данные").

Вы можете управлять этим ANY-указателем в режиме выполнения программы с помощью видов типа данных (Data Type View), что позволит Вам динамически задавать различные исходные области данных для копирования блоков.

Пример:

Пусть, необходимо выполнить копирование данных из области, которая определяется переменными *DATABLOCK, DATASTART* и *NUM_OF_BYTES* в переменную, которая имеет имя *SEND_MAILBOX.*

```
FUNCTION_BLOCK
VAR_INPUT
  AREA         : ANY;
  DATABLOCK    : INT;
  DATASTART    : INT;
  NUM_OF_BYTES : INT;
END_VAR
VAR_TEMP
  SFC_ERROR    : INT;
  SEND_MAILBOX : ANY;
  VEIW AT SENDEFACH : STRUCT;
     ID  : WORD;
     TYP : BYTE;
     NUM : INT;
     DBN : INT;
     PTR : DWORD;
     END_STRUCT;
END_VAR
BEGIN
VEIW.ID   := 16#10;
VEIW.TYP  := 16#02;
VEIW.NUM  := NUM_OF_BYTES;
VEIW.DBN  := DATABLOCK;
VEIW.PTR  := INT_TO_WORD(8*DATASTART);
SFC_ERROR := BLKMOV(
  SRCBLK  := AREA,
  DSTBLK  := SEND_MAILBOX);
END_FUNCTION_BLOCK
(* Вызов функционального блока *)
COPY.COPYDATA(
  AREA         := SEND_MAILBOX,
  DATABLOCK    := 309,
  DATASTART    := 32,
  NUM_OF_BYTES := 32);
```

Вы можете найти еще несколько примеров по данной теме на прилагаемой дискете в библиотеке SCL_Book в разделе "General Examples" ("Общие примеры").

30.7 Программирование Ваших собственных функций на STL

Использование функций FC с функциональным значением позволяет Вам создавать свои собственные функции с использованием языка программирования SCL. Тем не менее, так как Вы можете совместно использовать в Вашей программе блоки, созданные с использованием различных языков программирования, то Вы можете также создавать функции на языке STL и затем вызывать их из SCL-программы. Это позволяет Вам использовать более обширный набор функций STL, такие, например, как прямой доступ к адресам переменных или адресация с использованием адресного регистра.

Вы можете программировать STL-блоки двумя различными способами: инкрементным или путем написания исходных текстов программ (см. раздел 3.4 "Программирование кодовых блоков на STL").

Для второго способа создания программ - путем написания исходных текстов программ - процедура идентична созданию программ SCL-блоков:

1. Создание исходного STL-файла в разделе исходных файлов *Source Files*.
2. Открытие исходного STL-файла в разделе исходных файлов *Source Files* двойным щелчком кнопки манипулятора "мышь" на объекте.
3. Программирование исходной программы на языке программирования STL (см. замечания ниже по тексту).
4. Если Вы выбрали символьные имена для функций, то заполните таблицу символов *Symbol Table*.
5. Компилирование исходной STL-программы, для того, чтобы в разделе блоков программы *Blocks (Блоки)* были сохранены скомпилированные блоки функций.
6. Теперь Вы можете вызывать новые функции таким же образом, как, например, стандартные функции в SCL-программе.

При написании исходных текстов STL-программ используются почти такие же ключевые слова, как и при программировании блоков на языке SCL (см. таблицу 3.3 в разделе 3.4.3 "Программирование кодовых блоков на STL, ориентированное на создание исходных файлов").

Основное различие, относящееся к функциям с функциональным значениям, заключается в том, что функциональное значение в STL-программе имеет имя RET_VAL (или ret_val). То есть, Вы назначаете значение функции переменной RET_VAL в программе.

В качестве наших небольших примеров мы выбрали функции сканирования, запуска и сброса функций таймера. Целью создания этих функций является упрощение использования функций таймеров. В главе 7 "Функции таймеров (Timer Functions)" рассматривается, как функции таймеров программируются с использованием языка STL.

Функция T_SCAN возвращает состояние указанного в параметре адреса таймера:

```
FUNCTION T_SCAN : BOOL
VAR_INPUT
  T_NO : TIMER;
END_VAR
BEGIN
  U T_NO;  = RET_VAL;
END_FUNCTION
```

Функция T_PULSE запускает таймер в режиме управляемого импульса:

```
FUNCTION T_PULSE : VOID
VAR_INPUT
  T_NO  : TIMER;
  START : BOOL;
  Time_value : S5TIME;
END_VAR
BEGIN
  U Start; L Time_value; SI T_NO;
END_FUNCTION
```

Функция T_RESET сбрасывает таймер при каждом ее вызове:

```
FUNCTION T_RESET : VOID
VAR_INPUT
  T_NO  : TIMER;
END_VAR
BEGIN
  Set; R T_NO;
END_FUNCTION
```

После компиляции эти функции могут быть использованы в SCL-программе, например, следующего вида:

```
IF NOT T_SCAN(T1)
   THEN T_PULSE (T_NO  := T2,
                 START := I1.0,
                 Time_value := S5T#5s);
   ELSE T_RESET(T3);
END_IF;
```

Вы можете найти эти примеры на прилагаемой к книге дискете в библиотеке SCL_Book в разделе "General Examples" ("Общие примеры").

30.8 Краткое описание примеров использования языка SCL

30.8.1 Пример "Conveyor" ("Конвейер")

В примере "Conveyor" ("Конвейер") представлено применение двоичных логических операций, функций установки/сброса и вызовов блоков. Он разработан для языка программирования STL. Если Вы освоили STL, и теперь желаете изучить SCL, то Вы найдете здесь предложения по преобразованию типичных STL-функций в SCL-функции.

На рисунке 30.1 показана программа и структура данных примера. Детальное описание примеров дано в разделах:

- 5.5 "Пример системы управления ленточным конвейером" (FC 11)
- 8.7 "Пример счетчика деталей" (FC 12)
- 19.5.1 "Пример: ленточный конвейер" (FC 21)
- 19.5.2 "Пример: счетчик деталей" (FC 22)
- 19.5.3 "Пример: подающий механизм" (FC 20)

Вы можете найти эти примеры на прилагаемой к книге дискете в библиотеке SCL_Book в разделе "Conveyor Example" ("Пример конвейера").

Пример конвейера

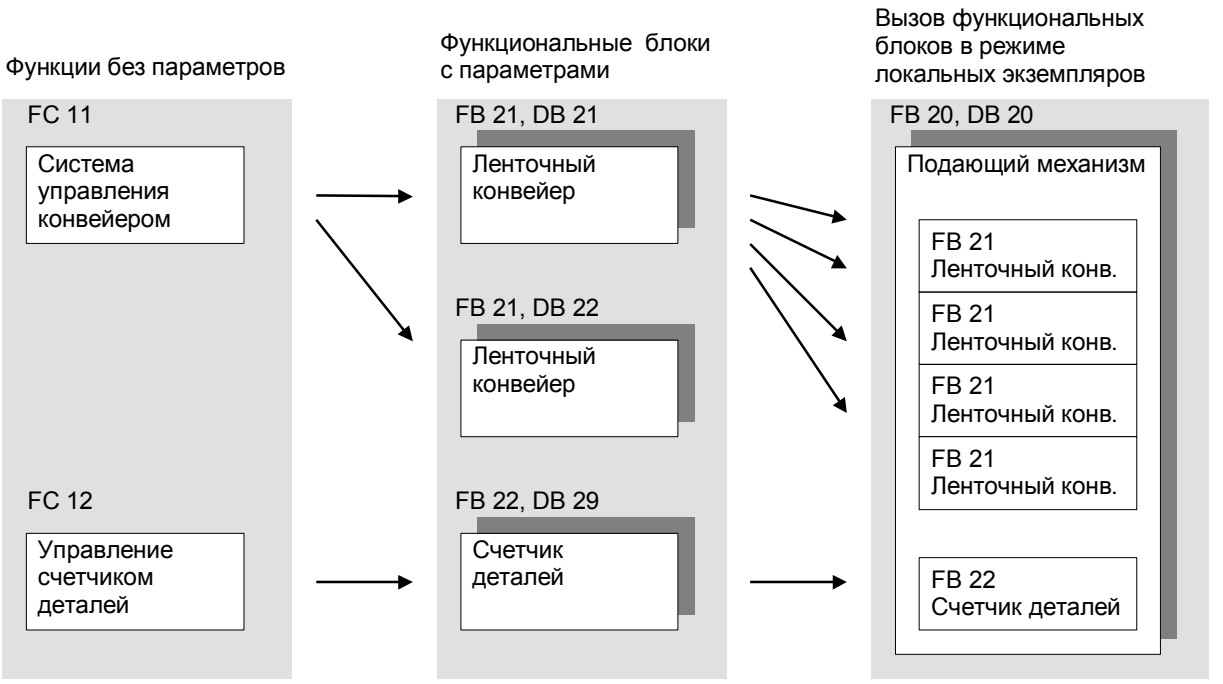


Рис. 30.1 Данные и структура программы примера ленточного конвейера

30.8.2 Пример фрейма сообщения

Пример "Message Frame" ("Фрейм сообщения") показывает, как обрабатывать пользовательские данные и как копировать данные из областей. В этом контексте разработан пример на языке программирования STL с использованием косвенной адресации посредством адресных регистров и с управлением ANY-указателем (см. раздел 26.4 "Краткое описание примера "Message Frame Example" (Пример фрейма сообщения)").

Те же функции можно было бы реализовать, используя язык программирования SCL, и в определенной степени более изящно. Здесь пользователь может использовать возможность обработки во время выполнения программы отдельных элементов массива с адресацией их посредством индексов (см. рис. 30.2).

Кроме того, язык программирования SCL лучше подходит для формулирования задачи (в результате программа получается яснее, что облегчает ее использование и снижает вероятность появления ошибок). Тем не менее, с помощью прямого доступа к переменным (чего нет в SCL) при использовании языка STL легко достигается требуемый результат при создании программы. Таким образом, при программировании одной и той же задачи при использовании языков SCL и STL применяются несколько различные подходы.

Вы можете найти программу для этого примера на прилагаемой к книге дискете в библиотеке SCL_Book в разделе "Message Frame Example" ("Пример фрейма сообщения").

30.8.3 Общие примеры

В общих примерах представлена обработка переменных сложных типов и управление ANY-указателем с помощью видов типа данных.

Следующие функции выполняют преобразования типов данных средствами языка SCL:

- FC 61 DT_TO_STRING
  функция для извлечения даты и преобразования в переменную типа STRING
- FC 62 DT_TO_DATE
  функция для извлечения даты и преобразования в переменную типа DATE
- FC 63 DT_TO_TOD
  функция для извлечения времени суток и преобразования в переменную типа TOD

Следующие функциональные блоки позволяют получить доступ к переменным сложных типов, а также - выполнить обработку данных в кольцевом буфере и в FIFO регистре:

- FB 61 Variable_length ("длина переменной")
- FB 62 Checksum ("контрольная сумма")
- FB 63 Ring_buffer ("кольцевой буфер")
- FB 64 FIFO_register ("FIFO регистр")

В исходных программах "STL Functions" ("STL-функции") и "Call STL Functions" ("Вызов STL-функций") показано, как записывать на STL простые функции и как вставлять эти функции в Вашу SCL-программу. Вы научитесь использовать SIMATIC таймеры и счетчики, используя язык программирования SCL, так же как в стандартных языках программирования.

Вы можете найти рассмотренные примеры на прилагаемой к книге дискете в библиотеке SCL_Book в разделе "General Example" ("Общие примеры").

Пример работы с фреймом сообщения

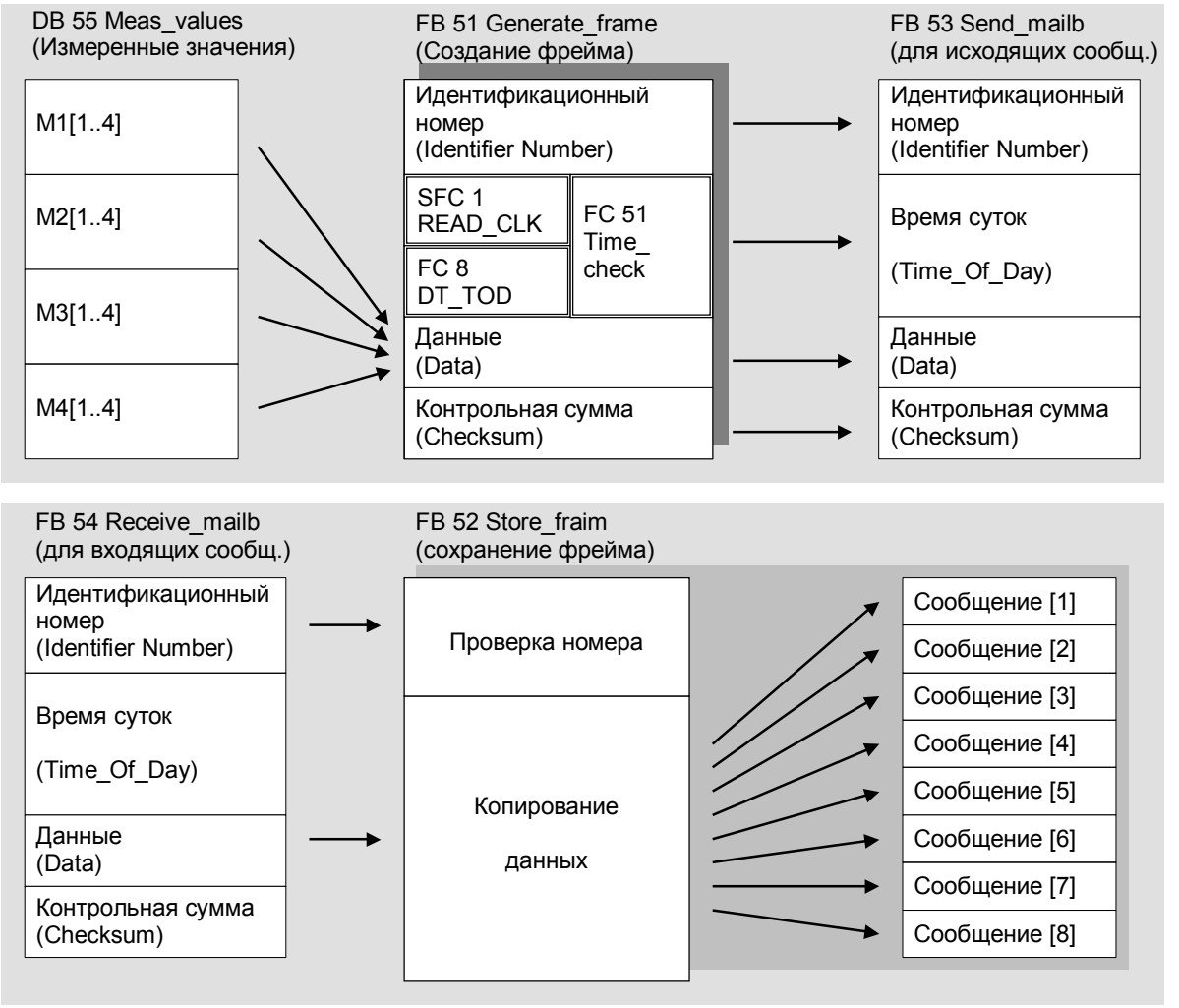


Рис. 30.2 Данные и структура программы для примера работы с фреймом сообщения

31 IEC-функции

IEC-функции - это загружаемые функции FC; эти функции поддерживаются в STEP 7. IEC-функции размещаются в стандартной библиотеке *Standard Library* в разделе *IEC Function Blocks (Функциональные блоки IEC стандарта)*. Они являются стандартными функциями в SCL и могут также использоваться в других языках, например, в таких как STL. Вы можете найти обзор всех IEC-функций в Приложениях. Эти функции могут быть разбиты на следующие функциональные группы:

- Функции преобразования
- Функции сравнения для данных типа DATE_AND_TIME
- Функции сравнения для данных типа STRING
- Функции для данных типа STRING
- Функции для данных типа Date/Time-of-day
- Функции для численных данных

Вызов функций представляется в SCL-нотации. Если Вы используете IEC-функции в STL-программе, то функциональные значения имеют имя RET_VAL и являются первым выходным параметром функции.

Пример:

Вызов в SCL:

```
CompResult := EQ_STRING(
        S1 := string1,
        S2 := string2);
```

Вызов в STL:

```
CALL EQ_STRING(
        S1 := string1,
        S2 := string2,
        RET_VAL := CompResult);
```

Некоторые IEC-функции устанавливают двоичный результат BR в качестве сообщения о групповой ошибке. BR может быть проверен в SCL посредством выхода ENO, а в STL непосредственным использованием двоичной проверки или с помощью функции перехода.

31.1 Функции преобразования (Conversion Functions)

Общая информация

Функции преобразования преобразуют типы переменных. Значение, которое необходимо преобразовать, является входным параметром функции, а функциональное значение является результатом преобразования и относится к другому типу данных.

Общий вид вызова:

```
var_out := функция_преобразования(var_in);
```

где:

var_in - входной параметр

var_out - выходной параметр (функциональное значение)

Некоторые функции преобразования устанавливают двоичный результат BR слова состояния или выход ENO в состояние FALSE (ЛОЖЬ), если в процессе выполнения преобразования типа данных происходит ошибка. В таком случае преобразование не выполняется.

Пример:

Пусть, целая величина INT, содержащаяся в переменной *Speed*, должна быть преобразована в строку символов, которая должна быть записана в переменную *Display*.

```
Display := I_STRNG(Speed);
IF ENO
   THEN (* преобразование выполнено корректно *);
   ELSE (* произошла ошибка *);
END_IF;
```

Если Вы назначаете функциональное значение типа STRING переменной типа STRING, размещенной во временных локальных данных, то Вы в программе должны присвоить этой переменной определенное значение с заданной длиной (так как предопределение ее значения во временных локальных данных посредством объявления невозможно).

Определенная область (число байтов) резервируется для переменной типа STRING, объявленной в области временных локальных данных. Вы можете установить длину в окне свойств компилятора Compiler Properties. Если Вы не объявляете размера переменной типа STRING, то для длины будет принято значение 254(+2) байта.

FC 33 S5TI_TIM
Преобразование типа данных S5TIME в TIME

Функция FC 33 S5TI_TIM выполняет преобразование данных формата S5TIME в данные формата TIME.

Функция FC 33 S5TI_TIM не сообщает об ошибках.

FC 40 TIM_S5TI
Преобразование типа данных TIME в S5TIME

Функция FC 40 TIM_S5TI выполняет преобразование данных формата TIME в данные формата S5TIME. При преобразовании данных происходит округление данных.

Если входной параметр больше, чем позволяет представить формат S5TIME (то есть значение больше величины TIME#02:46:30.000), то в результате выходной параметр будет иметь значение TIME#999.3, а двоичный результат или выходной параметр ENO получит значение FALSE (ЛОЖЬ).

FC 16 I_STRNG
Преобразование типа данных INT в STRING

Функция FC 16 I_STRNG выполняет преобразование данных формата INT в данные формата STRING. Результат (строка символов) будет представлен с "лидирующим" знаком (определенное количество цифр плюс знак).

Если строка символов, определенная как функциональное значение, слишком короткая, то преобразование не происходит, а двоичный результат или выходной параметр ENO получает значение FALSE (ЛОЖЬ).

FC 5 DI_STRNG
Преобразование типа данных DINT в STRING

Функция FC 5 DI_STRNG выполняет преобразование данных формата DINT в данные формата STRING. Результат (строка символов) будет представлен с "лидирующим" знаком (определенное количество цифр плюс знак).

Если строка символов, определенная как функциональное значение, слишком короткая, то преобразование не происходит, а двоичный результат или выходной параметр ENO получает значение FALSE (ЛОЖЬ).

FC 30 R_STRNG
Преобразование типа данных REAL в STRING

Функция FC 30 R_STRNG выполняет преобразование данных формата REAL в данные формата STRING. Результат (строка символов) будет представлен как строка из 14 разрядов:

±v.nnnnnnnE±xx , где
- ± - знак числа;
- v - число перед десятичной точкой;
- n - один из 7-ми разрядов десятичной дроби после десятичной точки;
- x - один из 2-х разрядов показателя степени с основанием 10 (которую символизирует символ E).

