

Ганс Бергер

Автоматизация с помощью Программ STEP7 LAD и FBD

Программируемые контроллеры
SIMATIC S7-300/400

Заказной номер::
6ES7810-4CA05-8AR0

Издание 2-е переработанное, 2001

Введение	
Программируемый контроллер SIMATIC S7-300/400	1
Программное обеспечение STEP 7	2
Программа SIMATIC S7	3
Операции бинарной логики	4
Функции для работы с памятью	5
Функции передачи	6
Таймеры	7
Счетчики	8
Функции сравнения	9
Арифметические функции	10
Математические функции	11
Функции преобразования	12
Функции сдвига	13
Побитовые логические операции	14
Биты состояния	15
Функции перехода	16
Главное реле управления	17
Функции для работы с блоками	18
Параметры блоков	19
Главная программа	20
Обработка прерываний	21
Особенности рестарта	22
Обработка ошибок	23
Дополнения к графическому программированию	24
Библиотеки блоков	25
Набор функций LAD	26
Набор функций FBD	27

Указания по безопасности

Это руководство содержит указания, которые вы должны соблюдать для обеспечения собственной безопасности, а также защиты продукта и подключенного оборудования. Эти указания выделены в руководстве предупреждающим треугольником и помечены следующим образом в соответствии с уровнем опасности:



Опасность

Указывает, что несоблюдение надлежащих предосторожностей приведет к смерти, тяжким телесным повреждениям или существенному повреждению имущества.



Предупреждение

Указывает, что несоблюдение надлежащих предосторожностей может привести к смерти, тяжким телесным повреждениям или существенному повреждению имущества.



Предостережение

Указывает, что несоблюдение надлежащих предосторожностей может привести к небольшим телесным повреждениям или порче имущества.

Замечание

Привлекает ваше внимание к особенно важной информации о продукте, обращении с продуктом или к определенной части документации.

Квалифицированный персонал

К установке и работе на данном оборудовании должен допускаться только квалифицированный персонал. К квалифицированному персоналу относятся лица, имеющие право пускать в эксплуатацию, заземлять и маркировать электрические цепи, оборудование и системы в соответствии с установленным порядком и стандартами.

Правильное использование

Примите во внимание следующее:



Предупреждение

Это устройство и его компоненты могут быть использованы только для приложений, описанных в каталоге или технических описаниях, и только в соединении с устройствами или компонентами других производителей, которые были одобрены или рекомендованы фирмой Siemens.

Этот продукт может правильно и безопасно функционировать только при правильной транспортировке, хранении, установке и инсталляции, а также эксплуатации и обслуживании в соответствии с рекомендациями.

Торговые марки

SIMATIC®, SIMATIC HMI® и SIMATIC NET® являются зарегистрированными торговыми марками SIEMENS AG.

Некоторые из других обозначений, использованных в этих документах, также являются зарегистрированными торговыми марками; права собственности могут быть нарушены, если эти обозначения используются третьей стороной для своих собственных целей.

Copyright © Siemens AG 2003 Все права сохраняются

Воспроизведение, передача или использование этого документа или его содержания не допускается без специального письменного разрешения. Нарушители будут нести ответственность за нанесенный ущерб. Все права, включая права, создаваемые патентным грантом или регистрацией сервисной модели или проекта, сохраняются.

Отказ от ответственности

Мы проверили содержание этого руководства на соответствие с описанной аппаратурой и программным обеспечением. Так как отклонения не могут быть полностью предотвращены, мы не гарантируем полного соответствия. Однако данные, приведенные в этом руководстве, регулярно пересматриваются и необходимые исправления вносятся в последующие издания. Приветствуются предложения по улучшению.

Siemens AG
Департамент техники автоматизации и приводов
Сфера деятельности: промышленные системы автоматизации
п/я 4848, D- 90327 Нюрнберг

©Siemens AG 2003
Технические данные могут изменяться.

Die Deutsche Bibliothek – CIP-Cataloguing-in-Publication-Data

Каталогизированная запись на исходную публикацию данного перевода доступна из Die Deutsche Bibliothek

Приведенные примеры программирования сосредоточены на описании функций LAD и FBD и предлагают пользователям SIMATIC S7 советы в области программирования решений специфических задач с использованием данного контроллера.