Если строка символов, определенная как функциональное значение, слишком короткая или, если во входном параметре находится значение, не являющееся корректным числом с плавающей запятой, то преобразование не происходит, а двоичный результат или выходной параметр ENO получает значение FALSE (ЛОЖЬ).

FC 38 STRNG_I
Преобразование типа данных STRING в INT

Функция FC 38 STRNG_I выполняет преобразование данных формата STRING в данные формата INT. Первый символ строки может быть знаком или цифрой, а все последующие символы должны быть цифрами.

Если длина строки символов равна 0 или больше 6, или если строка содержит неразрешенные символы, или если значение после преобразования выходит за пределы допустимых величин для данных формата INT, то преобразование не происходит, а двоичный результат или выходной параметр ENO получает значение FALSE (ЛОЖЬ).

FC 37 STRNG_DI
Преобразование типа данных STRING в DINT

Функция FC 37 STRNG_DI выполняет преобразование данных формата STRING в данные формата DINT. Первый символ строки может быть знаком или цифрой, а все последующие символы должны быть цифрами.

Если длина строки символов равна 0 или больше 11, или если строка содержит неразрешенные символы, или если значение после преобразования выходит за пределы допустимых величин для данных формата DINT, то преобразование не происходит, а двоичный результат или выходной параметр ENO получает значение FALSE (ЛОЖЬ).

FC 39 STRNG_R
Преобразование типа данных STRING в REAL

Функция FC 39 STRNG_R выполняет преобразование данных формата STRING в данные формата REAL. Исходная строка символов должна иметь следующий формат:

±v.nnnnnnnE±xx , где
- ± - знак числа;
- v - число перед десятичной точкой;
- n - один из 7-ми разрядов десятичной дроби после десятичной точки;
- x - один из 2-х разрядов показателя степени с основанием 10.

Если исходная строка содержит меньшее число символов, чем 14, или, если эта строка имеет иную структуру, нежели указано выше, значение, или, если значение после преобразования выходит за пределы допустимых величин для данных формата REAL, то преобразование не происходит, а двоичный результат или выходной параметр ENO получает значение FALSE (ЛОЖЬ).

31.2 Функции сравнения (Comparison Functions)

Функции сравнения позволяют сравнивать значения двух переменных и выдавать результат сравнения в виде функционального значения. Функциональное значение равно TRUE (ИСТИНА), если результат сравнения положительный, и равно FALSE (ЛОЖЬ), если результат сравнения отрицательный. Функции сравнения не сообщают о возникновении ошибок. Различают функции сравнения для работы с переменными типа DT и для работы с переменными типа STRING.

Общий вид вызова:

```
Result := функция_сравнения_DT(
          DT1 := DT_var1,
          DT2 := DT_var2);

Result := функция_сравнения_STRING(
          S1 := STRING_var1,
          S2 := STRING_var2);
```

FC 9 EQ_DT
Сравнение переменных типа DT (проверка на равенство переменных)

Функция FC 9 EQ_DT выполняет операцию сравнения двух переменных типа DATE_AND_TIME, при этом осуществляется проверка на равенство этих переменных. Функциональное значение равно TRUE (ИСТИНА), только в том случае, если результат сравнения положительный, то есть, если значение параметра DT1 равно значению параметра DT2.

FC 28 NE_DT
Сравнение переменных типа DT (проверка на неравенство переменных)

Функция FC 28 NE_DT выполняет операцию сравнения двух переменных в формате DATE_AND_TIME, при этом осуществляется проверка на неравенство этих переменных. Функциональное значение равно TRUE (ИСТИНА), только в том случае, если результат сравнения положительный, то есть, если значение параметра DT1 не равно значению параметра DT2.

FC 14 GT_DT
Сравнение переменных типа DT
(проверка переменных на отношение по формуле "больше, чем")

Функция FC 14 GT_DT выполняет операцию сравнения двух переменных в формате DATE_AND_TIME, при этом осуществляется проверка этих переменных на отношение по формуле "больше, чем". Функциональное значение равно TRUE (ИСТИНА), только в том случае, если результат сравнения положительный, то есть, если значение параметра DT1 больше (то есть, имеет более позднее значение времени), чем значение параметра DT2.

FC 12 GE_DT
Сравнение переменных типа DT
(проверка переменных на отношение по формуле "больше или равно")

Функция FC 12 GE_DT выполняет операцию сравнения двух переменных в формате DATE_AND_TIME, при этом осуществляется проверка этих переменных на отношение по формуле "больше или равно". Функциональное значение равно TRUE (ИСТИНА), только в том случае, если результат сравнения положительный, то есть, если значение параметра DT1 больше (то есть, имеет более позднее значение времени), чем значение параметра DT2 или равно этому значению (то есть, если оба параметра содержат одинаковое значение времени).

FC 23 LT_DT
Сравнение переменных типа DT
(проверка переменных на отношение по формуле "мен**ьше, чем")**

Функция FC 23 LT_DT выполняет операцию сравнения двух переменных в формате DATE_AND_TIME, при этом осуществляется проверка этих переменных на отношение по формуле "меньше, чем". Функциональное значение равно TRUE (ИСТИНА), только в том случае, если результат сравнения положительный, то есть, если значение параметра DT1 меньше (то есть, имеет более раннее значение времени), чем значение параметра DT2.

FC 18 LE_DT
Сравнение переменных типа DT
(проверка переменных на отношение по формуле "меньше или равно")

Функция FC 18 LE_DT выполняет операцию сравнения двух переменных в формате DATE_AND_TIME, при этом осуществляется проверка этих переменных на отношение по формуле "меньше или равно". Функциональное значение равно TRUE (ИСТИНА), только в том случае, если результат сравнения положительный, то есть, если значение параметра DT1 меньше (то есть, имеет более раннее значение времени), чем значение параметра DT2 или равно этому значению (то есть, если оба параметра содержат одинаковое значение времени).

FC 10 EQ_STRNG
Сравнение переменных типа STRING (проверка на равенство переменных)

Функция FC 10 EQ_STRNG выполняет операцию сравнения двух переменных в формате STRING, при этом осуществляется проверка на равенство этих переменных. Функциональное значение равно TRUE (ИСТИНА), только в том случае, если результат сравнения положительный, то есть, если строка символов в параметре S1 идентична строке символов в параметре S2.

FC 29 NE_STRNG
Сравнение переменных типа STRING (проверка на неравенство переменных)

Функция FC 29 NE_STRNG выполняет операцию сравнения двух переменных в формате STRING, при этом осуществляется проверка на неравенство этих переменных. Функциональное значение равно TRUE (ИСТИНА), только в том случае, если результат сравнения положительный, то есть, если строка символов в параметре S1 отличается от строки символов в параметре S2.

FC 15 GT_STRNG
Сравнение переменных типа STRING
(проверка переменных на отношение по формуле "больше, чем")

Функция FC 15 GT_STRNG выполняет операцию сравнения двух переменных в формате STRING, при этом осуществляется проверка этих переменных на отношение по формуле "больше, чем". Функциональное значение равно TRUE (ИСТИНА), только в том случае, если результат сравнения положительный, то есть, если строка символов в параметре S1 больше, чем строка символов в параметре S2. Переменные сравниваются

в соответствии с их ASCII-кодом (например, имеет ли 'A' большее значение кода, чем 'a' ?), начиная с левого символа строки. Первый отличающийся символ определяет результат операции сравнения. Если, начиная с первого символа, все символы одной строки идентичны соответствующим символам другой строки, то более длинная строка символов принимается, как имеющая большее значение в операции сравнения переменных S1 и S2.

FC 13 GE_STRNG
Сравнение переменных типа STRING
(проверка переменных на отношение по формуле "больше или равно")

Функция FC 13 GE_STRNG выполняет операцию сравнения двух переменных, имеющих формат STRING, при этом осуществляется проверка этих переменных на отношение по формуле "больше или равно". Функциональное значение равно TRUE (ИСТИНА), только в том случае, если результат сравнения положительный, то есть, если строка символов в параметре S1 больше, чем строка символов в параметре S2 или обе строки идентичны. Переменные сравниваются в соответствии с их ASCII-кодом (например, имеет ли 'A' большее значение кода, чем 'a' ?), начиная с левого символа строки. Первый отличающийся символ определяет результат операции сравнения. Если, начиная с первого символа, все символы одной строки идентичны соответствующим символам другой строки, то более длинная строка символов принимается, как имеющая большее значение в операции сравнения переменных S1 и S2.

FC 24 LT_STRNG
Сравнение переменных типа STRING
(проверка переменных на отношение по формуле "меньше, чем")

Функция FC 24 LT_STRNG выполняет операцию сравнения двух переменных в формате STRING, при этом осуществляется проверка этих переменных на отношение по формуле "меньше, чем". Функциональное значение равно TRUE (ИСТИНА), только в том случае, если результат сравнения положительный, то есть, если строка символов в параметре S1 меньше, чем строка символов в параметре S2. Переменные сравниваются в соответствии с их ASCII-кодом (например, имеет ли 'A' меньшее значение кода, чем 'a' ?), начиная с левого символа строки. Первый отличающийся символ определяет результат операции сравнения. Если, начиная с первого символа, все символы одной строки идентичны соответствующим символам другой строки, то менее длинная строка символов принимается, как имеющая меньшее значение в операции сравнения переменных S1 и S2.

FC 19 LE_STRNG
Сравнение переменных типа STRING
(проверка переменных на отношение по формуле "меньше или равно")

Функция FC 19 LE_STRNG выполняет операцию сравнения двух переменных, имеющих формат STRING, при этом осуществляется проверка этих переменных на отношение по формуле "меньше или равно". Функциональное значение равно TRUE (ИСТИНА), только в том случае, если результат сравнения положительный, то есть, если строка символов в параметре S1 меньше, чем строка символов в параметре S2 или обе строки идентичны. Переменные сравниваются в соответствии с

их ASCII-кодом (например, имеет ли 'A' меньшее значение кода, чем 'a' ?), начиная с левого символа строки. Первый отличающийся символ определяет результат операции сравнения. Если, начиная с первого символа, все символы одной строки идентичны соответствующим символам другой строки, то менее длинная строка символов принимается, как имеющая меньшее значение в операции сравнения переменных S1 и S2.

31.3 Функции для данных типа STRING (STRING Functions)

Функции для данных типа STRING (STRING Functions) позволяют обрабатывать строки символов. Некоторые функции для данных типа STRING устанавливают двоичный результат BR слова состояния или выход ENO в состояние FALSE (ЛОЖЬ), если при выполнении функции возникает ошибка.

Функции для данных типа STRING (STRING Functions) позволяют проверить фактический параметр на корректность (валидность) (например: проверить, имеет ли параметр блока, использующий переменную типа STRING достаточную длину?). Если Вы объявляете переменную типа STRING в области временных локальных данных и затем используете ее как фактический параметр, то Вы должны сначала в программе присвоить этой переменной некоторое (любое) значение формата STRING с требуемой длиной. Причина этого требования заключается в том, что предопределение значений переменных в области временных локальных данных при объявлении невозможно. То есть, без определения значений для этих переменных их содержимое носит случайный характер, а в случае переменных типа STRING случайными значениями заполнены также байты, несущие информацию о максимальной и текущей длине такой переменной. Указанные байты принимают значащие величины при присвоении переменной определенного значения.

Определенная область (число байтов) резервируется для переменной типа STRING, объявленной в области временных локальных данных. Вы можете установить длину в окне свойств компилятора Compiler Properties. Если Вы не объявляете размера переменной типа STRING, то для длины будет принято значение 254(+2) байта.

FC 21 LEN
Возвращение длины переменной типа STRING

Вызов функции:

```
int := LEN(string);
```

Функция FC 21 LEN возвращает текущую длину строки символов (число значащих символов) как функциональное значение. Пустая строка имеет нулевую длину. Максимальная длина для строки символов составляет 254 байта.

Функция не выдает сообщений об ошибках.

FC 11 FIND
Поиск в переменной типа STRING

Вызов функции:

```
int:=FIND(IN1:=string,IN2:=string);
```

Функция FC 11 FIND выполняет поиск позиции подстроки IN2 (второй переменной типа STRING) в строке символов IN1 (в первой переменной типа STRING). Поиск осуществляется с крайнего левого символа строки; при первом обнаружении строки IN2 функция выдает результат операции. Если строка IN1 не содержит строки IN2, то функция возвращает нулевое значение.

Функция не выдает сообщений об ошибках.

FC 20 LEFT
Возвращение указанного числа символов из левой части переменной типа STRING

Вызов функции:

```
string:=LEFT(IN:=string,L:=int);
```

Функция FC 20 LEFT выполняет возврат указанного числа символов (L) строковой переменной, начиная с 1-го символа слева (то есть, возврат первых L символов). Если число L больше, чем текущая длина строки символов, то функция возвращает входное значение (IN) целиком. Если L = 0 или входное значение IN = 0, то функция возвращает "пустую" строку.

Если L имеет отрицательное значение, то функция также возвращает "пустую" строку, а двоичный результат BR слова состояния или выход ENO получают значение FALSE (ЛОЖЬ).

FC 32 RIGHT
Возвращение указанного числа символов из правой части переменной типа STRING

Вызов функции:

```
string:=RIGHT(IN:=string,L:=int);
```

Функция FC 32 RIGHT выполняет возврат указанного числа символов (L) строковой переменной, начиная с 1-го символа справа (то есть, возврат последних L символов). Если число L больше, чем текущая длина строки символов, то функция возвращает входное значение (IN) целиком. Если L = 0 или входное значение IN = 0, то функция возвращает "пустую" строку.

Если L имеет отрицательное значение, то функция также возвращает "пустую" строку, а двоичный результат BR слова состояния или выход ENO получают значение FALSE (ЛОЖЬ).

FC 26 MID
Возвращение указанного числа символов из средней части переменной типа STRING

Вызов функции:

```
string:=MID(IN:=string,L:=int,P:=int);
```

Функция FC 26 MID выполняет возврат указанного числа символов (L) из средней части строковой переменной, начиная с P-го символа слева включительно. Если сумма значений L и P по величине больше, чем текущая длина строки символов, то функция возвращает часть строковой переменной, начиная с P-го символа до конца входного значения IN. целиком. Если L = 0 или входное значение IN = 0, то функция возвращает "пустую" строку.

Во всех остальных случаях (если значение P больше значения L или меньше 1, или если значение P и/или L имеют нулевое или отрицательное значения, то функция также возвращает "пустую" строку, а двоичный результат BR слова состояния или выход ENO получают значение FALSE (ЛОЖЬ).

FC 2 CONCAT
Операция конкатенации (сцепления) двух переменных типа STRING

Вызов функции:

```
string:=CONCAT(IN1:=string,IN2:=string);
```

Функция FC 2 CONCAT выполняет операцию конкатенации (сцепления) двух переменных типа STRING и создает из них одну строку символов. Если в результате данной операции получается строка символов, имеющая большую длину, чем задано для выходного параметра (RET_VAL), то результирующая строка ограничивается заданным для нее форматом, а двоичный результат BR слова состояния или выход ENO получают значение FALSE (ЛОЖЬ).

FC 17 INSERT
Операция вставки в переменную типа STRING

Вызов функции:

```
string:=INSERT(IN1:=string,IN2:=string,P:=int);
```

Функция FC 17 INSERT выполняет операцию вставки строки символов из параметра IN2 в строку символов из параметра IN1 после P-го символа. Если параметр P равен 0, то вторая строка *string2* вставляется перед первой строкой *string1*; если параметр P больше, чем текущая длина первой строки символов, то вторая строка вставляется после первой строки.

Если параметр P имеет отрицательное значение, в результате данной операции получается "пустая" строка символов, а двоичный результат BR слова состояния или выход ENO получают значение FALSE (ЛОЖЬ). Двоичный результат BR или выход ENO также получают значение FALSE (ЛОЖЬ), если в результате данной операции получается строка символов, имеющая большую длину, чем задано для выходного параметра; в таком случае результирующая строка ограничивается заданной для нее максимальной длиной.

FC 4 DELETE
Операция удаления части строки из переменной типа STRING

Вызов функции:

```
string:=DELETE(IN:=string,L:=int,P:=int);
```

Функция FC 4 DELETE выполняет операцию удаления L символов из исходной строки символов, начиная с P-го символа включительно. Если L = 0 и/или P = 0, или если значение P больше, чем текущее значение длины входной строки IN, то функция возвращает исходную строку. Если сумма значений L и P по величине больше, чем текущая длина строки символов, то при выполнении функции удаляется часть входной строковой переменной, начиная с P-го символа до ее конца.

Если значение P и/или L имеют отрицательное значение, то функция также возвращает "пустую" строку, а двоичный результат BR слова состояния или выход ENO получают значение FALSE (ЛОЖЬ).

FC 31 REPLACE
Операция замены части строки в переменной типа STRING

Вызов функции:

```
string:=REPLACE(IN1:=string,IN2:=string,L:=int,P:=int);
```

Функция FC 31 REPLACE выполняет операцию замены L символов в исходной строке символов (IN1), начиная с P-го символа включительно, второй строкой символов (IN2). Если L = 0, то функцией возвращается исходная строка (IN1). Если P = 0 или P = 1, то замена L символов в исходной строке символов (IN1), начиная с 1-го символа включительно. Если значение P больше, чем текущее значение длины входной строки (IN1), то функция вставляет вторую строку после исходной строки (сцепляет эти строки).

Если значение P и/или L имеют отрицательное значение, то функция также возвращает "пустую" строку, а двоичный результат BR слова состояния или выход ENO получают значение FALSE (ЛОЖЬ). Двоичный результат BR или выход ENO также получают значение FALSE (ЛОЖЬ), если в результате данной операции получается строка символов, имеющая большую длину, чем задано для выходного параметра; в таком случае результирующая строка ограничивается заданной для нее максимальной длиной.

31.4 Функции для данных типа Date/Time-of-Day (Date/Time-of-Day Functions)

Функции для данных типа Date/Time-of-Day (Date/Time-of-Day Functions) позволяют обрабатывать переменные типов DATE, TIME-OF-DAY и DATE_AND_TIME.

Некоторые функции для данных типа Date/Time-of-Day устанавливают двоичный результат BR слова состояния или выход ENO в состояние FALSE (ЛОЖЬ), если при выполнении функции возникает ошибка.

FC 3 D_TOD_DT
Объединение переменных форматов DATE и TIME_OF_DAY в переменную, имеющую формат DATE_AND_TIME

Вызов функции:

```
date_and_time:= D_TOD_DT(IN1:=date,IN2:=time_of_day);
```

Функция FC 3 D_TOD_DT объединяет данные форматов DATE (D#) и TIME_OF_DAY (TOD#) и преобразует их в данные формата DATE_AND_TIME (DT#). Входное значение IN1 принимать значение между границами DATE#1990-01-01 и DATE#2089-12-31.

Функция не сообщает об ошибках.

FC 6 DT_DATE
Извлечение значения даты DATE из переменной формата DATE_AND_TIME

Вызов функции:

```
date:=DT_DATE(date_and_time);
```

Функция FC 6 DT_DATE позволяет извлекать данные формата DATE (D#) из данных типа DATE_AND_TIME (DT#). Значение DATE должно находиться между граничными значениями DATE#1990-1-1 и DATE#2089-12-31.

Функция не сообщает об ошибках.

FC 7 DT_DAY
Извлечение информации о дне недели из переменной формата DATE_AND_TIME

Вызов функции:

```
int:=DT_DAY(date_and_time);
```

Функция FC 7 DT_DAY позволяет извлекать данные о дне недели из переменной формата DATE_AND_TIME (DT#).

День недели представляется в формате данных INT (от 1 до 7):

1 Воскресенье
2 Понедельник
3 Вторник
4 Среда
5 Четверг
6 Пятница
7 Суббота

Функция не сообщает об ошибках.

FC 8 DT_TOD
Извлечение данных о времени суток из переменной формата DATE_AND_TIME

Вызов функции:

```
time_of_day:=DT_TOD(date_and_time);
```

Функция FC 8 DT_TOD позволяет извлекать данные о времени суток (данные формата TIME_OF_DAY (TOD#)) из данных формата DATE_AND_TIME (DT#).

Функция не сообщает об ошибках.

FC 1 AD_DT_TM
Добавление временного промежутка к определенному значению времени суток

Вызов функции:

```
date_and_time:=AD_DT_TM(T:=date_and_time,D:=time);
```

Функция FC 1 AD_DT_TM позволяет добавить заданный промежуток времени (значение в формате TIME (T#)) к определенному значению времени суток (значение в формате DATE_AND_TIME (DT#)) и выдает в качестве результата новое значение времени суток (значение в формате DATE_AND_TIME (DT#)). Исходное значение времени суток (параметр T) должно находиться в диапазоне:
от DT#1990-01-01-00:00:00.000 до DT#2089-12-31-23:59:59.999.

Функция не выполняет проверки входного значения.

Если результат сложения не находится внутри указанного выше диапазона, то результат ограничивается соответствующим значением, и при этом двоичный результат BR слова состояния или выход ENO устанавливаются в состояние "FALSE" ("ЛОЖЬ").

FC 35 SB_DT_TM
Вычитание временного промежутка из определенного значения времени суток

Вызов функции:

```
date_and_time:=SB_DT_TM(T:=date_and_time,D:=time);
```

Функция FC 35 SB_DT_TM позволяет вычесть заданный промежуток времени (значение в формате TIME (T#)) из определенного значения времени суток (значение в формате DATE_AND_TIME (DT#)) и выдает в качестве результата новое значение времени суток (значение в формате DATE_AND_TIME (DT#)). Исходное значение времени суток (параметр T) должно находиться в диапазоне:
от DT#1990-01-01-00:00:00.000 до DT#2089-12-31-23:59:59.999.

Функция не выполняет проверки входного значения.

Если результат вычитания не находится внутри указанного выше диапазона, то результат ограничивается соответствующим значением, и при этом двоичный результат BR слова состояния или выход ENO устанавливаются в состояние "FALSE" ("ЛОЖЬ").

FC 34 SB_DT_DT
Вычитание одного значения времени суток из другого

Вызов функции:

```
date_and_time:=SB_DT_DT(T1:=date_and_time,
T2:=date_and_time);
```

Функция FC 34 SB_DT_DT позволяет выполнить операцию вычитания одного значения времени суток из другого (оба значения имеют формат DATE_AND_TIME (DT#)). Функция выдает в качестве результата значение интервала времени (значение в формате TIME (T#)).

Значения времени должны находиться в диапазоне:
от DT#1990-01-01-00:00:00.000 до DT#2089-12-31-23:59:59.999.

Функция не выполняет проверки входного значения.

Если первое значение времени (параметр T1) больше (имеет более позднее значение), чем второе значение (параметр T2), то результат положителен; если первое значение времени меньше (имеет более раннее значение), чем второе значение, то результат отрицателен.

Если результат вычитания не находится внутри указанного выше диапазона для данных формата TIME, то результат ограничивается соответствующим значением, и при этом двоичный результат BR слова состояния или выход ENO устанавливаются в состояние "FALSE" ("ЛОЖЬ").

31.5 Функции для обработки численных данных (Numerical Functions)

Функции для обработки численных данных (Numerical Functions) сохраняют неизменным численное значение и при этом двоичный результат BR слова состояния или выход ENO устанавливаются в состояние "FALSE" ("ЛОЖЬ"), если:

- параметризированная переменная относится к недопустимому типу данных;
- параметризированная переменная относится к другому, а не к ожидаемому, типу данных;
- переменная формата REAL представлена некорректным числом (ожидается формат числа с плавающей запятой и значение числа должно принадлежать допустимому диапазону).

FC 22 LIMIT
Ограничитель (Delimiter)

Вызов функции:

```
any_num:=LIMIT(MN:=any_num,IN:=any_num,MX:=any_num);
```

Функция FC 22 ограничивает числовое значение переменной IN задаваемыми граничными значениями: MN и MX. В качестве входных значений допускаются переменные типов INT, DINT и REAL. Все входные значения (фактические параметры) должны принадлежать к одному и тому же типу данных.

Нижнее граничное значение (параметр MN) должно быть меньше, чем верхнее граничное значение (параметр MX).

Функция сообщает о возникновении ошибки, если выполняется любое из вышеуказанных условий возникновения ошибки, а также, если: нижнее граничное значение (параметр MN) больше, чем верхнее граничное значение (параметр MX).

FC 25 MAX
Выбор максимального числа из трех исходных

Вызов функции:

```
any_num:=MAX(IN1:=any_num,IN2:=any_num,IN3:=any_num);
```

Функция FC 25 MAX позволяет выбрать наибольшее из числовых значений трех входных переменных. В качестве входных значений допускаются значения переменных, принадлежащих к типам данных INT, DINT и REAL. Все входные значения (фактические параметры) должны принадлежать к одному и тому же типу данных.

FC 27 MIN
Выбор минимального числа из трех исходных

Вызов функции:

```
any_num:=MIN(IN1:=any_num,IN2:=any_num,IN3:=any_num);
```

Функция FC 27 позволяет выбрать наименьшее из числовых значений трех входных переменных. В качестве входных значений допускаются значения переменных, принадлежащих к типам данных INT, DINT и REAL. Все входные значения (фактические параметры) должны принадлежать к одному и тому же типу данных.

FC 36 SEL
"Двоичный переключатель" (Binary selection)

Вызов функции:

```
any:=SEL(G:=bool,IN0:=any,IN1:=any);
```

Функция FC 36 SEL позволяет выбрать одно из двух значений переменных (IN0 и IN1) в зависимости от состояния переключателя (параметр G). В качестве входных значений IN0 и IN1 допускаются переменные, принадлежащие к любым простым типам (кроме переменных типа BOOL). Обе входные переменные и выходная переменная должны принадлежать к одному типу данных.

Приложения

В данной части настоящей книги представлены: инструкции по преобразованию программ, предназначенных для STEP 5, в программы для STEP 7, общий обзор содержимого библиотек блоков STEP 7 и общий обзор всех STL- и SCL-инструкций и функций.

Программное обеспечение **S5/S7 Converter** (**S5/S7-конвертер**) является опционным (то есть, поставляемым по отдельному заказу) программным продуктом. С помощью S5/S7-конвертера пользователь имеет возможность преобразовать имеющиеся программы, предназначенные для STEP 5, в STL-программы для STEP 7 в виде исходных файлов.

Наряду с другими программными продуктами в поставку ПО STEP 7 входят библиотеки блоков **Block Libraries** (**Библиотеки блоков**), содержащие загружаемые функции и функциональные блоки, а также заголовки и описания интерфейсов системных функций SFC и системных функциональных блоков SFB.

Загружаемые функции FC и функциональные блоки SFB являются скомпилированными блоками, которые Вы можете копировать в Вашу пользовательскую программу (или, более точно, в автономный [offline] раздел *Blocks [Блоки]*), чтобы в последствии их вызывать. Эти блоки занимают в памяти такое же пространство как и "обычные" пользовательские блоки и, также как пользовательские блоки, загружаются в CPU.

Пользователь может переименовывать загружаемые функции и функциональные блоки, например, в случае, если их номера уже были назначены Вашим собственным блокам. Тем не менее, в случае возникновения проблем, Вы можете получить корректную контекстную помощь (Help). Если Вы выделили интересующий Вас блок, то диалоговое окно справочной системы может быть вызвано с помощью функциональной клавиши F1. Справочная система ориентирована на помощь, которая может касаться вопросов, связанных со свойствами блоков FAMILY (Семейство) и NAME (Имя).

Системные функции SFC и системные функциональные блоки SFB включены в операционную систему CPU. Для того чтобы можно было вызывать эти блоки в автономном (offline) режиме, стандартная библиотека содержит заголовки и описания интерфейсов этих блоков (сама программа, конечно, при этом располагается в CPU). Пользователь может скопировать описания интерфейсов как скомпилированные блоки в автономный (offline) раздел *Blocks (Блоки)* и в дальнейшем вызывать соответствующие системные блоки.

Редактор программ узнает из описания интерфейса, сколько параметров имеет системный блок, к какому типу данных они относятся и какие имена имеют эти параметры.

При инкрементном программировании пользователь может перенести библиотечные блоки из каталога элементов программы в окно программы и, таким образом, обеспечить их вызов. Редактор программы затем автоматически копирует эти блоки в Вашу программу.

Если при программировании путем создания исходной программы Вы вызываете библиотечные блоки с символьными именами из библиотечной таблицы символов, то на этапе компилирования стандартные блоки также будут автоматически скопированы в Вашу программу.

Книга заканчивается обзором STL-операторов и общим обзором SCL-инструкций.

32 S5/S7-конвертер (S5/S7 Converter)

Инструкции для преобразования программ, предназначенных для STEP 5, в программы для STEP 7.

33 Библиотеки блоков (Block Libraries)

Организационные блоки (OB), системные функции (SFC), системные функциональные блоки (SFB), функциональные IEC-блоки (загружаемые IEC-функции), блоки для S5-S7-преобразования (загружаемые функции преобразования), блоки ПИД-управления (функции для автоматического управления), коммуникационные блоки (DP-функции).

34 Обзор STL-операторов

Все STL-операторы.

35 Общий обзор SCL-инструкций

Все SCL-инструкции и SCL-функции.

32 S5/S7-конвертер

С помощью S5/S7-конвертера пользователь имеет возможность преобразовать имеющиеся программы, предназначенные для STEP 5, в STL-программы (для STEP 7) в виде исходных файлов. S5/S7-конвертер обращает непосредственно конвертируемые S5-инструкции в соответствующие инструкции STEP 7. Инструкции STEP 5, которые не могут быть непосредственно преобразованы в инструкции STEP 7, оказываются закомментированными. S5/S7-конвертер учитывает (сохраняет) все комментарии в программе. По выбору (опционно) список назначений может также быть преобразован в импортируемую таблицу символов.

Чтобы преобразовать последовательное управление посредством GRAPH 5 (sequential control) в программу STEP 7, Вы должны вновь создать программу с использованием S7-GRAPH.

S5/S7-конвертер входит в поставку наряду с другими программными продуктами ПО STEP 7. Пользователю не требуется проводить авторизацию для того, чтобы использовать это программное средство.

В электронном каталоге CA01 (CD) Вы найдете поддержку для аппаратного преобразования конфигурации SIMATIC S5 в конфигурацию SIMATIC S7 в пункте меню: *Selection Aids -> SIMATIC (Выбор поддержки -> SIMATIC)*. После выбора конфигурации S5 с помощью опций меню: *Edit -> Generate Signal List (Правка -> Создать список сигналов)* и *Edit -> Generate Configuration (Правка -> Создать конфигурацию)* Вы можете создать S7-станцию из спецификаций для конфигурации S5.

32.1 Общая информация

Для преобразования программы STEP 5 Вам потребуются файл программы с именем *name*ST.S5D, список перекрестных ссылок с именем *name*XR.INI и, если есть в наличии, список назначений с именем *name*Z0.SEQ. Кроме того, Вы можете создать макро-файл. Этот файл содержит последовательность инструкций, которые конвертор может использовать вместо определенных инструкций STEP 5. Из этих файлов конвертор создает исходный файл программы для STEP 7, и, если требуется, таблицу символов. Все созданные файлы сохраняются в том же разделе, что и файлы STEP 5.

Конвертор пересылает организационные блоки с программой пользователя в соответствующие организационные блоки STEP 7, а все остальные кодовые блоки - в функции FC.

Номера блоков FC начинаются с нуля и назначаются в возрастающей последовательности; Вы можете изменять присвоенные номера блоков с помощью соответствующего диалогового окна.

На рис. 32.1 показано соответствие исходных файлов и файлов, полученных в процессе преобразования (конвертации) программы.

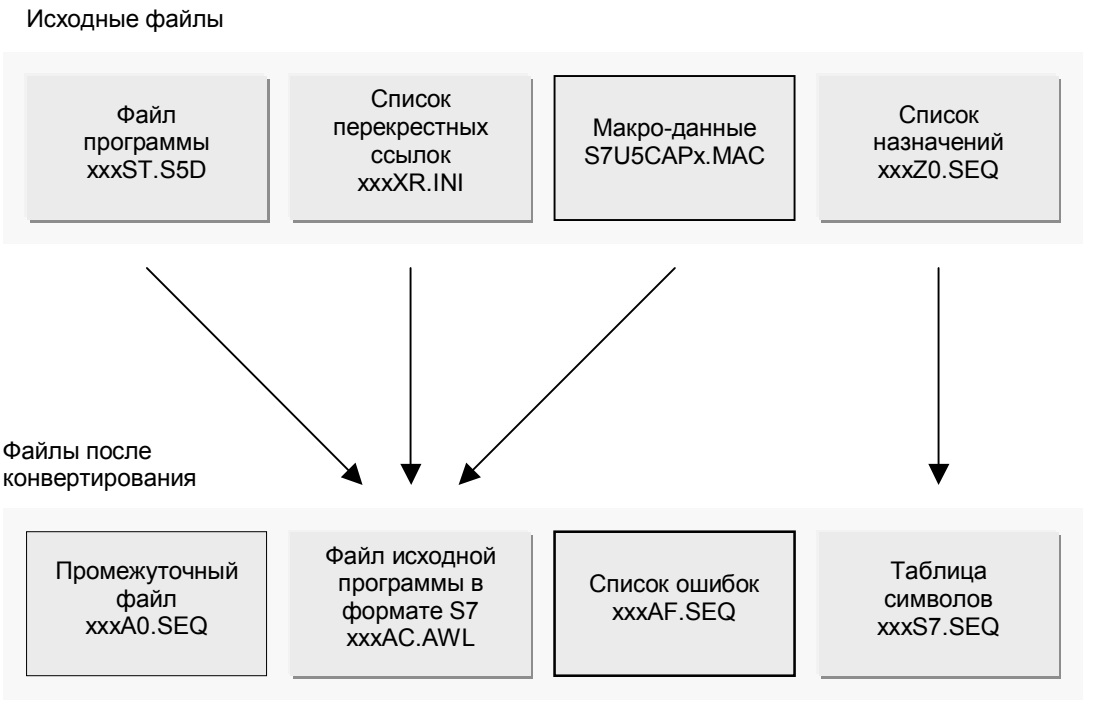


Рис. 32.1 Исходные файлы и файлы после конвертирования программы

Конвертер различает стандартные блоки из следующих прикладных пакетов программ Сименс (Siemens):

- арифметики с плавающей точкой;
- функций для обработки сигналов;
- базовых аналоговых функций;
- математических функций.

Наряду с другими программными продуктами в поставку ПО STEP 7 входит библиотека блоков замены при конвертировании программы *S5/S7 Converting Blocks (Блоки замены при S5/S7 конвертировании программы)*. Эта библиотека содержит блоки, которые заменяют стандартные блоки при S5/S7 конвертировании программы из перечисленных выше прикладных пакетов программ Сименс (Siemens). Вы также можете найти в этой библиотеке стандартные блоки ("integral functions" - "встроенные функции"), которые заменяют некоторые из функциональных блоков, встроенных в CPU S5-115U.

Если программа STEP 5 содержит блоки из указанных программных пакетов, то конвертер преобразует вызовы и сигналы, которые обрабатываются этими блоками в программе.

Вы можете скопировать соответствующие блоки из библиотеки в свою пользовательскую программу перед тем, как Вы начнете процесс компилирования конвертированной программы.

При конвертировании программы STEP 5 Вы можете следовать пунктам процедуры, представленной ниже:

- проверка выполнимости программы в среде STEP 7;
- если необходимо, подготовка STEP 5 - программы (например, удаление неконвертируемых разделов программы, которые должны заменяться параметризацией CPU);
- если необходимо, создание макросов (например, для управляемой замены инструкций STEP 5 на инструкции STEP 7 в процессе конвертации программы);
- процесс конвертации программы (создание исходной программы для STEP 7);
- создание проекта STEP 7 с последующим копированием в этот проект исходной программы и таблицы символов, а также, если это необходимо, копирование в этот проект используемых стандартных функциональных блоков;
- если это необходимо, после конвертирования внесение исправлений или добавлений в исходную программу STEP 7;
- компиляция исходной программы STEP 7.

Указанная последовательность шагов при конвертировании программы не является жестко заданной. Вы можете, например, начать конвертирование STEP 5 программы без ее предварительной подготовки и затем выполнить требуемые корректировки уже в исходной STEP 7 программе.

32.2 Подготовка

32.2.1 Проверка выполнимости программы в системе назначения (PLC)

Если Вам требуется использовать существующую программу STEP 5 в системе SIMATIC S7, Вы должны сначала убедиться, что программа может выполняться в системе назначения (PLC).

Например, необходимо проверить:

- Обладает ли CPU назначения требуемыми свойствами? Существуют ли требуемые характеристики выполнения программы?
- С какими модулями работает программа, предназначенная для STEP 5? К каким модулям обращается программа, предназначенная для STEP 7?
- Обладает ли CPU в системе STEP 7 требуемыми адресами (адресами входов, выходов, блоков)?

Вы можете использовать блок расширения S5 с интерфейсным модулем IM 463-2 или отдельные S5-модули в стойке станции S7-400. Также модули SIMATIC S5 могут быть подключены к системе SIMATIC S7 с использованием распределенной периферии (I/O) посредством шины PROFIBUS-DP.

32.2.2 Проверка параметров выполнения программы

Знакомые Вам по SIMATIC S5 "уровни выполнения программы" ("program execution levels") в основном соответствуют уровням выполнения программы в системе SIMATIC S7, где они называются "приоритетными классами" ("priority classes"). Вы можете заменить установки, которые Вы сделали в блоках данных DB1, или DX0, или, может быть, в системных данных при параметризации S7-CPU (например, параметры перезапуска, обработка таймерных ["watchdog"] прерываний).

Встроенные организационные блоки и встроенные функциональные блоки в S5 соответствуют системным блокам в S7. Если Вы использовали встроенные функции в S5, то в S7 Вам необходимо обеспечить выполнение этих функций с помощью системных блоков или с помощью параметризации CPU.

Блок данных DB1

В S5-115U параметры выполнения программы устанавливаются в блоке данных DB1 или в системных данных RS. В таблице 32.1.1 показано, как эти параметры могут быть обеспечены в SIMATIC S7.

Таблица 32.1.1 Установка параметров в SIMATIC S5 и в SIMATIC S7

Блок данных DB1 и системные данные System Data (S5-115U)			
Функция	941 - 944	945	В S7 заменяется на:
Restart delay (задержка перезапуска)	x	x	Параметр CPU "Restart" ("Перезапуск")
Retentive feature (ременантность)	x	x	Параметр CPU "Retentivity" ("Ременантность")
Cycle time monitoring (мониторинг времени цикла)	x	x	Параметр CPU "Cycle/Clock memory" ("Цикл/Тактовые меркеры")
Time interval for watchdog interrupt (интервал для таймерного прерывания)	x	x	Параметр CPU "Watchdog interrupt" ("Таймерное прерывание")
Software protection (защита программы)	x	x	Параметр CPU "Protection" ("Защита")
Output disable process images (блокировка выходов образа процесса)	x	x	Управление посредством образа подпроцессов SFC 26 UPDAT_PI, SFC 27 UPDAT_PO
Integral clock (встроенные часы)	x	x	Параметр CPU "Diagnostics/Clock" ("Диагностика/Часы") SFC 0 SET_CLK, SFC 1 READ_CLK
Delay interrupt OB 6 (прерывание с задержкой обработки)			
Time duration (продолжительность)	x	x	Параметр CPU "Interrupts" ("Прерывания")
Execution priority (приоритет выполнения)	x	-	Параметр CPU "Interrupts" ("Прерывания")
Sequential process image transfer (последовательная пересылка образа процесса)	-	x	- отсутствует -
Reduced PIQ transfer (сокращенная пересылка образа процесса)	-	x	- отсутствует -

Системные утилиты

CPU S5-115U поддерживают системные утилиты, которые Вы можете применять с помощью организационного блока OB 250 (CPU 945) или с использованием системных данных RS 125 (CPU 941 ... CPU 944).