Примеры программирования не являются завершенными решениями и могут не исполняться в будущих релизах STEP7 или версиях S7-300/400. Должны быть приняты дополнительные шаги для выполнения соответствующих правил безопасности.

Автор и издатель уделили большое внимание всем текстам и иллюстрациям этой книги. Однако, ошибок полностью избежать не удастся. Издатель и автор исключают свою ответственность, вне зависимости от юридического основания, за любой ущерб от использования примеров программирования.

Автор и издатель с благодарностью услышат ваши отклики на содержание книги.

Publicis MCD Corporate Publishing

Postfach 3240

D-91052 Erlangen

Federal Republic of Germany

Fax: ++49 9131/72 78 38

E-mail: publishing-books@publicis-mcd.de

Предисловие

Новая система автоматизации SIMATIC объединяет все подсистемы, используемые в решении задач автоматизации, – от полевого уровня до управления процессом – в рамках однородной системной архитектуры в гомогенное целое. Это достигается с помощью интегрированных конфигурирования и программирования, управления данными и коммуникации с программируемыми контроллерами (SIMATIC S7), компьютерами автоматизации (SIMATIC M7) и системами управления (SIMATIC C7).

Всем нуждам автоматизации процесса и производства отвечают три серии программируемых контроллеров: S7-200 – компактные контроллеры («микро-PLC»), S7-300 и S7-400 – контроллеры с возможностью модульного расширения, предназначенные для применения как в системах с минимальными требованиями, так в высокопроизводительных системах.

STEP 7, дальнейшее развитие STEP 5, является программным обеспечением разработки программ для нового SIMATIC. С целью использования преимущества знакомого пользовательского интерфейса стандартных ПК (PC) (окна, операции с мышью) в качестве операционной системы выбрана Microsoft Windows 95/98 или Windows NT.

Для структурного (блочного) программирования STEP 7 предоставляет языки программирования, соответствующие DIN EN 6.1131-3. К ним относятся STL (statement list – список операторов или список мнемоник; ассемблероподобный язык), LAD (ladder logic или ladder diagram – контактный план; представление, схожее с диаграммами релейной логики; многоступенчатая схема), FBD (function block diagram – диаграмма функциональных блоков или функциональный план) и пакет SCL (паскалеподобный язык высокого уровня), который является дополнительным и может не входить в стандартную поставку.

Несколько дополнительных пакетов предоставляют следующие языки: S7-GRAPH (последовательное управление), S7-HiGraph (программирование с диаграммами «со-

стояние-переход») и CFC (соединение блоков; похож на диаграмму функциональных блоков).

Различные методы представления позволяют каждому пользователю выбрать подходящее описание функции управления. Такая широкая адаптируемость в представлении решаемой задачи управления значительно упрощает работу со STEP 7.

Эта книга содержит описание языков программирования LAD и FBD для S7-300/400. Первый раздел книги знакомит с системой автоматизации S7-300/400 и приводит основы использования STEP 7. Следующий раздел адресован начинающим пользователям или пользователям, переходящим с управления на основе реле-контакторов; здесь рассказывается об «Основных функциях» двоичного управления. Информация о цифровых функциях поясняет, как комбинируются цифровые значения; например, основные вычисления, сравнения, преобразование типов данных.

С помощью LAD или FBD вы можете управлять обработкой программы (программным потоком) и разрабатывать структурированные программы. Наряду с циклической обработкой основной программы вы можете включить программные секции, управляемые событиями, а также повлиять на поведение контроллера при запуске и при возникновении событий-ошибок/сбоев.

Книгу завершает общий обзор системных функций и набора функций для LAD и FBD.

Книга содержит описание программного обеспечения STEP 7 версии 5.1.

Обзор содержания книги

Обзор программируемого логического контроллера S7-300/400

Функции PLC, сопоставимые с контакторной системой управления

Манипулирование числами и цифровыми операндами

Введение

1 Программируемый контроллер SIMATIC S7-300/400

Структура программируемого контроллера (аппаратные компоненты S7-300/400)

Области памяти;

Распределенные входы / выходы (I/O) (PROFIBUS DP);

Коммуникация (подсети);

Адреса модулей;

Области адресов

2 Программирование в пакете STEP 7

Редактирование проектов;

Конфигурирование станций;

Конфигурирование сети;

Символьное редактирование;

Редактор программ LAD/FBD;

Онлайновый режим; тестирование программ LAD и FBD

3 Программа SIMATIC S7

Обработка программы;

Типы блоков;

Программирование кодовых блоков (Code Block) и блоков данных (Data Block);

Адресация переменных, представление констант, описание типов данных

Основные функции

4 Операции бинарной логики

Функции AND, OR и исключающее OR;

Функции вставки

5 Функции для работы с памятью

Присваивание, установка, сброс; коннектор; оценка фронта;

Пример системы управления ленточным конвейером

6 Функции передачи

Функции загрузки и перемещения;

Блочный элемент MOVE;

Системные функции для перемещения данных

7 Таймеры

Запуск SIMATIC-таймеров с пятью различными характеристиками, сброс и сканирование;

Функции IEC-таймера

8 Счетчики

SIMATIC-счетчики; прямой счет, обратный счет, установка, сброс и сканирование счетчиков;

Функции IEC-счетчика

Числовые функции

9 Функции сравнения

Сравнение в соответствии с типами данных INT, DINT и REAL

10 Арифметические функции

Четыре арифметических функции с числами INT, DINT и REAL

11 Математические функции

Тригонометрические функции;

Обратные тригонометрические функции;

Возведение в квадрат, извлечение квадратного корня, возведение в степень, логарифмы

12 Функции преобразования

Преобразование типов данных;

Формирование дополнения

13 Функции сдвига

Сдвиг и циклический сдвиг

14 Побитовые логические операции

Обработка логических операций AND, OR и исключающего OR над словами

Управление исполнением программы, функции блоков

Обработка пользовательской программы

Управление программным потоком	Обработка программы	Приложение
15 Биты состояния Двоичные флаги, цифровые флаги; Установка и вычисление битов состояния; Механизм EN/ENO	20 Главная программа Структура программы; Управление циклом сканирования (время отклика, стартовая информация, фоновое сканирование); Программные функции; Коммуникация через распределенные входы/выходы и глобальные данные; SFC и SFB-коммуникация	24 Дополнения к графическому программированию Защита блока KNOW_HOW_PROTECT; Косвенная адресация, указатели: общие замечания; Краткое описание «Примера фрейма сообщения»
16 Функции перехода Безусловный переход; Переход, если RLO = «1»; Переход, если RLO = «0»;	21 Обработка прерываний Аппаратные прерывания; Циклические прерывания; Прерывания по времени суток; Прерывания задержки времени; Мультипроцессорные прерывания; Обработка событий по прерываниям	25 Библиотеки блоков Организационные блоки; Системные функциональные блоки; Функциональные блоки IEC; Блоки преобразования S5 - S7; Блоки преобразования TI - S7; Блоки PID-управления; Коммуникационные блоки
17 Главное реле управления MCR-зависимость, MCR-область, MCR-зона	22 Особенности рестарта Холодный рестарт, теплый рестарт, полный рестарт; STOP, HOLD, сброс памяти; Параметризация модулей	26 Набор функций LAD Основные функции; Числовые функции; Управление программным потоком
18 Функции для работы с блоками Вызов блока, Конец блока Временные и статические локальные данные, локальные экземпляры; Адресация операндов данных, открытие блока данных	23 Обработка ошибок Синхронные ошибки Асинхронные ошибки; Системная диагностика	27 Набор функций FBD Основные функции; Числовые функции; Управление программным потоком

Обзор содержимого дискеты

Эта книга содержит множество рисунков, демонстрирующих использование языков программирования LAD и FBD. Все программы, представленные в книге, вы можете найти на прилагаемой дискете в двух библиотеках: LAD_Book и FBD_Book. После разархивирования с помощью функции Retrieve (Восстановление) эти библиотеки займут приблизительно 2 Мб (в зависимости от используемой файловой системы PC/PK).

Библиотеки LAD_Book и FBD_Book содержат восемь программ, которые по существу являются иллюстрациями графического представления. Программирование функций, функциональные блоки и локальные экземпляры (пример конвейера), а также обработка данных (пример фрейма сообщения) показаны в двух исчерпывающих примерах. Все примеры содержат символы и комментарии.