В таблице 32.1.2 содержатся указания к применению конвертирования этих системных утилит в SIMATIC S7.

Таблица 32.1.2 Установка параметров в SIMATIC S5 и в SIMATIC S7

Системные утилиты OB 250 и BS 125			
Функция	OB 250	BS 125	В S7 заменяется на:
Time intervals for watchdog interrupts (интервалы для таймерных прерываний)	x	-	Параметр CPU "Watchdog interrupt" ("Таймерное прерывание")
Time duration of delay interrupt (продолжительность интервала для прерывания с задержкой обработки)	x	-	Параметр CPU "Interrupts" ("Прерывания")
Reduced PIQ transfer (сокращенная пересылка образа процесса)	x	-	- отсутствует -
Read/write DBA/DBL register (чтение/запись в DBA/DBL регистр)	x	-	Чтение, например, с использованием L DBNO, L DBLG; прямая запись недопустима
Call DX/FX blocks indirectly (вызов блоков DX/FX с косвенной адресацией)	x	-	Вызов блоков с косвенной адресацией
Change block ID (смена идентификатора блока)	x	-	- отсутствует -
Update configuration image (обновление отображения конфигурации)	x	x	- отсутствует -
Set up block address list (создание списка адресов блоков)	x	x	- отсутствует -
Create data block (создание блока данных)	x	x	SFC 22 CREAT_DB
I/O accesses without QVZ (обращение к периферии (I/O) без QVZ)	x	x	Обработка синхронных ошибок: SFC 36 MSK_FLT, SFC 37 DMSK_FLT, SFC 38 READ_ERR,
Disable/Enable digital outputs (блокировка/разблокирование дискретных выходов)	x	x	Master Control Relay MCR (Главное управляющее реле)
Delete block (удаление блока)	x	x	Удаление блока данных: посредством SFC 23 DEL_DB
Update process image (обновление образа процесса)	-	x	Обработка с использованием блоков управления отображением подпроцессов: SFC 26 UPDAT_PI, SFC 27 UPDAT_PO
Interpret data block DB1 (пояснения к блоку данных DB1)	-	x	- отсутствует -

Блок данных DX0

Для CPU, относящихся к категории высокопроизводительных CPU, информация в блоке данных DX0 определяет параметры выполнения программы.

В таблице 32.1.3 содержатся указания по переходу к SIMATIC S7.

Таблица 32.1.3 Установка параметров в SIMATIC S5 и в SIMATIC S7

Блок данных DX0			
Функция	135U	155U	В S7 заменяется на:
Restart characteristics (параметры перезапуска)	x	x	Параметр CPU "Restart" ("Перезапуск")
Number of processed timer cells (число обрабатываемых таймеров)	x	x	- отсутствует - (число фиксировано)
Cycle time monitoring (мониторинг времени цикла)	x	x	Параметр CPU "Cycle/Clock memory" ("Цикл/Тактовые меркеры")
Multiprocessor restart, interprocessor communication flags (перезапуск в мультипроцессорном режиме, флаги для межпроцессорных коммуникаций)	x	x	- отсутствует -
Accuracy of floating point arithmetic (точность выполнения арифметических операций с плавающей запятой)	x	-	- отсутствует -
Timed interrupt handling (обработка временных прерываний)	-	x	Параметр CPU "Watchdog interrupt" ("Таймерное прерывание")
Process interrupt handling (обработка прерываний процесса)	x	x	Параметр CPU "Interrupts" ("Прерывания")
Process interrupt level/ level-triggered interrupt (уровень прерываний процесса/ прерывания запускаемые от уровня)	x	-	Параметризация модуля
Addressing error monitoring (мониторинг ошибок адресации)	x	-	OB 122 (обработка ошибок доступа к входу/выходу [I/O])
Error handling (system stop) (обработка ошибок (остановка системы))	x	-	Заменяется OB обработки ошибок

32.2.3 Проверка модулей

I/O модули

Сравните технические описания используемых I/O модулей со спецификациями SM-модулей для системны S7. Имеются ли аналоговые модули с соответствующими областями? Когда Вы организуете прямой доступ к аналоговым модулям, обратите внимание на различия в формате данных для S5 и S7.

Интеллектуальные I/O модули IP

Вы можете также использовать некоторые IP-модули в станции S7-400 вместе с адаптером:

- IP 240 Модуль позиционирования, модуль декодирования позиции, модуль счетчика;

- IP 242B Модуль счетчика;
- IP 244 Модуль контроля температуры;
- IP 246/247 Модули позиционирования;
- WF 721/723 Модули позиционирования;
- WF 705 Модуль декодирования позиции.

Стандартные блоки для этих модулей поставляются вместе с адаптером. Если Вы используете эти модули, Вы должны заменить блоки стандарта S5 на блоки стандарта S7 и привести Вашу программу в соответствие с новыми значениями инициализации для параметров. Для того, чтобы оставить для использования IP-модули, Вы должны применять соответствующие FM-модули.

Коммуникационные процессоры CP

Используемые в S5 коммуникационные процессоры необходимо заменить на CP-модули с соответствующими функциями. CP-модули в S7 обеспечивают доступ с помощью SFB-коммуникаций (коммуникаций посредством системных функциональных блоков) вместо блоков обработки данных стандарта S5. Их функции похожи, но их реализация обеспечивается ресурсами языков программирования STEP 7. Вы должны настроить соответствующую S5-программу с функциями обработки данных для работы с SFB-блоками.

S5-модули в станции S7-400

Вы можете также подключить модули расширения S5 к станции S7-400, используя *интерфейсный модуль IM 463-2.* К каждому из двух интерфейсов может быть подключено до четырех модулей расширения S5. При этом в одной центральной монтажной стойке может быть установлено до четырех интерфейсных модуля IM 463-2. Соединениями в модулях расширения S5 управляет интерфейсный модуль IM 314. При этом допускается использование только дискретных и аналоговых модулей. Прерывания процесса не переносятся (при конвертировании программы). Вы можете задавать области I/O для S5-модулей в S5-интерфейсном модуле IM 314 (как обычно в системе S5). Поддерживается доступ к областям входов/выходов (I/O) P, Q, IM3 и IM4.

Вы можете использовать в S7-400 некоторые модули IP и WF (см. выше) с адаптерами. Вы должны при этом, как обычно, задать S5-адреса модулей.

Параметризация S5-адресов с назначением S7-адресов производится с помощью утилиты конфигурирования оборудования Hardware Configuration. Вы можете найти интерфейсный модуль IM 463-2 и адаптер ("*adapter casing"*) в каталоге модулей в соответствии с путем размещения: Simatic 400 -> IM-400 -> S5 Adapter. После размещения модулей в монтажной стойке, Вы должны назначить адреса для этих модулей, как для сигнальных S7-модулей, в области периферийных входов/выходов (I/O), отдельно для входных и выходных адресов.

Примечание:

Необходимо обеспечить, чтобы адресные области для адресов S7 и для адресов S5 не перекрывались.

32.2.4 Проверка адресации

Необходимо проверить число доступных адресов в выбранном CPU назначения. Имеется ли достаточное количество входов, выходов, меркеров, таймеров и счетчиков? Конвертер преобразует меркеры из "расширенной" ("extended") области (область S меркеров) в область меркеров, начиная с адреса M 256.0.

В S7 существует одна единая область периферийных входов/выходов (I/O). Все модули, адресованные в S5 в областях входов/выходов (I/O), P, Q, IM3 и IM4 и в "глобальной области" ("global area"), теперь будут адресованы в S7 в области P периферийных входов/выходов (I/O) (Вы должны внимательно отнестись к этому, если у Вас адресовано большое количество модулей в "расширенных" ["extended"] областях I/O и Вы подключаете эти модули к станции S7-400, например, с помощью IM 463-2). Область "страницы" памяти ("the page memory area") пропускается без замещения.

Конвертер преобразует все блоки с пользовательской программой (исключая организационные блоки) в функции, то есть, общее число всех программных блоков (PB), шаговых блоков без программы секвенсора (SB) и функциональных блоков (FB и FX) не должно превышать максимально допустимого числа функций (FC). Аналогично и общее число блоков данных (DB и DX) не должно превышать числа блоков данных S7. На практике эти ограничения касаются только тех случаев, когда Вы используете в качестве целевой системы (PLC) станцию S7-300.

Области системных данных RI, RJ, RS и RT пропускаются без замены в S7. Любая информация, которая располагалась в этих областях сохраняется в S7 в блоках глобальных данных или в области меркеров. После преобразования Вы можете получать системную информацию из RS-области посредством системных функций; инициализацию функций Вы можете выполнять с помощью данной области посредством системных функций или при параметризации CPU.

Подготовка программы STEP 5

Перед преобразованием Вы можете подготовить программу STEP 5 для ее дальнейшего использования в качестве программы для системы STEP 7 (но Вы не должны так непосредственно ее использовать; Вы должны будете сначала после преобразования выполнить все соответствующие корректировки в исходном файле программы STEP 7). С помощью предварительной подготовки программы STEP 5 Вы можете уменьшить число предупреждений (warnings) и сообщений об ошибках при проведении преобразования. Например, Вы можете подготовить программу STEP 5 перед преобразованием, выполнив следующие операции:

- Удаление блоков данных с параметрами программы DB1 или DX0;
- Удаление всех вызовов встроенных блоков или обращений к области системных данных RS; их функции могут быть обеспечены путем параметризации S7-CPU;
- Приведение адресов областей входов, выходов, периферийных входов/выходов (I/O) в соответствие с "новыми" адресами модулей; здесь необходимо обеспечить, чтобы не нарушался диапазон адресов STEP 5, иначе уже на первом проходе процесса преобразования будет

выдано сообщение об ошибке; при этом эти инструкции не будут конвертированы;

- При удалении многократно повторяющихся неконвертируемых разделов программы Вы можете удалять эти разделы, заменяя их на "уникальную" STEP 5-инструкцию в каждом разделе; для этого Вы должны назначить макросу (последовательности инструкций STEP 7) эту "уникальную" STEP 5-инструкцию, заменяющую раздел программы.
- Если Ваша программа содержит много блоков (или большой блок) неструктурированных данных (что используется, например, в буферах данных), то Вы можете значительно уменьшить число инструкций, которые должны быть скомпилированы, и, следовательно, исходный код программы, если Вы удаляете все кроме одного слова данных в данном блоке. После конвертирования программы (перед компилированием) содержимое этого блока данных в исходном файле объявляется массивом, например:

```
Buffer : ARRAY[1..256] OF WORD;
```

Вы можете использовать конвертер не только для конвертирования целых программ, но также для конвертирования отдельных блоков.

32.3 Конвертирование

32.3.1 Создание макросов

Вы можете создавать макросы для замены неконвертируемых инструкций STEP 5 перед тем, как начать процесс конвертирования программы, или для выполнения изменений в программе после стандартного преобразования (конвертирования). Вы можете создавать "макросы конвертирования" ("conversion macros") при помощи конвертера. Если макрос определен дважды, то используется первое его определение. Макросы для набора инструкций SIMATIC с использованием немецких мнемоник хранятся в файле S7U5CAPA.MAC, а макросы для набора инструкций с использованием интернациональных (английских) мнемоник хранятся в файле S7U5CAPB.MAC. Конвертер распознает макросы для инструкций (instruction macros) и макросы для OB (OB macros). Пользователю предоставляется возможность создать 256 макросов для инструкций и 256 макросов для OB.

Макросы для инструкций (instruction macros) заменяют инструкции STEP 5 на соответствующие последовательности инструкций STEP 7. Общая структура макросов для инструкций (instruction macros) имеет следующий вид:

```
$MACRO: <инструкция STEP 5>

<последовательность инструкций STEP 7>

$ENDMACRO
```

Инструкции STEP 5 при этом должны быть определены со своими полными адресами. Тогда конвертер вставит вместо инструкций STEP 5 соответствующие последовательности инструкций STEP 7.

Пример:

Пусть в программе STEP 5 для CPU 945 использовалось прерывания с задержкой обработки ("delay interrupt" - организационный блок OB 6). Активация прерывания осуществляется посредством вызова специальной функции OB 250:

```
L   KF   +200
L   KB      1
JU  OB    250
```

В первой инструкции загрузки содержится число микросекунд, определяющих период задержки вызова организационного блока OB 6. Эта инструкция может быть оставлена, тогда как оставшиеся две инструкции должны быть заменены инструкциями STEP 5, которые не встречаются в Вашей программе, например, TB RT 200.0; таким образом, Ваша программа для STEP 5 перед конвертированием имеет следующий вид:

```
L   KF   +200
TB  RT   200.0
```

Теперь Вы можете записать следующую макро-инструкцию:

```
$MACRO: TB RT 200.0
T MD 250;
  CALL SFC 32 (
  OB_NO     := 20,
  DTIME     := MD 250,
  SIGN      := W#16#0000,
  RET_VAL   := MW 254);
$ENDMACRO
```

Инструкция STEP 5 TB RT 200.0 заменяется при конвертировании программы определенной последовательностью инструкций STEP 7. Значение времени задержки, выраженное в миллисекундах, загружается в ("сверхоперативную память") в слово меркеров MW 250; после этого вызывается системная функция SFC 32. В диалоговом окне перед запуском преобразования конвертер предлагает номер 20 вместо номера 6 для организационного блока обработки прерывания.

Макросы для организационных блоков (OB macros) заменяют вызовы OB (JU OB или JC OB) в программе STEP 5 на определенные последовательности инструкций STEP 7. Общая структура макросов для организационных блоков (OB macros) имеет следующий вид:

```
$MACRO: <Номер OB>
<последовательность инструкций STEP 7>
$ENDMACRO
```

Пример:

Пусть в программе STEP 5 для CPU 945 Вы использовали организационный блок OB 160 для запуска прерывания с задержкой обработки.

В STEP 7 функция задержки времени осуществляется посредством вызова системной функции SFC 47 WAIT. Если Вы запрограммируете следующий макрос:

```
$OBCALL: 160
T MW 250;
CALL SFC 47 (WT := MW 250);
$ENDMACRO
```

то конвертер будет заменять каждый вызов организационного блока OB 160 (в том числе и все вызовы по условию) определенной последовательностью инструкций.

Ввод макроса начинается с выбора следующих опций меню: *Edit -> Replace Macro (Редактор -> Макрос замены).* При этом открывается файл S7U5CAPA.MAC, в который Вы вводите макрос. После окончания ввода Вы сохраняете файл: *File -> Save (Файл -> Сохранить).* Завершение работы с файлом макроса выполняется с помощью опций меню: *File -> Exit (Файл -> Выход).*

32.3.2 Подготовка к конвертированию

Если у Вас нет таблицы перекрестных ссылок *name*XR.INI для Вашей программы в STEP 5, то для преобразования программы Вам необходимо создать эту таблицу (в системе STEP 5 с помощью опций меню: *Manage -> Create XREF (Управление -> Создание таблицы перекрестных ссылок*).

Теперь Вы можете:

- создать Ваш собственный рабочий раздел (папку) для преобразованной программы и скопировать требуемые данные в этот раздел или
- запустить процесс преобразования программы в разделе (в папке), содержащем файлы STEP 5 (если Вы работаете с одним и тем же программатором в системах STEP 5 [и STEP 7]) или
- запустить процесс преобразования программы на дискете (если Вы создали файлы STEP 5 с помощью иного программатора).

Раздел (папка) для преобразованной программы должен содержать файлы *name*ST.S5D, *name*XR.INI, а также, если есть назначение, файл *name*Z0.SEQ. Конвертер в свою очередь добавит в этот раздел "целевые файлы" *name*AC.AWL, nameA0.SEQ и, если есть назначение, *name*AF.SEQ и *name*S7.SEQ.

Файл S7S5CAPx.MAC сохраняется в разделе Windows.

32.3.3 Запуск конвертера

Вызов S5/S7-конвертера производится с помощью опций меню, панели задач Windows 95/NT: *Start -> Simatic -> Step 7 -> Convert File (Пуск ->*

Simatic -> Step 7 -> Конвертировать файл). С помощью опций: *File -> Open (Файл -> Открыть)* Вы можете выбрать файл S5-программы, которую необходимо конвертировать. При щелчке на кнопке "OK" конвертер отобразит исходный файл и целевой файл, так же как и назначение "новых" блоков "старым". Если это необходимо, Вы можете изменить имена целевых файлов (файлов назначения) в соответствующем текстовом поле. Для изменения назначенных номеров блоков, дважды щелкните кнопкой манипулятора "мышь" в соответствующей строке и введите новый номер блока в диалоговом окне.

Конвертер отмечает стандартные блоки звездочкой (Вы должны в дальнейшем скопировать эти блоки из библиотеки блоков в Ваш автономный (offline) пользовательский раздел перед тем, как начнется процесс компилирования исходного S7-файла).

Запуск конвертера осуществляется кнопкой "Start" ("Пуск"). При первом проходе исходной программы конвертер преобразует S5-программу в текстовый файл в формате S5-ASCII (nameA0.SEQ), а на втором проходе переводит этот файл в исходный файл S7-программы. Список (таблица) назначений преобразуется в таблицу символов. В конце процесса конвертирования программы на экране отображаются сообщения об ошибках и предупреждения. Записи обо всех ошибках и предупреждениях содержатся в файле протоколирования ошибок *name*AF.SEQ.

Сообщения об ошибках поступают, если отдельные фрагменты S5-программы не могут быть конвертированы и могут быть перенесены в S7-программу только в виде комментариев. Предупреждения содержат информацию о возможных проблемах; предупреждения поступают, если конвертируемые инструкции вновь требуют проверки. Предупреждения могут касаться S5-программы (например, обнаружен недопустимый код MC 5) или же могут оноситься к S7-программе (например, если обнаружена непреобразованная инструкция). Если Вы щелкнете кнопкой манипулятора "мышь" на строке сообщения, конвертер в специальном окне отобразит ситуацию, касающуюся этого сообщения.

Рекомендуется распечатывать список возникающих ошибок, чтобы эффективно выполнить отладку программы.

32.3.4 Конвертертируемые функции

В таблице 32.2 представлены инструкции, которые по существу остаются неизменными при конвертировании S5-программы в S7-программу. К их числу относятся также инструкции с адресами, которые заменяются в системе STEP 7 другими адресами (например, с такими адресами, как меркеры из "расширенной S области" ["extended S memory bits"], которые заменяются на адреса меркеров из области M, начиная с адреса 256). При преобразовании инструкций из этого списка могут также производиться изменения в их синтаксисе (например, вместо +G будет записано +R). Обычно Вам не придется корректировать рассматриваемые инструкции.

Таблица 32.2 Преобразование функций

Функции в STEP 5	**Функции в STEP 7**
Двоичные логические операции, операции с памятью	Двоичные логические операции, операции с памятью
Функции таймеров и счетчиков (Timers/counters functions)	Функции таймеров и счетчиков (Timers/counters functions)
Функции проверки битов (Bit test functions)	Заменяется операцией установки SET с последующей проверкой или операцией двойного инвертирования Set/Reset (Установка/Сброс)
Функции загрузки (load) и пересылки (transfer) (исключая системные данные и абсолютную адресацию)	Функции загрузки (load) и пересылки (transfer)
Функции сравнения (Comparison functions)	Функции сравнения (Comparison functions)
Функции вычисления (Calculation functions)	Функции вычисления (Calculation functions)
Логические операции с числами (Digital logic operations)	Логические операции со словами (WORD logic operations)
Функции сдвига (Shift functions)	Функции сдвига (Shift functions)
Функции перехода (Jump functions)	Функции перехода (Jump functions)
Функции преобразования (Conversion functions)	Функции преобразования (Conversion functions)
Блокировка/разблокировка прерываний (Disable/Enable interrupts)	Заменяется на SFC 41, SFC 42
Функции Stop (Stop functions)	Заменяется на SFC 46
Нуль-операции (NOP, ***, пустая строка) (Null operations)	NOP, NETWORK, // (пустая строка комментария)

Инструкции замены (осуществляющие доступ к параметрам блока) большей частью конвертируются. Некоторые корректировки должны быть выполнены для инструкций, касающихся функций счетчиков и функций таймеров (например, SEC =*parname [имя_параметра]*), так же как и для обработки параметров блока (DO =*parname [имя_параметра]*). В этом случае и кодовые блоки и блоки данных могут быть использованы в качестве фактических операндов и (что очень важно!): в результате конвертирования номер блока может быть изменен.