Библиотеки поставляются в архивированном виде. Перед началом работы с ними вы должны разархивировать библиотеки. В SIMATIC-менеджере (SIMATIC Manager) выберите пункт меню File → Dearchive (Файл → Разархивировать) и следуйте инструкциям (смотрите также файл README.TXT на диске).

Для испытания программ настройте проект в соответствии с вашим аппаратным обеспечением, затем скопируйте программу, включая таблицу символов, из библиотеки в проект. Теперь вы можете вызвать примеры программ, адаптировать их к вашим целям и тестировать их в онлайн-режиме.

Если у вас нет полной версии STEP 7 или STEP 7 Mini, вы также можете просмотреть примеры с помощью демо-версии STEP 7 на прилагаемом компакт-диске.

Библиотека LAD_Book

Типы данных

Примеры определения и приложения

FB 101 Простые типы данных

FB 102 Сложные типы данных

FB 103 Типы параметров

Основные функции

Примеры представлений LAD

FB 104 Глава 4: Последовательные и параллельные схемы

FB 105 Глава 5: Функции для работы с памятью

FB 106 Глава 6: Функции перемещения

FB 107 Глава 7: Функции таймера

FB 108 Глава 8: Функции счетчика

Цифровые функции

Примеры представлений LAD

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: Побитовые логические операции

Управление программным потоком

Примеры представлений LAD

FB 115 Глава 15: Биты состояния

FB 116 Глава 16: Функции перехода

FB 117 Глава 17: Главное реле управления

FB 118 Глава 18: Функции для работы с блоками

FB 119 Глава 19: Параметры блоков

Обработка программы

Примеры SFC-вызовов

FB 120 Глава 20: Главная программа

FB 121 Глава 21: Обработка прерываний

FB 122 Глава 22: Характеристики запуска

FB 123 Глава 23: Обработка ошибок

Пример конвейера

Примеры базовых функций и локальных экземпляров

FC 11 Управление ленточным транспортером

FC 12 Управление счетчиком

FB 20 Питание (устройство подачи)

FB 21 Ленточный транспортер конвейера

FB 22 Счетчик деталей

Пример фрейма сообщения

Примеры обработки данных

UDT 51 Структура данных для заголовка фрейма

UDT 52 Структура данных для сообщения

FB 51 Генерирование фрейма сообщения

FB 52 Хранение фрейма сообщения

FC 51 Проверка времени суток

FC 52 Копирование области данных с использованием косвенной адресации

Общие примеры

FC 41	Монитор диапазона
FC 42	Определение граничного значения
FC 43	Вычисление сложного процента
FC 44	32-битная (двойное слово) оценка фронта

Библиотека FBD_Book

Типы данных

Примеры определения и приложения

FB 101	Простые типы данных
FB 102	Сложные типы данных
FB 103	Типы параметров

Основные функции

Примеры представлений FBD

FB 104	Глава 4: Последовательные и параллельные схемы
FB 105	Глава 5: Функции для работы с памятью
FB 106	Глава 6: Функции перемещения
FB 107	Глава 7: Функции таймера
FB 108	Глава 8: Функции счетчика

Цифровые функции

Примеры представлений FBD

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: Побитовые логические операции

Управление программным потоком

Примеры представлений FBD

FB 115	Глава 15: Биты состояния
FB 116	Глава 16: Функции перехода
FB 117	Глава 17: Главное реле управления
FB 118	Глава 18: Функции для работы с блоками
FB 119	Глава 19: Параметры блоков

Обработка программы

Примеры SFC-вызовов

FB 120	Глава 20: Главная программа
FB 121	Глава 21: Обработка прерываний
FB 122	Глава 22: Характеристики запуска
FB 123	Глава 23: Обработка ошибок

Пример конвейера

Примеры базовых функций и локальных экземпляров

FC 11	Управление ленточным транспортером
FC 12	Управление счетчиком
FB 20	Питание (устройство подачи)
FB 21	Ленточный транспортер конвейера
FB 22	Счетчик деталей

Пример фрейма сообщения

Примеры обработки данных

UDT 51 Структура данных для заголовка фрейма

UDT 52 Структура данных для сообщения

FB 51 Генерирование фрейма сообщения

FB 52 Хранение фрейма сообщения

FC 51 Проверка времени суток

FC 52 Копирование области данных с использованием косвенной адресации

Общие примеры

FC 41 Монитор диапазона

FC 42 Определение граничного значения

FC 43 Вычисление сложного процента

FC 44 32-битная (двойное слово) оценка фронта

Автоматизация с использованием STEP 7

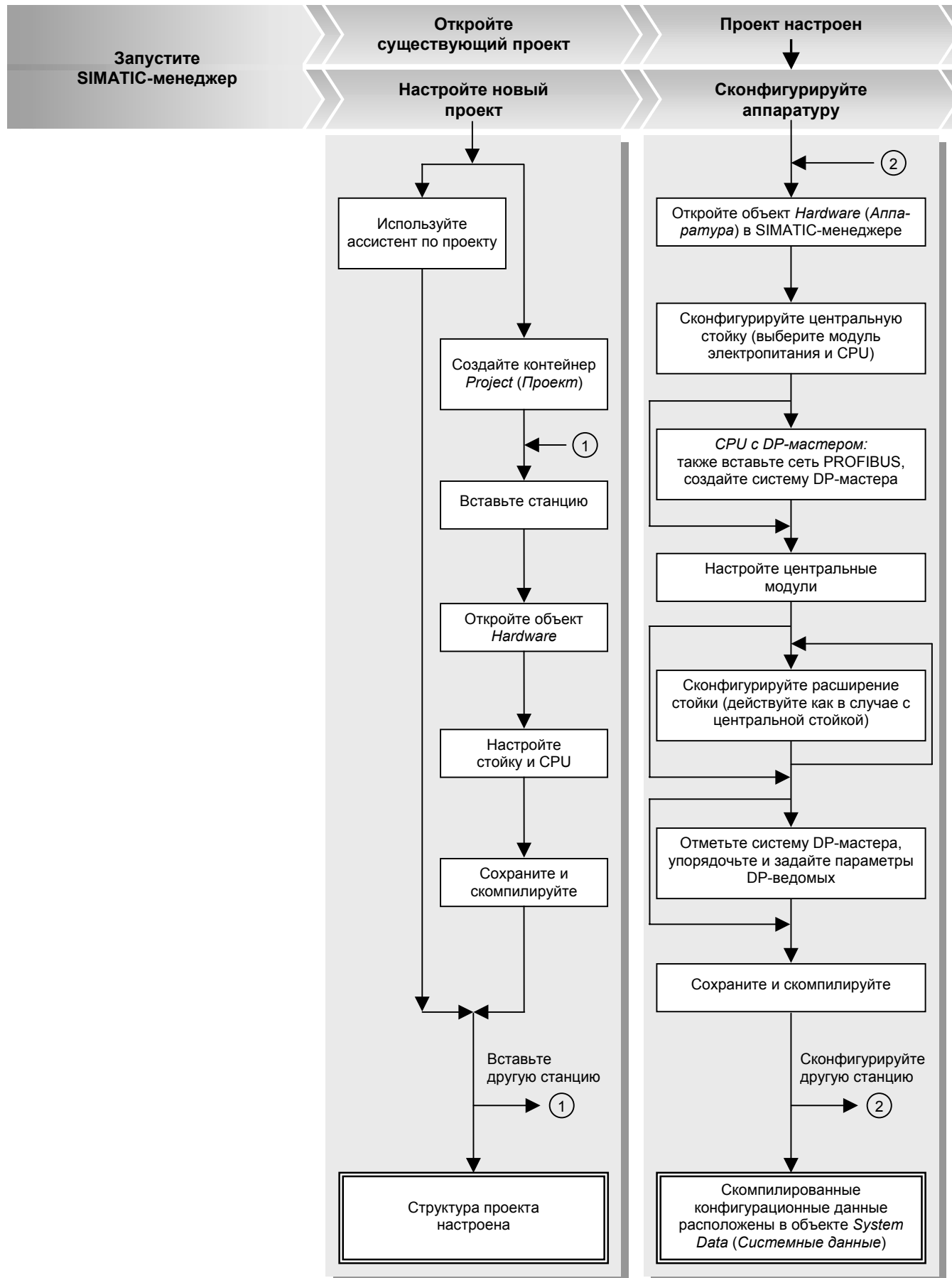
На следующих двух страницах показана основная процедура использования программного обеспечения STEP 7.