Организационные блоки содержат номера, используемые в STEP 7. Все остальные блоки с пользовательской программой становятся функциями FC. Конвертер конвертирует блоки данных DB в блоки глобальных данных с такими же номерами. Блоки данных DX конвертируются в блоки данных DB, начиная с номера 256 (блок DX 1 становится блоком DB 257, и т.д.). Конвертер предлагает пользователю номера блоков; и пользователь может изменить все назначенные номера блоков в диалоговом окне перед запуском процесса конвертирования.

Конвертер принимает библиотечные номера блоков в строке AUTHOR (Автор) в заголовке блока. Имена функциональных блоков принимаются как NAME (имя) без учета специальных символов (другими словами, имя принимается без специальных символов с комментированием исходного имени).

Вызовы специальных функций не конвертируются (они должны быть заменены, например, системными функциями).

Адреса входов и выходов принимаются неизменными. В случае использования инструкций загрузки (load) и пересылки (transfer) с адресами из P-области конвертер использует периферийные входы PI и периферийные выходы PQ с неизменными адресами. Адреса из Q-области накладываются на адресное пространство P-области (периферийные входы/выходы [I/O]), начиная с адреса 256 (так, инструкция `L OB 0` заменяется на `L PIB 256;` инструкция `T OB 1` заменяется на `T PQB 257` и т.д.).

Адреса меркеров области F принимаются без изменения. Это же распространяется на меркеры, используемые как "сверхоперативная память" ("scratchpad memory"), начиная с байта меркеров FY 200 до FY 255. Если Вы конвертируете Вашу программу для STEP 5 в основном без изменения, Вы можете оставить "сверхоперативную память" ("scratchpad memory") как обычно. Если Вам необходимо продолжить использование программы STEP 5 или ее фрагментов в среде STEP 7, то автор данной монографии рекомендует "сверхоперативную память" как блок во временных локальных данных. Это особенно касается случая, если Вы желаете переслать Ваши собственные стандарты из программы STEP 5 в программу STEP 7. Меркеры из "расширенной S области" ["extended S memory bits"] размещаются в адресном пространстве меркеров, начиная с адреса 256 (так, инструкция `A S 0.0` заменяется на `A M 256.0;` инструкция `L SY 2` заменяется на `L MB 258` и т.д.).

Функции таймеров и счетчиков конвертируются без изменения. После конвертирования этих функций становится невозможным прямой доступ в системе STEP 7 к отдельным битам слова значения для таймера и для счетчика. Обработка фронта сигнала состояния в отдельных битах этих слов с помощью операторов проверки состояния бита может быть заменена с использованием операторов SET и CLR вместе с соответствующими операциями таймера и счетчика.

Необходимо отметить, что в системе STEP 7 данные адресуются побайтно (в отличие от STEP 7 в системе STEP 5 данные адресуются "пословно"). Так, DL 0 заменяется на DBB 0; DR 0 заменяется на DBB 1.

В таблице 32.3 Вы можете видеть результаты преобразования адресов при конвертировании программы.

При прямой и косвенной адресации конвертер использует корректные S7-адреса; при адресации данных с помощью параметров блока Вы должны выполнить преобразование для побайтной адресации самостоятельно.

Числа с плавающей запятой принимаются при конвертировании без изменения при том, что они определены как константы в операциях загрузки (load) или они используются как фактические параметры, и они трактуются при преобразовании как числа с плавающей запятой в системе STEP 7. Стандартные блоки, принимаемые для замены стандартных блоков STEP 5, также обрабатывают числа с плавающей запятой в формате STEP 7 (тип данных REAL).

Таблица 32.3 Преобразование адресов при конвертировании программы

STEP 5	STEP 7
DL [n]	DBB [2n]
DR [n]	DBB [2n+1]
DW [n]	DBW [2n]
DD [n]	DBD [2n]
D [(n).0..7]	DBX [(2n+1).0..7]
D [(n).8..15]	DBX [(2n).0..7]

Если Вы в Вашей программе для STEP 5 самостоятельно набрали данные в формате чисел с плавающей запятой или, если Вы получили их от других устройств, например, посредством коммуникаций, то Вы должны преобразовать представление этих данных (чисел с плавающей запятой) в STEP 5 в тип данных REAL.

32.4 Последующее редактирование

32.4.1 Создание проекта в STEP 7

Для завершения процесса конвертирования Вы должны создать проект STEP 7, который должен соответствовать по структуре Вашей системе назначения (PLC) (если Вы еще не создали такого проекта во время ознакомления с адресацией модулей S7). Если необходимо изменить адреса модулей, параметризацию модулей или изменить параметры работы (execution properties) CPU, Вам необходимо выполнить конфигурирование оборудования (то есть, полностью установить проект). Если установки по умолчанию для параметров модулей не могут быть изменены, то достаточно установить независимую от модулей программу (module-independent program).

Итак для создания проекта:

- Вы создаете станцию (S7-300 или S7-400), открываете объект *Hardware (Оборудование),* после чего конфигурируете станцию. Также с помощью утилиты конфигурирования оборудования Вы устанавливаете свойства CPU (например, номера OB прерывания). Вместе с CPU утилита SIMATIC Manager создает также разделы для объектов следующего уровня.
- Выделив объект *Sources (Исходные),* Вы с помощью опций меню: *Insert -> External Sources File... (Вставка -> Внешний исходный файл...)* вносите созданный файл *name*AC.AWL в раздел (в папку) для исходной программы.
- Если в Вашей программе используются стандартные S5-блоки, откройте библиотеку *S5/S7 Converting Blocks (Блоки для S5/S7-преобразования)* в разделе стандартной библиотеки *Standard Library* и

скопируйте в автономный раздел *Blocks (Блоки)* Вашего проекта стандартные S7-блоки, отмеченные конвертером в списке блоков звездочкой. Если Вы используете системные S7-блоки в конвертированной программе (например, SFC 20 BLKMOV), то откройте библиотеку *System Function Blocks (Системные функциональные блоки)* и скопируейте используемые системные блоки, в автономный раздел *Blocks (Блоки)* Вашего проекта.

- Если в Вашей программе используется символьная адресация, откройте (пустую) таблицу символов *Symbols* и выберите с помощью опций меню: *Symboltable -> Import... (Таблица символов -> Импорт...)* конвертированные символы *name*S7.SEQ для программы.

Выполнив указанные подготовительные пункты, Вы можете теперь с помощью редактора обработать исходный файл программы перед тем, как начнете процесс ее компиляции (Вы можете уменьшить число сообщений об ошибках, если Вы выполните все корректировки до компиляции программы).

32.4.2 Неконвертертируемые функции

После выполнения конвертирования программы обычно требуется редактирование исходного файла программы. Корректировка должна выполняться в отношении инструкций, перечисленных в таблице 32.4.

Таблица 32.4 Неконвертируемые функции

Функции в STEP 5	**Примечание**
Функции загрузки (load) и пересылки (transfer) с системными данными с абсолютными адресами	 Заменяются системными функциями Должны заменяться новой программой
Операции с регистрами (LIR, TIR, LDI, TDI, MBA, MAB, MSA, MAS, MSB, MBR, ABR, ACR)	Должны заменяться новой программой
Функции пересылки блока (TNB, TNW, TXB, TXW)	Должны быть заменены системной функцией SFC 20 BLKMOV
Функции DO DO DW, DO FW DO RS	 Конвертируется Должна заменяться новой программой
Вызов специальных функций	Заменяется специальными функциями SFC
Функции LIM, SIM, IAE, RAE	Могут заменяться функциями из диапазона SFC 39 ... SFC 42
"Семафорные" функции (Semafore functions: SED, SEE, TSC, TSG)	Нет замещения
Другие функции (IAI, RAI, ASM, UBE)	Нет замещения

32.4.3 Изменение адресов

Вопрос изменения адресов особенно касается входных и выходных модулей. Вы должны адаптировать систему доступа в конкретных условиях к входам и выходам, так же как и систему прямого доступа к периферии (I/O) к новым значениям адресов для модулей. Вы должны выполнить эту адаптацию в файле STEP 5 перед его конвертированием (если в системе STEP 5 поддерживается требуемый объем адресного пространства), или Вы можете заменить абсолютные адреса в исходном S7-файле с помощью функции замены "Replace", встроенной в редактор (будьте внимательны, особенно в случае, если "старые" и "новые" адресные области перекрываются).

В случае, если при программировании Вы используете символьную адресацию, Вы можете также создать исходную программу с символьными адресами; измените абсолютные адреса в таблице символов и затем вновь перекомпилируйте ее. Порядок выполнения действий предлагается ниже:

- Требуется, чтобы в распоряжении пользователя была таблица символов с символами для всех абсолютных адресов, которые должны быть изменены; требуется также, чтобы программа была скомпилирована без ошибок (блоки, содержащие абсолютные адреса, должны быть доступны в скомпилированной форме).
- Установите редактор на использование символьной адресации: посредством опций меню *Options -> Customize (Опции -> Установки пользователя)* откройте диалоговое окно, в котором на вкладке "Editor" ("Редактор") выберите опцию для включения режима символов *Symbolic Representation (Символьное представление)*.
- В редакторе посредством выбора опций меню: *File -> Generate Source File... (Файл -> Создать исходный файл...)* начните создание нового исходного файла. После ввода имени файла выберите все блоки в диалоговом окне, которые необходимы для исходного файла с символьной адресацией. Теперь новый исходный файл будет содержать инструкции с символьными адресами.
- Далее, измените все абсолютные адреса в таблице символов *Symbols* со "старых" значений для версии S5 на "новые" значения для версии S7.
- Если теперь Вы скомпилируете новый исходный файл, то в скомпилированных блоках будут содержаться новые абсолютные адреса.

32.4.4 Косвенная адресация

Конвертер может распознавать также косвенную адресацию в функциях DO MW и DO DW в системе STEP 7. Тем не менее, необходимо при этом преобразовать (конвертировать) указатель в формат STEP 7, что, с учетом буферизации содержимого аккумулятора и слова состояния, ведет к возрастанию требуемого объема памяти.

При правильном программировании обычно Вы можете использовать косвенную адресацию - или косвенную адресацию посредством памяти, или косвенную адресацию посредством регистра - с использованием небольшого количества инструкций, получив в результате программу с ясной структурой. Если косвенная адресация используется часто, то программирование в системе STEP 7 имеет определенные преимущества.

Программирование в системе STEP 7 имеет значительные преимущества, в случае, если косвенная адресация используется максимально часто.

- Косвенная адресация таймеров, счетчиков и блоков
  Такая адресация конвертируется в косвенную адресацию посредством памяти с использованием одного слова в области временных локальных данных.
- Косвенная адресация блоков
  Размещение номеров новых блоков не может приниматься в расчет (корректируется в ручном режиме).
- Косвенная адресация
  Конвертируется в побитовом и "пословном" режиме посредством AR1, с буферированием содержимого слова состояния STW и аккумуляторов Accu 1 и Accu 2 во временных локальных данных (см. ниже).
- Косвенная адресация посредством регистра BR
  Конвертирование невозможно. Корректируется в ручном режиме с использованием адресных регистров.
- Другие варианты косвенной адресации
  Должны корректироваться в ручном режиме.

Функции перехода (Jump functions)	Заменяются функцией распределителя переходов SPL
Функции сдвига (Shift functions)	Заменяются функцией сдвига с числом позиций, указываемых в аккумуляторе Accum 2
Функции TNB, TNW	Заменяются системной функцией SFC 20 BLKMOV с "переменной" указатель ANY.
Функции LIR, TIR	Прямая замена невозможна
Функции декрементирования или инкрементирования	Прямая замена невозможна

Конвертер изменяет косвенную адресацию с DO MW и DO DW двоичных логических операций, функций с памятью и функций загрузки (load) и пересылки (transfer) для STEP 7 -программы. Указатель из программы для STEP 5 должен быть изменен на внутризонный указатель в формате STEP 7 (с буферированием содержимого аккумулятора и слова состояния). В результате получается длинная цепочка инструкций (см. пример ниже).

Если Вы в Вашей программе изменили большое число косвенных адресов, то ручной режим конвертирования программы может дать определенное преимущество.

У пользователя есть неограниченный доступ к двум адресным регистрам AR1 и AR2 как индексным регистрам (в функциях FC). Вы можете также адресовать меркеры или данные в режиме косвенной адресации посредством памяти как в системе STEP 5, но Вам при этом потребуется на индексный регистр одно двойное слово вместо одного машинного слова.

В таблице 32.5 представлен пример, в котором в первом столбце расположена программа для STEP 5, выполняющая сравнение значения входного слова с массивом данных, состоящим из битовых элементов; в случае совпадения сравниваемых значений устанавливается меркер. Во втором столбце содержится программа после конвертирования. Используя оба адресных регистра, Вы можете записать новую программу для прямого сравнения значений, для которой потребуется значительно меньшее число инструкций (см. третий столбец таблицы).

Сначала адресные регистры загружаются с указателями (примите во внимание побайтную адресацию данных!). Доступ к словам и к меркерам организован с помощью косвенной адресации посредством регистра. После каждой операции сравнения значение в адресном регистре AR1 инкрементируется на 2 байта, а адресный регистр AR 2 инкрементируется на 1 бит (конвертирование в байтовый адрес пропущено). В рассматриваемом примере указатель на слова данных используется в соответствии с критерием "точки разрыва" ("break criterion"), как в системе STEP 5; в такой точке система STEP 7 организует циклический переход LOOP.

32.4.5 Доступ к "чрезмерно большим" блокам данных

Доступ к "чрезмерно большим" блокам данных, то есть, доступ к данным, которые имеют значение байтового адреса больше 255, в системе STEP 5 выполнялось с помощью абсолютной адресации. При этом вычислялся начальный адрес блока данных, добавлялось адресное смещение и производилось обращение к данным или прямое (посредством функций LIR/TIR), или с помощью регистра BR (посредством функций LRW/TRW).

В системе STEP 7 Вы можете назначать адреса данных непосредственно (прямым указанием адреса) в разрешенном диапазоне вплоть до значения 8095 (в системе S7-300) и 32767 (в системе S7-400).

Следовательно, заменяя доступ с применением абсолютной адресации при переходе к системе STEP 7, Вы можете использовать адреса данных в "обычных" STL-инструкциях.

32.4.6 Использование абсолютных адресов

Абсолютная адресация может использоваться, если необходимо обрабатывать абсолютные адреса в памяти в системе STEP 5, если Вы назначаете адреса данных в "чрезмерно больших" блоках данных, или если Вы используете косвенную адресацию посредством BR-регистра, или если Вы используете передачу (transfer) блока. Доступ к абсолютным

Таблица 32.5 Конвертирование программы с косвенной адресацией

Программа в STEP 5	Программа после конвертирования	Программа после оптимизации
FB 174 Name : COMP	FUNCTION FC 4 VOID NAME: COMP VAR_TEMP conv_accum1 : dword; conv_accum2 : dword; conv_stw : word; END_VAR BEGIN NETWORK	FUNCTION FC 4 VOID NAME: COMP BEGIN
:L KB 20 :T DW 2 :L KB 50 :T DW 3 LOOP :L IW 10	L 20; T DBW 4; L 50; T DBW 6; LOOP: L IW 10;	LAR1 P#40.0; LAR2 P#50.0; LOOP: L IW 10;
	T conv_accum1; L STW; T conv_stw; L DBB 5; SLW 4; LAR1; L conv_stw; T STW; L conv_accum1;	
:DO DW 2 :L DW 0	L DBW[AR1,P#0.0];	L DBW[AR1,P#0.0];
:>F	>I;	>I;
	T conv_accum1; TAK; T conv_accum2; L STW; T conv_stw; L DBB 6; SLW 5; SRW 5; L DBB 7; SLW 3; OW; LAR1; L conv_stw; T STW; L conv_accum2; L conv_accum1;	
:DO DW 3 := F 0.0	= M[AR1,P#0.0];	= M[AR2,P#0.0];
:L DW 2 :I 1 :T DW 2 :L KB 100 :>F :JC =END :L DL 3 :I 1 :T DL 3 :L KB 8 :<F :JC =LOOP :L DR 3 :I 1 :T DW 3 :JU =LOOP END :NOP 0 :BE	L DBW 4; INC 1; T DBW 4; L 100; >I; JC END; L DBB 6; INC 1; T DBB 6; L 8; <I; JC LOOP; L DBB 7; INC 1; T DBW 6; JU LOOP; END: NOP 0; END_FUNCTION	+AR1 P#2.0; CAR1; L P#200.0; >D; JC END; +AR2 P#0.1; JU LOOP; END: NOP 0; END_FUNCTION

адресам далее не поддерживается в STEP 7; адресный счетчик STEP (для "связанных" операций) удаляется без замены.

Доступ к данным с адресами в "чрезмерно больших" блоках данных обеспечивается в STEP 7 непосредственно с помощью "обычных" инструкций. В связи с этим, вычисление адреса в блоках данных также пропускается. Очевидное решение для косвенной адресации с помощью BR-регистра заключается в использовании "косвенной адресации посредством регистра" и , если необходимо, то "межзонной адресации".

Системная функция SFC 20 BLKMOV заменяет операции в S5-программе по пересылке блоков (transfer). Вы определяете переменные или области памяти, которые должны быть скопированы непосредственно как параметры. Если Вы хотите переопределять исходные области памяти или области назначения во время выполнения программы, то Вы должны будете использовать "переменную" ANY-указатель в качестве фактического параметра.

32.4.7 Инициализация параметров

Конвертер принимает фактические параметры, которые используются при вызове блока без изменения. Если у Вас имеются определенные адреса с фактическими параметрами, Вы можете проверить их и, если необходимо, изменить.

Примеры:

- Определение числа в формате слова (WORD):
  Такая адресация конвертируется в побайтную адресацию.
- Определение адреса (I/O):
  При конвертировании должен использоваться новый адрес модуля.
- Передача (transfer) блока:
  При конвертировании должен использоваться новый номер модуля.

32.4.8 Специальные функции организационных блоков

В STEP 7 Вы можете использовать системные функции или STL-инструкции для замены организационных блоков специальными функциями (см. табл. 32.6). Некоторые функции пропускаются полностью (например, такие, как страничная адресация ("page addressing"), доступ к системной программе).

32.4.9 Обработка ошибок

Способ сигнализации о нарушении диапазона допустимых значений посредством битов состояния OV и OS в STEP 7 аналогичен способу

Таблица 32.6 Конвертирование специальных функций организационных блоков

Функция	115U	135U	155U	Заменяется в STEP 7
Байт условного кода для процесса (Process condition code byte)	-	110	-	Последовательность инструкций
Аккумуляторы процесса (Process accumulators)	-	111-113	131-133	Последовательность инструкций
Обработка прерываний (Handle interrupts)	-	120-123	122 141-143	SFC 39 DIS_IRT, SFC 40 EN_IRT, SFC 41 DIS_AIRT, SFC 42 EN_AIRT
Активация задания таймера (Activate a timer job)	-	151	151	SFC 28 SET_TINT, SFC 29 CAN_TINT, SFC 30 ACT_TINT, SFC 31 QRY_TINT
Обработка прерывания с задержкой обработки (Handle a delay interrupt)	-	153	153	SFC 32 SRT_DINT, SFC 33 QRY_DINT, SFC 34 CAN_DINT
Переменная времени ожидания (Variable waiting time)	160	-	-	SFC 43 WAIT
Удаление блока (Delete block)	-	-	124	Блок данных: SFC 23 DEL_DB
Создание блока (Create block)	125	-	125	Блок данных: SFC 22 CREAT_DB
Считывание стека блоков данных (Read block stack)	-	170	-	- отсутствует -
Проверка блока данных (Test data block)	-	181	-	SFC 24 TEST_DB
Обращение к блоку данных (Data block access)	-	180	-	- отсутствует -
Копирование блоков данных (Copy data blocks)	183, 184	254, 255	254, 255	SFC 20 BLKMOV (области данных)
Копирование области данных (Copy data areas)	182 190-193	182 190-193	-	SFC 20 BLKMOV
Задание и считывание времени суток (Set & read time-of-day)	-	150	121, 150	SFC 0 SET_CLK, SFC 1 READ_CLK
Данные по циклу (Cycle statistics)	-	152	-	Стартовая информация OB 1, SFC 6 RD_SINFO
Считывание информации о состоянии (Read status inf)	-	228	-	Стартовая информация, SFC 6 RD_SINFO

Таблица 32.6 (*продолжение*) Конвертирование специальных функций организационных блоков

Функция	115U	135U	155U	Заменяется в STEP 7
Коммуникации мульти-процессорного режима (Multiprocessor communications)	-	200-205	200-205	Замена: GD-коммуникации
Сравнение типов перезапуска (Compare restart types)	-	223	223	- отсутствует -
Флаги передачи межпро-цессорных коммуникаций (Transfer interprocessor communication flags)	-	224	-	GD-коммуникации
Задание времени цикла (Set cycle time)	-	221	-	Параметризация CPU
Перезапуск контроля цикла (Cycle time triggering)	-	222	31, 222	SFC 43 RE_TRIGR
Передача образа процесса (Transfer process images)	254, 255	-	126	SFC 26 UPDAT_PI, SFC 27 UPDAT_PO
Кольцевой счетчик (регистр) (Counter loop)	-	160-163	-	Последовательность инструкций
Заполнение знаком (Sign extension)	220	220	-	Последовательность инструкций
Постраничный доступ (Page accesses)	-	216-218	-	- отсутствует -
Доступ к системной программе (System program access)	-	226, 227	-	- отсутствует -
Регистр сдвига процесса (Process shift register)	-	240-248	-	- отсутствует -
Обработка блоков (Handling blocks)	-	230-237	-	Коммуникационные блоки SFC
ПИД-алгоритм (PID algorithm)	251	250-251	-	Стандартные блоки для ПИД-регулирования
Обслуживание работы системы (Execute system service)	250	-	-	(см. выше "Контроль параметров выполнения программы")

сигнализации в STEP 5, но при этом существуют небольшие отличия. Если Вы проверяете биты состояния OV и OS, обращайте внимание на точность выполнения функций для соответствующих инструкций (например, арифметических функций).

Почти все системные функции SFC сигнализируют об ошибках посредством функционального значения (возвращаемого значения функции) RET_VAL.

Для обработки ошибок в STEP 7 существуют организационные блоки: для обработки синхронных ошибок - блоки OB 121, OB 122 и для обработки асинхронных ошибок - блоки с номерами от OB 80 до OB 87. В таблице 32.7 Вы можете видеть, как заменяются блоки обработки ошибок при переводе программы из STEP 5 в STEP 7.

Таблица 32.7 Конвертирование организационных блоков обработки ошибок

Функция	S5-115	S5-135	S5-155	Замена в S7
Вызов незагруженного блока	19	19	19	OB 121
Задержка квитирования при прямом доступе к I/O модулям	23	23	23	OB 122
Квитирование при обновлении образа процесса	24	24	24	OB 122
Ошибки адресации	-	25	25	OB 122
Превышено максимальное время цикла	26	26	26	OB 80
Ошибки подстановки	27	27	27	-
Остановка по условию (Conditional STOP)	-	28	-	-
Задержка квитирования в случае входного байта IB 0	-	-	28	OB 85
Некорректный код операции	-	29	-	STOP
Задержка квитирования при прямом доступе к области периферийных входов/выходов (I/O)	-	-	29	OB 122
Некорректные параметры	-	30	-	-
Ошибки проверки на четность или подтверждения в случае доступа к пользовательской памяти	-	-	30	OB 122
Групповые ошибки специальной функции	-	31	-	-
Ошибки передачи в блоках данных	32	32	32	OB 121
Ошибки таймера при запуске, управляемом по времени	33	33	33	OB 80
Отказ батареи питания	34	-	-	OB 81
Ошибки контроллера	-	34	-	-
Ошибки при создании блока данных	-	-	34	(SFC)
I/O ошибки	35	-	-	OB 86
Ошибки интерфейса	-	35	-	OB 84
Ошибки самоконтроля (Self-test errors)	-	-	36	-

33 Библиотеки блоков

Базовое ПО STEP 7 включает в себя стандартную библиотеку *Standard Library,* которая содержит следующие разделы для блоков различных типов:

- Организационные блоки (OB);
- Системные функциональные блоки (SFB);
- Функциональные IEC-блоки (загружаемые IEC-функции);
- Блоки для S5-S7-преобразования (загружаемые функции преобразования);
- Блоки для TI-S7-преобразования (загружаемые функции преобразования);
- Блоки ПИД-управления (функции для автоматического управления);
- Коммуникационные блоки (DP-функции).

Вы можете копировать блоки или интерфейсы системных функций или системных функциональных блоков в свой собственный проект или библиотеки.

33.1 Организационные блоки (OB)

(Prio = приоритетный класс, принимаемый по умолчанию)

OB	Prio	Назначение	
1	1	Основная программа	(Main program)
10	2	Прерывание по времени суток 0	(Time-of-day interrupt 0)
11	2	Прерывание по времени суток 1	(Time-of-day interrupt 1)
12	2	Прерывание по времени суток 2	(Time-of-day interrupt 2)
13	2	Прерывание по времени суток 3	(Time-of-day interrupt 3)
14	2	Прерывание по времени суток 4	(Time-of-day interrupt 4)
15	2	Прерывание по времени суток 5	(Time-of-day interrupt 5)

OB	Prio	Назначение	
16	2	Прерывание по времени суток 6	(Time-of-day interrupt 6)
17	2	Прерывание по времени суток 7	(Time-of-day interrupt 7)
20	3	Прерывание с задержкой обработки 0	(Time-delay interrupt 0)
21	4	Прерывание с задержкой обработки 1	(Time-delay interrupt 1)
22	5	Прерывание с задержкой обработки 2	(Time-delay interrupt 2)
23	6	Прерывание с задержкой обработки 3	(Time-delay interrupt 3)
30	7	Прерывание таймерное 0 (5 с)	(Watchdog interrupt 0 [5 s])
31	8	Прерывание таймерное 1 (2 с)	(Watchdog interrupt 1 [2 s])
32	9	Прерывание таймерное 2 (1 с)	(Watchdog interrupt 2 [1 s])
33	10	Прерывание таймерное 3 (500 мс)	(Watchdog interrupt 3 [500 ms])
34	11	Прерывание таймерное 4 (200 мс)	(Watchdog interrupt 4 [200 ms])
35	12	Прерывание таймерное 5 (100 мс)	(Watchdog interrupt 5 [100 ms])
36	13	Прерывание таймерное 6 (50 мс)	(Watchdog interrupt 6 [50 ms])
37	14	Прерывание таймерное 7 (20 мс)	(Watchdog interrupt 7 [20 ms])
38	15	Прерывание таймерное 8 (10 мс)	(Watchdog interrupt 8 [10 ms])
40	16	Аппаратное прерывание 0	(Hardware interrupt 0)
41	17	Аппаратное прерывание 1	(Hardware interrupt 1)
42	18	Аппаратное прерывание 2	(Hardware interrupt 2)
43	19	Аппаратное прерывание 3	(Hardware interrupt 3)
44	20	Аппаратное прерывание 4	(Hardware interrupt 4)
45	21	Аппаратное прерывание 5	(Hardware interrupt 5)
46	22	Аппаратное прерывание 6	(Hardware interrupt 6)
47	23	Аппаратное прерывание 7	(Hardware interrupt 7)
60	25	Мультипроцессорное прерывание	(Multiprocessor interrupt)
70	25	Ошибка резервирования I/O [1)]	(I/O redundancy error) [1)]
72	28	Ошибка резервирования CPU	(CPU redundancy error)
73	25	Ошибка резервирования коммуникаций	(Communication redundancy error)
80	26	Временная ошибка [1)]	(Time error) [1)]
81	26	Неисправность источника питания [1)]	(Power supply fault) [1)]
82	26	Диагностическое прерывание [1)]	(Diagnostics interrupt) [1)]
83	26	Прерывание вставки/удаления модуля [1)]	(Insert/remove-module interrupt) [1)]
84	26	Неисправность CPU [1)]	(CPU hardware fault) [1)]
85	26	Ошибка приоритетного класса [1)]	(Priority class error) [1)]

OB	Prio	Назначение	
86	26	DP-ошибка [1]	(DP error) [1]
87	26	Ошибка системы связи [1]	(Communication error) [1]
90	29	Обработка в фоновом режиме	(Background processing)
100	27	Полный перезапуск	(Complete restart)
101	27	Перезапуск	(Restart)
102	27	"Холодный" перезапуск	(Cold restart)
121	-	Ошибка программирования	(Programming error)
122	-	Ошибка доступа к I/O	(I/O access error)

[1] Prio = 28 при перезапуске (Prio = приоритетный класс, принимаемый по умолчанию)

33.2 Системные функциональные блоки (SFB)

IEC-функции таймеров и IEC-функции счетчиков

SFB	Имя	Назначение	
0	CTU	Счетчик в режиме прямого счета	(Up counter)
1	CTD	Счетчик в режиме обратного счета	(Down counter)
2	CTUD	Счетчик в режиме прямого/обратного счета	(Up/down counter)
3	TP	Генерация импульса	(Pulse)
4	TON	Задержка включения импульса	(On delay)
5	TOF	Задержка выключения импульса	(Off delay)

Коммуникации посредством сконфигурированных соединений

SFB	Имя	Назначение	
8	USEND	Нескоординированная передача	(Uncoordinated send)
9	URVC	Нескоординированный прием	(Uncoordinated receive)
12	BSEND	Поблочная передача	(Block-oriented send)
13	BRVC	Поблочный прием	(Block-oriented receive)
14	GET	Считывание данных от партнера	(Read data from partner)
15	PUT	Запись данных в партнере	(Write data to partner)
16	PRINT	Печать данных на принтере	(Write data to printer)

Коммуникации посредством сконфигурированных соединений (*продолжение*)

SFB	Имя	Назначение	
19	START	Инициация полного перезапуска партнера	(Initiate complete restart in partner)
20	STOP	Перевод партнера в режим STOP	(Set partner to STOP)
21	RESUME	Инициация перезапуска партнера	(Initiate restart in partner)
22	STATUS	Проверка состояния партнера	(Check status of partner)
23	USTATUS	Прием данных о состоянии партнера	(Receive status of partner)

SFC	Имя	Назначение	
62	CONTROL	Проверка состояния коммуникаций	(Check communications status)

Встроенные функции CPU 312/314/614

SFB	Имя	Назначение	
29	HS_COUNT	"Быстрый" счетчик	(High-speed counter)
30	FREQ_MES	Измеритель частоты	(Frequency meter)
38	HSC_A_B	Управление счетчиком "A/B"	(Control "Counter A/B")
39	POS	Функция позиционирования	(Control "Positioning")
41	CONT_C	Функция непрерывного управления	(Continuous closed-loop control)
42	CONT_S	Функция пошагового управления	(Step-action control)
43	PULSEGEN	Генерация импульса	(Generate pulse)

SFC	Имя	Назначение	
63	AB_CALL	Функция вызова блока скомпонованного кода	(Call assembler block)

Функции системной диагностики

SFC	Имя	Назначение	
6	RD_SINFO	Считывание стартовой информации	(Read start information)
51	RDSYSST	Считывание информации подсписка SYS ST	(Read SYS ST sublist)
52	WR_USMSG	Ввод в диагностический буфер	(Entry in the diagnostics buffer)

Генерация сообщений, связанных с блоками

SFB	Имя	Назначение	
33	ALARM	Сообщения с подтверждением	(Messages with acknowledgment display)
34	ALARM_8	Сообщения без сопутствующих значений	(Messages without accompanying values)
35	ALARM_8P	Сообщения с сопутствующими значениями	(Messages with accompanying values)
36	NOTIFY	Сообщения без подтверждения	(Messages without acknowledgment display)
37	AR_SEND	Посылка архивных данных	(Send archive data)

SFC	Имя	Назначение	
9	EN_MSG	Разрешить сообщения	(Enable messages)
10	DIS_MSG	Заблокировать сообщения	(Disable messages)
17	ALARM_SQ	Сообщения, которые могут быть квитированы	(Messages that can be acknowledged)
18	ALARM_S	Сообщения, которые всегда квитируются	(Messages that are always acknowledged)
19	ALARM_SC	Определение состояния квитирования	(Determine acknowledgment status)

Встроенные часы CPU и измерение времени наработки

SFC	Имя	Назначение	
0	SET_CLK	Установить часы	(Set clock)
1	READ_CLK	Считать значение времени	(Read clock)
2	SET_RTM	Установить измеритель времени наработки	(Set run-time meter)
3	CTRL_RTM	Скорректировать измеритель времени наработки	(Modify run-time meter)
4	READ_RTM	Считать значение измерителя времени наработки	(Read run-time meter)
48	SNC_RTCB	Синхронизировать вторичные (ведомые) часы	(Synchronize slave clocks)
64	TIME_TCK	Считать системное время	(Read system time)

Генератор последовательностей Drum

SFC	Имя	Назначение	
32	DRUM	Функция генератора последовательностей	(Drum)

Функции копирования и функции для работы с блоками

SFC	Имя	Назначение	
20	BLKMOV	Копирование данных из области памяти	(Copy data area)
21	FILL	Инициализация области памяти	(Pre-assign data area)
22	CREAT_DB	Создание блока данных	(Generate data block)
23	DEL_DB	Удаление блока данных	(Delete data block)
24	TEST_DB	Тестирование блока данных	(Test data block)
25	COMPRESS	"Сжатие" памяти	(Compress memory)
44	REPL_VAL	Введение заменяющего значения	(Enter substitute value)
81	UBLKMOV	Копирование данных "без зазоров"	(Copy Data area without gaps)

Адреса модулей

SFC	Имя	Назначение	
5	GADR_LGC	Определение логического адреса	(Determine logical address)
49	LGC_GADR	Определение слота	(Determine slot)
50	RD_LGADR	Определение всех логических адресов	(Determine all logical addresses)

Распределенная периферия (I/O)

SFC	Имя	Назначение	
7	DP_PRAL	Инициация аппаратного прерывания	(Initiate haidware interrupt)
11	DPSYN_FR	Синхронизация групп устройств	(SYNC/FREEZE)
12	D_ACT_DP	Деактивация/активация ведомого (slave) DP-устройства	(Deactivate or activate DP slave)
13	DPNRM_DG	Считывание диагностических данных	(Read diagnostics data)
14	DPRD_DAT	Считывание данных из ведомого (slave) DP-устройства	(Read slave data)
15	DPWR_DAT	Запись данных в ведомое (slave) DP-устройство	(Write slave data)

Управление программой

SFC	Имя	Назначение	
43	RE_TRIGR	Перезапуск контроля времени цикла	(Retrigger cycle time monitor)
46	STP	Переход в режим STOP	(Change to STOP state)
47	WAIT	Ожидание в течение времени задержки	(Wait for delay time)

Пересылка записей данных

SFC	Имя	Назначение	
54	RD_DPARM	Считывание предопределенного параметра	(Read predefined parameter)
55	WR_PARM	Запись динамического параметра	(Write dynamic parameter)
56	WR_DPARM	Запись предопределенного параметра	(Write predefined parameter)
57	PARM_MOD	Параметризация модуля	(Parameterize module)
58	WR_REC	Запись записи данных	(Write data record)
59	RD_REC	Считывание записи данных	(Read data record)

Обновление образа процесса

SFC	Имя	Назначение	
26	UPDAT_PI	Обновление таблицы входов образа процесса	(Update process-image input table)
27	UPDAT_PO	Обновление таблицы выходов образа процесса	(Update process-image output table)
79	SET	Установка битовых массивов I/O	(Set I/O bit field)
80	RSET	Сброс битовых массивов I/O	(Reset I/O bit field)

События прерываний

SFC	Имя	Назначение	
28	SET_TINT	Установка прерывания по времени суток	(Set time-of-day interrupt)
29	CAN_TINT	Отмена прерывания по времени суток	(Cancel time-of-day interrupt)
30	ACT_TINT	Активация прерывания по времени суток	(Activate time-of day interrupt)
31	QRY_TINT	Запрос прерывания по времени суток	(Query time-of-day interrupt)

События прерываний (*продолжение*)

SFC	Имя	Назначение	
32	SRT_DINT	Запуск прерывания с задержкой обработки	(Start time-delay interrupt)
33	CAN_DINT	Отмена прерывания с задержкой обработки	(Cancel time-delay interrupt)
34	QRY_DINT	Запрос прерывания с задержкой обработки	(Query time-delay interrupt)
35	MP_ALM	Запуск предупреждения для мультипроцессорного режима	(Trigger multiprocessor alarm)
36	MSK_FLT	Маскирование синхронных ошибок	(Mask synchronous errors)
37	DMSK_FLT	Демаскирование синхронных ошибок	(Unmask synchronous errors)
38	READ_ERR	Считывание регистра состояния событий	(Read event status register)
39	DIR_IRT	Блокировка обработки асинхронных ошибок	(Disable asynchronous errors)
40	EN_IRT	Разблокировка обработки асинхронных ошибок	(Enable asynchronous errors)
41	DIS_AIRT	Задержка обработки асинхронных ошибок	(Delay asynchronous errors)
42	EN_AIRT	Разрешение обработки асинхронных ошибок	(Enable asynchronous errors)

Коммуникации посредством несконфигурированных соединений

SFC	Имя	Назначение	
65	X_SEND	Передача данных партнеру по связи вне локальной станции	(Send data externally)
66	X_RCV	Прием данных от партнера по связи вне локальной станции	(Receive data externally)
67	X_GET	Чтение данных от партнера по связи вне локальной станции	(Read data externally)
68	X_PUT	Запись данных в партнере по связи вне локальной станции	(Write data externally)
69	X_ABORT	Отмена связи вне локальной станции	(Abort external connection)
72	I_GET	Чтение данных от партнера по связи в пределах локальной станции	(Read data internally)
73	I_PUT	Запись данных в партнере по связи в пределах локальной станции	(Write data internally)
74	I_ABORT	Отмена связи в пределах локальной станции	(Abort internal connection)

Коммуникации посредством глобальных данных

SFC	Имя	Назначение	
60	GD_SND	Посылка GD-пакета	(Send GD packet)
61	GD_RCV	Прием GD-пакета	(Receive GD packet)

H CPU

SFC	Имя	Назначение	
90	H_CTRL	Управление режимами работы H-CPU	(Control Operating Modes on H-CPU)

33.3 Функциональные IEC-блоки

Функции сравнения

FC	Имя	Назначение	
9	EQ_DT	Проверка данных формата DT на равенство	(Compare DT for equal to)
28	NE_DT	Проверка данных формата DT на неравенство	(Compare DT for not equal to)
14	GT_DT	Сравнение данных формата DT по критерию "больше чем"	(Compare DT for greater than)
12	GE_DT	Сравнение данных формата DT по критерию "больше или равно"	(Compare DT for greater than or equal to)
23	LT_DT	Сравнение данных формата DT по критерию "меньше чем"	(Compare DT for less than)
18	LE_DT	Сравнение данных формата DT по критерию "меньше или равно"	(Compare DT for less than or equal to)
10	EQ_STRNG	Проверка данных формата STRING на равенство	(Compare STRING for equal to)
29	NE_STRNG	Проверка данных формата STRING на неравенство	(Compare STRING for not equal to)
15	GT_STRNG	Сравнение данных формата STRING по критерию "больше чем"	(Compare STRING for greater than)
13	GE_STRNG	Сравнение данных формата STRING по критерию "больше или равно"	(Compare STRING for greater than or equal to)
24	LT_STRNG	Сравнение данных формата STRING по критерию "меньше чем"	(Compare STRING for less than)
19	LE_STRNG	Сравнение данных формата STRING по критерию "меньше или равно"	(Compare STRING for less than or equal to)