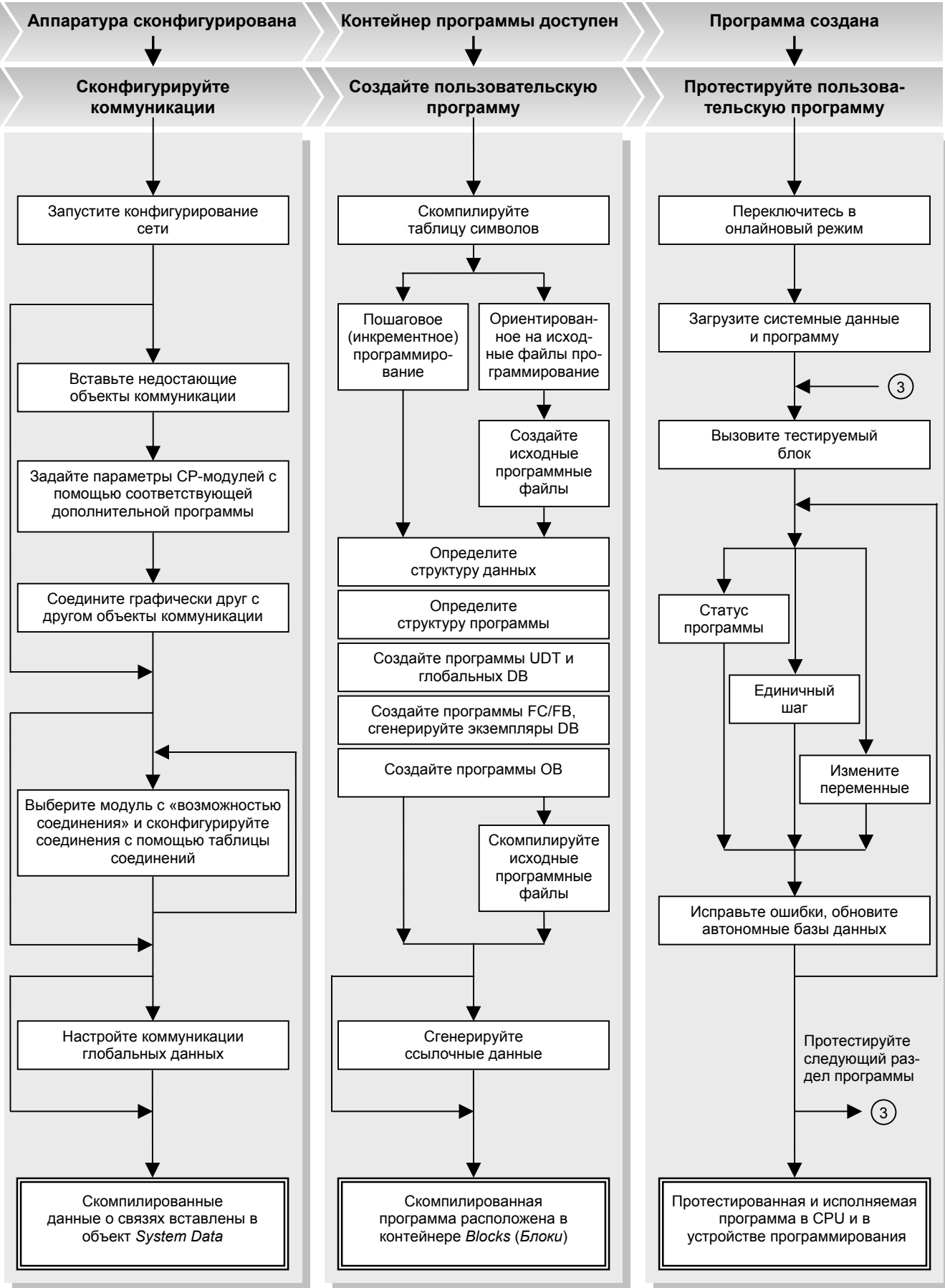
Запустите программу SIMATIC-менеджер (SIMATIC Manager) и настройте новый проект (начните разработку нового проекта) или откройте существующий. В проекте все данные для задачи автоматизации хранятся в форме объектов. При настройке проекта вы создаете контейнеры для аккумулируемых данных путем установки требуемых станций, содержащих, по крайней мере, CPU; затем также создаются контейнеры для пользовательских программ. Также вы можете создать программный контейнер непосредственно в проекте.