Функции даты и времени

FC	Имя	Назначение	
3	D_TOD_DT	Объединение данных типов DATE и TOD в значение формата DT	(Combine DATE and TOD to DT)
6	DT_DATE	Извлечение данных формата DATE из данных формата DT	(Extract DATE from DT)
7	DT_DAY	Извлечение данных о дне недели из данных формата DT	(Extract day-of-the-week from DT)
8	DT_TOD	Извлечение данных о времени суток из данных формата DT	(Extract TOD from DT)
33	S5TI_TIM	Конвертация данных формата S5TIME в значение формата TIME	(Convert S5TIME to TIME)
40	TIM_S5TI	Конвертация данных формата TIME в значение формата S5TIME	(Convert TIME to S5TIME)
1	AD_DT_TM	Прибавление значения формата TIME к значению формата DT	(Add TIME to DT)
35	SB_DT_TM	Вычитание значения формата TIME из значения формата DT	(Subtract TIME from DT)
34	SB_DT_DT	Вычитание значения формата DT из значения формата DT	(Subtract DT from DT)

Математические функции

FC	Имя	Назначение	
22	LIMIT	Функция-ограничитель	(Limiter)
25	MAX	Выбор максимального значения из трех значений	(Maximum selection)
27	MIN	Выбор минимального значения из трех значений	(Minimum selection)
26	SEL	Функция "двоичный переключатель"	(Binary selection)

Функции для обработки данных типа STRING

FC	Имя	Назначение	
21	LEN	Возврат длины строки символов	(Length of a STRING)
20	LEFT	Возврат части строки слева	(Left section of a STRING)
32	RIGHT	Возврат части строки справа	(Right section of a STRING)
26	MID	Возврат части строки из середины	(Middle section of a STRING)
2	CONCAT	"Сшивание" двух строк символов	(Concatenate STRINGS)
17	INSERT	Вставка строки символов	(Insert STRING)
4	DELETE	Удаление части строки символов	(Delete STRING)
31	REPLACE	Замена части строки символов	(Replace STRING)

Функции для обработки данных типа STRING (*продолжение*)

FC	Имя	Назначение	
11	FIND	Поиск строки символов	(Find STRING)
16	I_STRNG	Конвертация данных формата INT в формат STRING	(Convert INT to STRING)
5	DI_STRNG	Конвертация данных формата DINT в формат STRING	(Convert DINT to STRING)
30	R_STRNG	Конвертация данных формата REAL в формат STRING	(Convert REAL to STRING)
38	STRNG_I	Конвертация данных формата STRING в формат INT	(Convert STRING to INT)
37	STRNG_DI	Конвертация данных формата STRING в формат DINT	(Convert STRING to DINT)
39	STRNG_R	Конвертация данных формата STRING в формат REAL	(Convert STRING to REAL)

33.4 Блоки для S5-S7-преобразования

Арифметические операции для чисел в формате с плавающей запятой

FC	Имя	Назначение	
61	GP_FPGP	Конвертация данных из формата с фиксированной запятой в формат с плавающей запятой	(Convert fixed-point to floating-point)
62	GP_GPFP	Конвертация данных из формата с плавающей запятой в формат с фиксированной запятой	(Convert floating-point to fixed-point)
63	GP_ADD	Сложение чисел в формате с плавающей запятой	(Add floating-point numbers)
64	GP_SUB	Вычитание чисел в формате с плавающей запятой	(Subtract floating-point numbers)
65	GP_MUL	Перемножение чисел в формате с плавающей запятой	(Multiply floating-point numbers)
66	GP_DIV	Деление чисел в формате с плавающей запятой	(Divide floating-point numbers)
67	GP_VGL	Сравнение чисел в формате с плавающей запятой	(Compare floating-point numbers)
68	GP_RAD	Извлечение квадратного корня из числа в формате с плавающей запятой	(Find the square root of a floating-point number)

Базовые функции

FC	Имя	Назначение	
85	ADD_32	Сложение 32-разрядных чисел в формате с фиксированной запятой	(32-bit fixed-point adder)
86	SUB_32	Вычитание 32-разрядных чисел в формате с фиксированной запятой	(32-bit fixed-point subtractor)
87	MUL_32	Перемножение 32-разрядных чисел в формате с фиксированной запятой	(32 bit fixed-point multipher)
88	DIV_32	Деление 32-разрядных чисел в формате с фиксированной запятой	(32-bit fixed-point divider)
89	RAD_16	Извлечение квадратного корня из 16-разрядного числа в формате с фиксированной запятой	(16-bit fixed point square root extractor)
90	REG_SCHB	Побитовый сдвиг в регистре	(Bitwise shift register)
91	REG_SCHW	Пословный сдвиг в регистре	(Wordwise shift register)
92	REG_FIFO	Буфер FIFO	(Buffer (FIFO))
93	REG_LIFO	Стек LIFO	(Stack (LIFO))
94	DB_COPY1	Прямое копирование данных из области памяти	(Copy data area (direct))
95	DB_COPY2	Косвенное копирование данных из области памяти	(Copy data area (indirect)
96	RETTEN	Сохранение содержимого "сверхоперативной" памяти (S5-155U)	(Save scratchpad memory (S5-155U))
97	LADEN	Загрузка содержимого "сверхоперативной" памяти (S5-155U)	(Load scratchpad memory (S5-155U))
98	COD_B8	Конвертация данных из формата BCD в двоичный формат (для 8 декад)	(BCD-binary conversion 8 decades)
99	COD_32	Конвертация данных из двоичного формата в формат BCD (для 8 декад)	(Binary-BCD conversion 8 decades)

Функции для обработки сигнала

FC	Имя	Назначение	
69	MLD_TG	Генератор тактовых импульсов	(Clock pulse generator)
70	MLD_TGZ	Генератор тактовых импульсов с функцией таймера	(Clock pulse generator with timer function)
71	MLD_EZW	Начальное значение параметра одинарного сигнала (слово)	(Initial value single blinking wordwise)
72	MLD_EDW	Начальное значение параметра двойного сигнала (слово)	(Initial value double blinking wordwise)
73	MLD_SAMW	Групповой сигнал (слово)	(Group signal wordwise)
74	MLD_SAM	Групповой сигнал	(Group signal)
75	MLD_EZ	Начальное значение параметра одинарного сигнала	(Initial value single blinking)
76	MLD_ED	Начальное значение параметра двойного сигнала	(Initial value double blinking)
77	MLD_EZWK	Начальное значение параметра одинарного сигнала (слово меркеров)	(Initial value single blinking (wordwise) memory bit)
78	MLD_EZDK	Начальное значение параметра двойного сигнала (слово меркеров)	(Initial value double blinking (wordwise) memory bit)
79	MLD_EZK	Начальное значение параметра одинарного сигнала меркера	(Initial value single blinking memory bit)
80	MLD_EDK	Начальное значение параметра двойного сигнала меркера	(Initial value double blinking memory bit)

Встроенные функции

FC	Имя	Назначение	
81	COD_B4	Конвертация данных из формата BCD в двоичный формат (для 4 декад)	(BCD-binary conversion 4 decades)
82	COD_16	Конвертация данных из двоичного формата в формат BCD (для 4 декад)	(Binary-BCD conversion 4 decades)
83	MUL_16	Перемножение 16-разрядных чисел в формате с фиксированной запятой	(16-bit fixed-point multipher)
84	DIV_16	Деление 16-разрядных чисел в формате с фиксированной запятой	(16-bit fixed-point divider)

Аналоговые функции

FC	Имя	Назначение	
100	AE_460_1	Аналоговый входной модуль 460	(Analog input module 460)
101	AR_460_2	Аналоговый входной модуль 460	(Analog input module 460)
102	AR_463_1	Аналоговый входной модуль 463	(Analog input module 463)
103	AE_463_2	Аналоговый входной модуль 463	(Analog input module 463)
104	AE_464_1	Аналоговый входной модуль 464	(Analog input module 464)
105	AE_464_2	Аналоговый входной модуль 464	(Analog input module 464)
106	AE_466_1	Аналоговый входной модуль 466	(Analog input module 466)
107	AE_466_2	Аналоговый входной модуль 466	(Analog input module 466)
108	RLG_AA1	Аналоговый выходной модуль	(Analog output module)
109	RLG_AA2	Аналоговый выходной модуль	(Analog output module)
110	PER_ET1	Распределенные I/O ET 100	(ET 100 distributed I/O)
111	PER_ET2	Распределенные I/O ET 100	(ET 100 distributed I/O)

Математические функции

FC	Имя	Назначение	
112	SINUS	Синус	(Sine)
113	COSINUS	Косинус	(Cosine)
114	TANGENS	Тангенс	(Tangent)
115	COTANG	Котангенс	(Cotangent)
116	ARCSIN	Арксинус	(Arc sine)
117	ARCCOS	Арккосинус	(Arc cosine)
118	ARCTAN	Арктангенс	(Arc tangent)
119	ARCCOT	Арккотангенс	(Arc cotangent)
120	LN_X	Натуральный логарифм	(Natural logarithm)
121	LG_X	Десятичный логарифм	(Logarithm to base 10)
122	B_LOG_X	Логарифм с любым основанием	(Logarithm to any base)
123	E_H_N	Экспоненциальная функция с основанием *e*	(Exponential function with base e)
124	ZEHN_H_N	Экспоненциальная функция с основанием 10	(Exponential function with base 10)
125	A2_H_A1	Экспоненциальная функция с любым основанием	(Exponential function with any base)

33.5 Блоки для TI-S7-преобразования

FB	Имя	Назначение	
80	LEAD_LAG	Алгоритм "опережение/отставание"	(Lead/lag algorithm)
81	DCAT	Дискретное управление прерыванием по времени	(Dirscrete control time interrupt)
82	MCAT	Прерывание по времени для управления двигателем	(Motor control time interrupt)
83	IMC	Сравнение индексов матрицы	(Index matrix comparison)
84	SMC	Сканирование матрицы	(Matrix scanner)
85	DRUM	Маскирование последовательности событий	(Event maskable drum)
86	PACK	Сбор/распределение табличных данных	(Collect/distribute table data)

FC	Имя	Назначение	
80	TONR	Фиксация задержки	(Latching ON delay)
81	IBLKMOV	Прямая передача данных области памяти	(Transfer data area indirectly)
82	RSET	Побитовый сброс образа процесса	(Reset process image bit by bit)
83	SET	Побитовая установка образа процесса	(Set process image bit by bit)
84	ATT	Входное значение в таблице	(Enter value in table)
85	FIFO	Выходное первое значение в таблице	(Output first value in table)
86	TBL_FIND	Нахождение значения в таблице	(Find value in table)
87	LIFO	Выходное последнее значение в таблице	(Output last value in table)
88	TBL	Выполнение обработки таблицы	(Execute table operation)
89	TBL_WRD	Копирование значения из таблицы	(Copy value from the table)
90	WSR	Сохранение данных	(Save datum)
91	WRD_TBL	Совместная обработка элементов таблицы	(Combine table element)
92	SHRB	Побитовый сдвиг в регистре сдвига	(Shift bit in bit shift register)
93	SEG	Распределение битов для 7-сегментного индикатора	(Bit pattern for 7-segment display)

FC	Имя	Назначение	
94	ATH	Конвертация данных из формата ASCII в шестнадцатеричное представление	(ASCII hexadecimal conversion)
95	HTA	Конвертация данных из шестнадцатеричного формата в ASCII-формат	(Hexadecimal-ASCII conversion)
96	ENCO	Установление младшего бита	(Least significant set bit)
97	DECO	Установление бита в слове	(Set bit in word)
98	BCDCPL	Нахождение обратного BCD-числа	(Generate ten's complement)
99	BITSUM	Подсчет установленных битов	(Count set bits)
100	RSETI	Побайтовый сброс отображения выходов PQ процесса	(Reset PQ byte by byte)
101	SETI	Побайтовая установка образа выходов PQ процесса	(Set PQ byte by byte)
102	DEV	Вычисление стандартного отклонения	(Calculate standard deviation)
103	CDT	Таблица связанных данных	(Correlated data tables)
104	TBL_TBL	Объединение таблиц	(Table combination)
105	SCALE	Коэффициенты масштабирования	(Scale values)
106	UNSCALE	Обратные коэффициенты	(Unscale values)

33.6 Блоки ПИД-управления

FB	Имя	Назначение	
41	CONT_C	Непрерывное управление	(Continuous control)
42	CONT_S	Пошаговое управление	(Step control)
43	PULSGEN	Генерация импульса	(Generate pulse)

33.7 Коммуникационные блоки

FC	Имя	Назначение	
1	DP_SEND	Посылка данных	(Send data)
2	DP_RECV	Прием данных	(Receive data)
3	DP_DIAG	Диагностика	(Diagnostics)
4	DP_CTRL	Управление	(Control)

34 Общий обзор STL-инструкций

В общем обзоре, предлагаемом ниже, представлен список инструкций с абсолютными адресами.

Возможны следующие типы адресации:

A I [doubleword]	косвенная адресация посредством памяти MD двойное слово меркеров LD двойное слово локальных данных DBD двойное слово глобальных данных DID двойное слово в экземплярном DB	все адреса
A I [AR1,P#offset] A I [AR2,P#offset] A [AR1,P#offset] A [AR2,P#offset]	внутризонная косвенная адресация посредством регистра AR1 внутризонная косвенная адресация посредством регистра AR2 межзонная косвенная адресация посредством регистра AR1 межзонная косвенная адресация посредством регистра AR2	кроме функций таймеров, счетчиков и блоков
A *#name*	косвенная адресация посредством параметра	все адреса

34.1 Базовые функции

34.1.1 Двоичные логические операции

A	-	операция И (AND) для проверки присутствия уровня "1"
AN	-	операция И (AND) для проверки присутствия уровня "0"
O	-	операция ИЛИ (OR) для проверки присутствия уровня "1"
ON	-	операция ИЛИ (OR) для проверки присутствия уровня "0"
X	-	операция Исключающее ИЛИ (Exclusive OR) для проверки присутствия уровня "1"
XN	-	операция Исключающее ИЛИ (Exclusive OR) для проверки присутствия уровня "0"
-	I	вход
-	Q	выход
-	M	меркер
-	L	бит в области локальных данных
-	T	функция таймера
-	C	функция счетчика
-	DBX	бит в области глобальных данных
-	DIX	бит в экземплярном DB
-	==0	значение результата операции равно нулю
-	<>0	значение результата операции не равно нулю
-	>0	значение результата операции больше нуля
-	>=0	значение результата операции больше нуля или равно нулю
-	<0	значение результата операции меньше нуля
-	<=0	значение результата операции меньше нуля или равно нулю
-	UO	значение результата операции неверно
-	OV	переполнение
-	OS	сохраненное переполнение
-	BR	двоичный результат
A(		операция И (AND) с открывающей скобкой
AN(		операция И (AND) с открывающей скобкой
O(		операция ИЛИ (OR) с открывающей скобкой
ON(		операция ИЛИ (OR) с открывающей скобкой
X(		операция Исключающее ИЛИ (Exclusive OR) с открывающей скобкой
XN(		операция Исключающее ИЛИ (Exclusive OR) с открывающей скобкой
)		закрывающая скобка
O		операция ИЛИ (OR), объединяющая операции И (AND)
NOT		операция отрицания RLO
SET		операция установки RLO
CLR		операция сброса RLO
SAVE		операция фиксации RLO в BR

34.1.2 Операции с памятью

=	-	операция присвоения
S	-	операция установки
R	-	операция сброса
FP	-	положительный фронт сигнала
FN	-	отрицательный фронт сигнала
-	I	вход
-	Q	выход
-	M	меркер
-	L	бит в области локальных данных
-	DBX	бит в области глобальных данных
-	DIX	бит в экземплярном DB

34.1.3 Функции передачи

L	-	операция загрузки (load)
T	-	операция передачи (transfer)
-	IB	входной байт
-	IW	входное слово
-	ID	входное двойное слово
-	QB	выходной байт
-	QW	выходное слово
-	QD	выходное двойное слово
-	MB	байт меркеров
-	MW	слово меркеров
-	MD	двойное слово меркеров
-	LB	байт локальных данных
-	LW	слово локальных данных
-	LD	двойное слово локальных данных
-	DBB	байт глобальных данных
-	DBW	слово глобальных данных
-	DBD	двойное слово глобальных данных
-	DIB	байт в экземплярном DB
-	DIW	слово в экземплярном DB
-	DID	двойное слово в экземплярном DB
-	STW	слово состояния

L	PIB	загрузка (load) периферийного входного байта
L	PIW	загрузка (load) периферийного входного слова
L	PID	загрузка (load) периферийного входного двойного слова
T	PQB	передача (transfer) периферийного выходного байта
T	PQW	передача (transfer) периферийного выходного слова
T	PQD	передача (transfer) периферийного выходного двойного слова
L	T	"обычная" загрузка значения таймера
LC	T	загрузка значения таймера в BCD-коде
L	C	"обычная" загрузка значения счетчика
LC	C	загрузка значения счетчика в BCD-коде
L	*const*	загрузка (load) константы
L	P#..	загрузка (load) указателя
L	P#var	загрузка (load) начального адреса переменной

Функции аккумуляторов

PUSH	сдвиг содержимого аккумуляторов "вперед"
POP	сдвиг содержимого аккумуляторов "назад"
ENT	сдвиг содержимого аккумуляторов 2 и 3 "вперед"
LEAVE	сдвиг содержимого аккумуляторов 3 и 4 "вперед"
TAK	обмен содержимым между аккумуляторами 1 и 2
CAW	обмен содержимым между байтами 0 и 1 аккумулятора 1
CAD	обмен содержимым между всеми байтами аккумулятора 1

34.1.4 Функции таймеров

SP	T	запуск таймера в режиме "управляемого импульса"
SE	T	запуск таймера в режиме "расширенного импульса"
SD	T	запуск таймера в режиме "с задержкой включения"
SS	T	запуск таймера в режиме "с задержкой включения с памятью"
SF	T	запуск таймера в режиме "с задержкой выключения"
R	T	сброс таймера
FR	T	разрешение перезапуска таймера

34.1.5 Функции счетчиков

CU	C	запуск счетчика в режиме "прямой счет"
CD	C	запуск счетчика в режиме "обратный счет"
S	C	установка счетчика
R	C	сброс счетчика
FR	C	разрешение перезапуска счетчика

34.2 Функции для обработки чисел

34.2.1 Функции сравнения

==I	проверка данных формата INT на равенство
<>I	проверка данных формата INT на неравенство
>I	сравнение данных формата INT по критерию "больше чем"
>=I	сравнение данных формата INT по критерию "больше или равно"
<I	сравнение данных формата INT по критерию "меньше чем"
<=I	сравнение данных формата INT по критерию "меньше или равно"
==D	проверка данных формата DINT на равенство
<>D	проверка данных формата DINT на неравенство
>D	сравнение данных формата DINT по критерию "больше чем"
>=D	сравнение данных формата DINT по критерию "больше или равно"
<D	сравнение данных формата DINT по критерию "меньше чем"
<=D	сравнение данных формата DINT по критерию "меньше или равно"
==R	проверка данных формата REAL на равенство
<>R	проверка данных формата REAL на неравенство
>R	сравнение данных формата REAL по критерию "больше чем"
>=R	сравнение данных формата REAL по критерию "больше или равно"
<R	сравнение данных формата REAL по критерию "меньше чем"
<=R	сравнение данных формата REAL по критерию "меньше или равно"

34.2.2 Математические функции

SIN	синус
COS	косинус
TAN	тангенс

ASIN	арксинус
ACOS	арккосинус
ATAN	арктангенс
SQR	нахождение квадрата числа
SQRT	извлечение квадратного корня из числа
EXP	экспонента по основанию *e*
LN	натуральный логарифм

34.2.3 Арифметические функции

+I		сложение двух чисел формата INT
-I		вычитание двух чисел формата INT
*I		умножение двух чисел формата INT
/I		деление двух чисел формата INT
+D		сложение двух чисел формата DINT
-D		вычитание двух чисел формата DINT
*D		умножение двух чисел формата DINT
/D		деление двух чисел формата DINT (целая часть)
MOD		деление двух чисел формата DINT (остаток)
+R		сложение двух чисел формата REAL
-R		вычитание двух чисел формата REAL
*R		умножение двух чисел формата REAL
/R		деление двух чисел формата REAL
+	*const*	сложение с константой
+	P#..	сложение с указателем
DEC	*n*	декрементирование
INC	*n*	инкрементирование

34.2.4 Функции преобразования

ITD	конвертирование данных формата INT в формат DINT
ITB	конвертирование данных формата INT в формат BCD
DTB	конвертирование данных формата DINT в формат DINT
DTR	конвертирование данных формата DINT в формат REAL

BTI	конвертирование данных формата BCD в формат INT
BTD	конвертирование данных формата BCD в формат DINT
	Конвертирование данных формата REAL в формат DINT, при этом происходит:
RND+	округление данных до ближайшего большего целого числа
RND-	округление данных до ближайшего меньшего целого числа
RND	округление данных до ближайшего целого числа
TRUNC	усечение дробной части числа
INVI	нахождение обратного кода двоичного числа формата INT
INVD	нахождение обратного кода двоичного числа формата DINT
NEGI	инвертирование числа формата INT
NEGD	инвертирование числа формата DINT
NEGR	инвертирование числа формата REAL
ABS	нахождение абсолютного значения числа формата REAL

34.2.5 Функции сдвига

SLW	-	побитовый сдвиг влево содержимого младшего слова аккумулятора 1
SLD	-	побитовый сдвиг влево содержимого аккумулятора 1
SRW	-	побитовый сдвиг вправо содержимого младшего слова аккумулятора 1
SRD	-	побитовый сдвиг вправо содержимого аккумулятора 1
SSI	-	побитовый сдвиг со знаком содержимого младшего слова аккумулятора 1
SSD	-	побитовый сдвиг со знаком содержимого аккумулятора 1
RLD	-	циклический сдвиг влево содержимого аккумулятора 1
RRD	-	циклический сдвиг вправо содержимого аккумулятора 1
-	*n*	на *n* позиций
-		на число позиций, указанное в аккумуляторе Accum 2
RLDA	-	циклический сдвиг влево с использованием бита CC1
RRDA	-	циклический сдвиг вправо с использованием бита CC1

34.2.6 Логические функции для слов данных

AW	-	операция И (AND) для слова данных
AD	-	операция И (AND) для двойного слова данных

OW	-	операция ИЛИ (OR) для слова данных
OD	-	операция ИЛИ (OR) для двойного слова данных
XOW	-	операция Исключающее ИЛИ (Exclusive OR) для слова данных
XOD	-	операция Исключающее ИЛИ (Exclusive OR) для двойного слова данных
-	*const*	с константой формата слова данных или двойного слова данных
-		с содержимым аккумулятора Accum 2

34.3 Функции управления в программе

34.3.1 Функции перехода

JU	*метка*	безусловный переход
		Выполняется переход,
JC	*метка*	если RLO = "1"
JCB	*метка*	если RLO = "1" с сохранением RLO
JCN	*метка*	если RLO = "0"
JNB	*метка*	если RLO = "0" с сохранением RLO
JBI	*метка*	если BR = "1"
JNBI	*метка*	если BR = "0"
		Выполняется переход,
JZ	*метка*	если результат = "0"
JN	*метка*	если результат <> "0"
JP	*метка*	если результат > "0"
JPZ	*метка*	если результат >= "0"
JM	*метка*	если результат < "0"
JMZ	*метка*	если результат <= "0"
JUO	*метка*	если результат некорректен
JO	*метка*	переход выполняется при переполнении
JOS	*метка*	переход выполняется при запомненном переполнении
JL	*метка*	распределитель переходов
LOOP	*метка*	циклический переход

34.3.2 Главное управляющее реле MCR

MCRA	активация области MCR
MCRD	деактивация области MCR
MCR(	открытие зоны MCR
)MCR	закрытие зоны MCR

34.3.3 Функции обработки блоков

CALL	FB	вызов функционального блока
CALL	FC	вызов функции
CALL	SFB	вызов системного функционального блока
CALL	SFC	вызов системной функции
UC	FB	безусловный вызов функционального блока
CC	FB	вызов функционального блока по условию
UC	FC	безусловный вызов функции
CC	FC	вызов функции по условию
BEU		безусловное завершение обработки блока
BEC		завершение обработки блока по условию
BE		безусловное завершение обработки блока
OPN	DB	вызов глобального блока данных
OPN	DI	вызов экземплярного блока данных
CDB		обмен данными между регистрами блока
L	DBNO	загрузка (load) номера глобального блока данных
L	DINO	загрузка (load) номера экземплярного блока данных
L	DBLG	загрузка (load) размера глобального блока данных
L	DILG	загрузка (load) размера экземплярного блока данных
NOP	0	нуль-операция
NOP	1	нуль-операция
BLD	*n*	инструкции отображения программы

34.4 Косвенная адресация

LAR1	-	загрузка в адресный регистр AR1 из
LAR2	-	загрузка в адресный регистр AR2 из
-	MD	двойного слова меркеров
-	LD	двойного слова локальных данных
-	DBD	двойного слова глобальных данных
-	DID	двойного слова экземплярных данных
LAR1		загрузка в AR1 содержимого аккумулятора Accum 1
LAR2		загрузка в AR2 содержимого аккумулятора Accum 1
LAR1	AR2	загрузка в AR1 содержимого регистра AR2
LAR1	P#..	загрузка в AR1 указателя
LAR2	P#..	загрузка в AR2 указателя
LAR1	P#var	загрузка в AR1 начального адреса переменной
LAR2	P#var	загрузка в AR2 начального адреса переменной
TAR1	-	передача из адресного регистра AR1
TAR2	-	передача из адресного регистра AR2
-	MD	в двойное слово меркеров
-	LD	в двойное слово локальных данных
-	DBD	в двойное слово глобальных данных
-	DID	в двойное слово экземплярных данных
TAR1	-	передача из адресного регистра AR1 в аккумулятор Accum 1
TAR2	-	передача из адресного регистра AR2 в аккумулятор Accum 1
TAR1	AR2	передача из адресного регистра AR1 в регистр AR2
CAR		обмен данными между регистрами AR1 и AR2
+AR1		прибавить содержимое аккумулятора Accum 1 к содержимому AR1
+AR2		прибавить содержимое аккумулятора Accum 1 к содержимому AR2
+AR1	P#..	прибавить указатель к содержимому AR1
+AR2	P#..	прибавить указатель к содержимому AR2

35 Общий обзор SCL-инструкций и функций

35.1 Операторы

Тип оператора	Наименование	Оператор	Приоритет
Скобки	(*Выражение*)	(,)	1
Арифметические	Степень	**	2
	Унарный плюс, унарный минус (знак)	+, -	3
	Умножение, деление	*,/,DIV,MOD	4
	Сложение, вычитание	+, -	5
Сравнения	Меньше, меньше или равно, больше, больше или равно	<, <=, >, >=	6
	Равно, неравно	=, <>	7
Двоичная логика	Отрицание (унарная операция)	NOT	3
	Логическая операция И (AND)	AND, &	8
	Логическая операция Исключающее ИЛИ (Exclusive OR)	XOR	9
	Логическая операция ИЛИ (OR)	OR	10
Присвоение	Присвоение	:=	11

35.2 Управляющие операторы

IF	Ветвление программы, обусловленное значением булевой переменной
CASE	Ветвление программы, обусловленное значением целой переменной (INT)
FOR	Организация цикла в программе с использованием переменной "счетчик цикла"
WHILE	Организация цикла в программе с проверкой условия выполнения цикла
REPEAT	Организация цикла в программе с проверкой условия завершения цикла
CONTINUE	Прервать текущее выполнение (проход) программы цикла
EXIT	Выход из программы цикла
GOTO	Переход к метке перехода
RETURN	Выход из блока (прекращение выполнения блока)

35.3 Вызов блоков

Функции FC с функциональным значением	*Variable* := FC*x(...);* *Variable* := *FCname(...);*
Системные функции SFC с функциональным значением	*Variable* := SFC*x(...);* *Variable* := *SFCname(...);*
Функции FC без функционального значения	FC*x(...);* *FCname(...);*
Функциональные блоки FB с блоками данных	FBx.DBx(...); *FBname.DBname(...);*
Системные функциональные блоки SFB с блоками данных	SFBx.DBx(...); *SFBname.DBname(...);*
Функциональные блоки FB и системные функциональные блоки SFB как локальные экземпляры	*localname(...);*

(Идентификаторы: *Variable* = идентификатор переменной,
name = идентификатор блока (функции))

Инициализация параметров блока обязательна для блоков FC и SFC и необязательна для блоков SFB.

35.4 Стандартные функции CSL

35.4.1 Функции таймеров

Вызов	Тип данных	Комментарий
`Time_BCD :=`	WORD	
`Timer_function(`	(см. ниже)	
`T_NO = Timer_address,`	TIMER	адрес таймера
`S = Start_input,`	BOOL	вход запуска
`TV = Timer_duration,`	S5TIME	длительность
`R = Reset,`	BOOL	сброс
`Q = Timer_status,`	BOOL	состояние таймера
`BI = Binary_time);`	WORD	время в двоичном формате

В качестве функции `Timer_function` могут быть использованы следующие функции, обеспечивающие работу таймеров в следующих рабочих режимах:

S_PULSE	Режим управляемого импульса (Pulse time)
S_PEXT	Режим расширенного импульса (Extended pulse)
S_ODT	Режим с задержкой включения (ON delay)
S_ODTS	Режим с задержкой включения с памятью (Latching OFF delay)
S_OFFDT	Режим с задержкой выключения (Off delay)

35.4.2 Функции счетчиков

Вызов в режиме прямого счета	Тип данных	Комментарий
`BCD count value :=` `S_CU(`	WORD	
`C_NO := Count_address,`	COUNTER	адрес счетчика
`CU := Count_up,`	BOOL	режим прямого счета
`S := Set_input,`	BOOL	вход установки счетчика
`PV := Count_value,`	WORD	значение счетчика
`R := Reset,`	BOOL	сброс
`Q := Counter_status,`	BOOL	состояние счетчика
`CV := Bin_count_val);`	WORD	значение счетчика в двоичном формате
Вызов в режиме обратного счета		
`BCD count value :=` `S_CD(`	WORD	
`C_NO := Count_address,`	COUNTER	адрес счетчика
`CD := Count_down,`	BOOL	режим обратного счета
`S := Set_input,`	BOOL	вход установки счетчика
`PV := Count_value,`	WORD	значение счетчика
`R := Reset,`	BOOL	сброс
`Q := Counter_status,`	BOOL	состояние счетчика
`CV := Bin_count_val);`	WORD	значение счетчика в двоичном формате
Вызов в режиме прямого/обратного счета		
`BCD count value :=` `S_CU(`	WORD	
`C_NO := Count_address,`	COUNTER	адрес счетчика
`CU := Count_up,`	BOOL	режим прямого счета
`CD := Count_down,`	BOOL	режим обратного счета
`S := Set_input,`	BOOL	вход установки счетчика
`PV := Count_value,`	WORD	значение счетчика
`R := Reset,`	BOOL	сброс
`Q := Counter_status,`	BOOL	состояние счетчика
`CV := Bin_count_val);`	WORD	значение счетчика в двоичном формате

35.4.3 Функции преобразования

Неявные функции преобразования

Преобразование: тип исходный - тип целевой	Примечание
BOOL_TO_BYTE BOOL_TO_WORD BOOL_TO_DWORD BYTE_TO_WORD BYTE_TO_DWORD WORD_TO_DWORD	заполнение лидирующими нулями
INT_TO_DINT INT_TO_REAL DINT_TO_REAL	заполнение знаком
CHAR_TO_STRING	

Явные функции преобразования

Преобразование: тип исходный - тип целевой	Примечание
BYTE_TO_BOOL WORD_TO_BOOL DWORD_TO_BOOL WORD_TO_BYTE DWORD_TO_BYTE DWORD_TO_WORD	заполняются значением младшие бит, байт, слово
CHAR_TO_BYTE BYTE_TO_CHAR CHAR_TO_INT INT_TO_CHAR	без изменения назначения битов
STRING_TO_CHAR	

Явные функции преобразования (продолжение)

Преобразование: тип исходный - тип целевой	Примечание
WORD_TO_INT DWORD_TO_DINT INT_TO_WORD DINT_TO_DWORD REAL_TO_DWORD DWORD_TO_REAL	без изменения назначения битов (без конвертирования)
DINT_TO_INT REAL_TO_DINT REAL_TO_INT	с округлением до целого INT или DINT
TRUNC ROUND	преобразование данных формата REAL в DINT
DINT_TO_TIME DINT_TO_TOD DINT_TO_DATE DATE_TO_DINT TIME_TO_DINT TOD_TO_DINT	без изменения назначения битов
BLOCK_DB_TO_WORD WORD_TO_BLOCK_DB	без изменения назначения битов

35.4.4 Математические функции

Вызов	Тип данных

`Result :=`
`Math_function(`
`Input_value);`
	`REAL`
	(см. ниже)
	`ANY_NUM`

(Идентификаторы: `Result` = идентификатор переменной,
`Math_function` = математическая функция,
`Input_value` = входное значение)

В качестве функции `Math_function` могут быть использованы следующие математические функции:

SIN	синус
COS	косинус
TAN	тангенс
ASIN	арксинус
ACOS	арккосинус
ATAN	арктангенс
EXP	экспонента по основанию *e*
EXPD	экспонента по основанию 10
LN	натуральный логарифм
LOG	десятичный логарифм
SQR	нахождение квадрата числа
SQRT	извлечение квадратного корня из числа

ABS *функция нахождения абсолютного значения числа:*

Вызов функции ABS	Тип данных
`Result :=`	`ANY_NUM`
`ABS(Input_value);`	`ANY_NUM`

(Идентификаторы: `Result` = идентификатор переменной,
`Input_value` = входное значение)

35.4.5 Функции сдвига и циклического сдвига

Вызов	Тип данных
`Result :=`	`ANY_BIT`
`Shift_function(`	(см. ниже)
`IN := Input_value,`	`ANY_NUM`
`N  := Num_of_places);`	`INT`

(Идентификаторы: `Result` = идентификатор переменной,
`Shift_function` = функция сдвига,
`Input_value` = входное значение,
`Num_of_places` = число позиций сдвига)

В качестве функции `Shift_function` могут быть использованы следующие функции:

SHL	сдвиг влево
SHR	сдвиг вправо
ROL	циклический сдвиг влево
ROR	циклический сдвиг вправо

Предметный указатель

П

Р

P

R

S

T

Сокращения

AI	Analog Input	- Аналоговый вход
AO	Analog Output	- Аналоговый выход
AS	Automation System	- Автоматическая система
ASI	Actuator-Sensor Interface	- Интерфейс "привод-датчик"
BR	Binary Result	- Двоичный результат
CFC	Continuous Function Chard	- "Функциональный план"
CP	Communication Processor	- Коммуникационный процессор
CPU	Cential Processor Unit	- Центральный процессор
DB	Data Block	- Блок данных
DI	Digital Input	- Дискретный вход
DO	Digital Output	- Дискретный выход
DP	Distributed I/O	- Распределенные I/O
EPROM	Erasable Programmable ROM	- Стираемое программируемое ПЗУ
FB	Function Block	- Функциональный блок
FBD	Function Block Diagramm	- "Функциональная блок-схема"
FC	Function	- Функция
FEPROM	Flash Erasable Programmable ROM	- Флэш-ПЗУ
FM	Function Module	- Функциональный модуль
IM	Interface Module	- Интерфейсный модуль
LAD	Ladder Diagramm	- "Контактный план"
MCR	Master Control Relay	- Главное управляющее реле
MPI	Multi Point Interface	- "Многоточечный" интерфейс
OB	Organization Block	- Организационный блок
OP	Operator Panel	- Панель оператора
PG	Programming Device	- Программатор
PS	Power Supply	- Источник питания

RAM	Random Access Memory	- Память с произвольным доступом
RLO	Result of Logic Operation	- Результат логической операции
SCL	Structured Control Language	- Структурированный язык управления
SDB	System Data Block	- Системный блок данных
SFB	System Function Block	- Системный функциональный блок
SFC	System Function Call	- Системная функция
SM	Signal Module	- Сигнальный модуль
STL	Statement List	- "Список мнемоник"
SSL	System Status List	- Список состояний системы
UDT	User Data Type	- Пользовательский тип данных
VAT	Variable Table	- Таблица размещения переменных

Демонстрационные программы для STEP 7

Содержание

На прилагаемом компакт-диске Вы найдете:

- Демонстрационную версию ПО STEP 7 V 5.0
- Демонстрационную версию ПО S7 SCL V4.01 для программирования задач управления с помощью программируемой логики с использованием языка программирования высокого уровня
- Демонстрационную версию ПО S7 Giaph V4.0 для программирования систем непрерывного управления с использованием графических средств
- Демонстрационную версию ПО S7 HiGraph V4.01 для программирования задач управления с помощью программируемой логики с использованием графа состояний
- Демонстрационную версию ПО CFC V4.02 для программирования с использованием графических средств с "дружественным" по отношению к пользователю интерфейсом
- Интернет-проводник MS Internet Explorer с ограниченными функциональными возможностями
- Программу Adobe Acrobat Reader для чтения содержимого руководств по программному обеспечению

Инсталляция (установка) программного обеспечения

Для инсталляции (установки) программного обеспечения необходима следующая минимальная конфигурация программатора или персонального компьютера:

- Процессор
  80486 или Pentium
- ОЗУ (RAM)
  минимум 32 MB (рекомендуется 64 MB)

- Требуемый размер доступной памяти на жестком диске
  около 150 MB для демонстрационной версии ПО STEP 7 V 5.0 (на одну локализацию (один язык интерфейса) - стандартная инсталляция)
  около 10 MB для демонстрационной версии ПО S7 SCL V4.01
  около 20 MB для демонстрационной версии ПО S7 Giaph V4.0
  около 9 MB для демонстрационной версии ПО S7 HiGiaph V4.01
  около 31 MB для демонстрационной версии ПО CFC V4.02
- Операционная система
  Windows 95 / Windows 98 / Windows NT

Для инсталляции (установки) программного обеспечения вставьте компакт-диск в CD-привод, зарегистрируйтесь как пользователь и выбирайте требуемую программу.

Демонстрационная версия ПО STEP 7 может быть установлена на компьютер, в котором не установлена система STEP 7 или установлена система STEP 7 Mini ("full version" - "полная версия").

Перед тем, как установить опционные пакеты S7 SCL, S7 Giaph, S7 HiGraph и CFC, необходимо сначала установить демонстрационную версию или полную версию ПО STEP 7 V 5.0. Кроме того, указанные опционные пакеты должны быть инсталлированы на тот же диск, что и демонстрационная версия или полная версия ПО STEP 7 V 5.0.

Более подробную информацию по данным вопросам Вы можете найти в файле README.WRI на прилагаемом CD.

Ограничения, имеющиеся в демонстрационных версиях ПО, по сравнению с полными версиями программных продуктов.

- Проекты и данные проектов (например, блоки) не могут быть сохранены
- Исходные файлы не могут быть скомпилированы
- Только существующие программы могут загружаться в программируемый контроллер
- Что касается CFC: см. файл README

Демонстрационные версии ПО на прилагаемом CD не предназначены для использования в производственных целях для программирования, отладки программ и настройки программируемых контроллеров SIMATIC. Siemens не несет никакой ответственности за любые повреждения, полученные при использовании демонстрационного программного обеспечения, если претензии не обоснованы, например, при ущербе, нанесенном частной собственности или персоналу в результате умышленных действий или халатной небрежности.