На следующих этапах вы конфигурируете аппаратные средства и, если требуется, коммуникационные соединения. Затем вы создаете и тестируете программу.

Порядок создания данных автоматизации не жесткий. Применяются лишь следующие установки: если вы хотите обработать объекты (данные), они должны существовать; если вы хотите вставить объекты, соответствующие контейнеры должны быть доступны.

Можно прервать процесс обработки в проекте в любое время и при следующем запуске SIMATIC-менеджера снова продолжить с любого места.





Введение

Эта часть книги содержит обзор SIMATIC S7-300/400.

Программируемый контроллер S7-300/400 имеет модульное построение. Модули, из которых он состоит, могут быть центральными (расположенными рядом с CPU) или распределенными без обязательных специальных установок или назначений параметров. В системах SIMATIC S7 распределенный вход/выход (I/O) является интегрированной частью системы. CPU с его различными областями памяти формирует аппаратную основу для обработки пользовательских программ. Загрузочная память (load memory) содержит программу пользователя в полном объеме: части программы в соответствии с ее исполнением в любой данный момент времени находятся в рабочей памяти (work memory), малое время доступа к которой необходимо для быстрой обработки программы.

STEP 7 – это программное обеспечение для S7-300/400, основным инструментом для решения задач автоматизации является SIMATIC-менеджер (SIMATIC Manager). SIMATIC-менеджер – это приложение Windows 95/98/NT и содержит все функции, требуемые для установки проекта. При необходимости SIMATIC-менеджер запускает дополнительный инструментарий, например, для конфигурирования станций, инициализации модулей, написания и тестирования программ.

Для формулирования решения задачи автоматизации вы будете использовать языки программирования STEP 7. Программа SIMATIC S7 структурирована, то есть она состоит из блоков с определенными функциями, образованных из сетей (networks) или цепей (rungs). Различные приоритетные классы предоставляют механизм ранжированных прерываний пользовательской программы, исполняемой в текущий момент. STEP 7 работает с переменными различных типов данных, начиная с бинарных переменных (тип данных BOOL – логический), далее с числовыми переменными (типы данных INT или REAL – целый или вещественный – для вычислительных задач), заканчивая сложными или комплексными типами данных, такими как массивы или структуры (формирование одной переменной из комбинации переменных различных типов).

Первая глава содержит обзор аппаратного обеспечения программируемого контроллера S7-300/400, вторая глава обзорно рассказывает о программном обеспечении STEP 7. Основой для описания служит функциональный набор STEP 7 версии 5.1.

Глава 3 «Программа SIMATIC S7» является введением в самые важные элементы S7-программы и демонстрирует программирование отдельных блоков на языках программирования LAD и FBD. В следующих главах книги приводится описание функций и операций LAD и FBD. Пояснения сопровождаются краткими примерами.

1 Программируемый контроллер SIMATIC S7-300/400

Структура программируемого контроллера; распределенные входы/выходы; коммуникации; адреса модулей; области операндов

2 Программное обеспечение STEP 7

SIMATIC-менеджер; обработка проекта; конфигурирование станции; конфигурирование сети; написание программ (таблица символов, редактор программ); переключение в онлайн-режим; тестирование программ

3 Программа SIMATIC S7

Обработка программы с приоритетными классами; программные блоки; адресация переменных; программирование блоков с помощью LAD и FBD; переменные и константы; типы данных (обзор)

Глава 1

Программируемый контроллер SIMATIC S7-300/400

Содержание главы 1

1	<u>Программируемый контроллер SIMATIC S7-300/400</u>	4
1.1	<u>Структура программируемого контроллера</u>	4
1.1.1	<u>Компоненты</u>	4
1.1.2	<u>Станция S7-300</u>	6
1.1.3	<u>Станция S7-400</u>	7
1.1.4	<u>Области памяти CPU</u>	9
1.1.5	<u>Карта памяти</u>	10
1.1.6	<u>Системная память</u>	11
1.2	<u>Распределенные входы/выходы</u>	13
1.2.1	<u>Система DP-мастера</u>	13
1.2.2	<u>DP-мастер</u>	14
1.2.3	<u>DP-ведомые</u>	14
1.2.4	<u>Подключение к PROFIBUS-PA</u>	16
1.2.5	<u>Подключение к AS-интерфейсу</u>	17
1.2.6	<u>Подключение к последовательному интерфейсу</u>	18
1.3	<u>Коммуникации</u>	20
1.3.1	<u>Вводная часть</u>	20
1.3.2	<u>Подсети</u>	23
1.3.3	<u>Коммуникационные сервисы</u>	26
1.3.4	<u>Соединения</u>	27
1.4	<u>Адреса модулей</u>	29
1.4.1	<u>Путь сигнала</u>	29
1.4.2	<u>Адрес слота</u>	29
1.4.3	<u>Стартовый адрес модуля</u>	29
1.4.4	<u>Диагностический адрес</u>	31
1.4.5	<u>Адреса для узлов шины</u>	32
1.5	<u>Области адресов</u>	33
1.5.1	<u>Область пользовательских данных</u>	33
1.5.2	<u>Образ процесса</u>	34
1.5.3	<u>Память меркеров</u>	36

1 Программируемый контроллер SIMATIC S7-300/400

1.1 Структура программируемого контроллера

1.1.1 Компоненты

SIMATIC S7-300/400 является модульным программируемым контроллером, включающим следующие компоненты:

- Стойки (Racks)
Служат для размещения модулей и соединения их между собой;
- Электропитание (Power Supply, PS)
Обеспечивает подачу электроэнергии к внутренним устройствам;
- Центральное процессорное устройство (Central Processing Unit, CPU)
Хранит и обрабатывает программу пользователя;
- Интерфейсные модули (Interface Modules, IM)
Соединяют стойку с другой стойкой;
- Сигнальные модули (Signal Modules, SM)
Адаптируют системные сигналы к внутреннему уровню сигнала или управляют приводами посредством цифровых или аналоговых сигналов;
- Функциональные модули (Function Modules, FM)
Выполняют сложную или критичную по времени обработку независимо от CPU;
- Коммуникационные процессоры (Communications Processors, CP)
Устанавливают соединения со вспомогательными сетями (подсетями);
- Подсети (Subnets)
Соединяют программируемые контроллеры друг с другом и с другими устройствами.

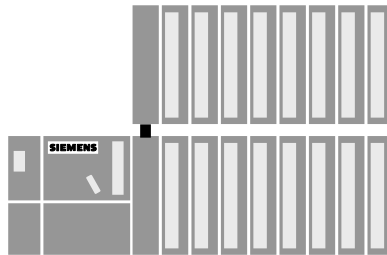
Программируемый контроллер (или станция) может состоять из нескольких стоек, которые связаны между собой шинными кабелями. Модуль электропитания, CPU и модули входов/выходов (SM, FM и CP) (или I/O-модули) включены в центральную стойку. Если для модулей входов/выходов в центральной стойке недостаточно места или если вы хотите некоторые или все I/O-модули вывести из центральной стойки, можно использовать стойки расширения, которые соединены с центральной стойкой посредством интерфейсных модулей (рисунок 1.1). Также есть возможность подключить распределенные входы/выходы к станции (обратитесь к параграфу 1.2 «Распределенные входы/выходы»).

Модульная конфигурация станции S7-300

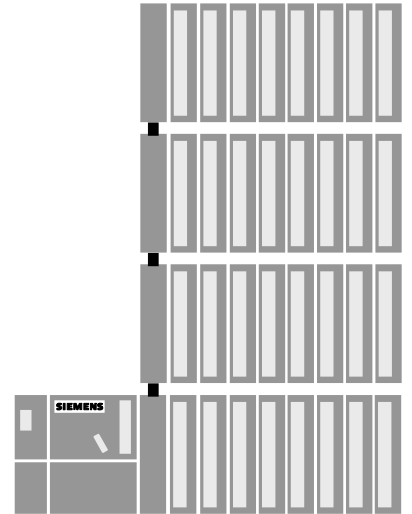
Одноуровневая конфигурация



Двухуровневая конфигурация с IM 365



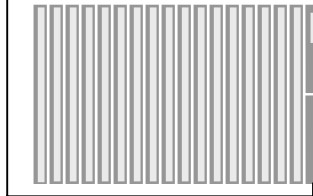
Четырехуровневая конфигурация с IM 365 и IM 361



В центральной контроллерной стойке:
 IM 460-1
 IM 460-0
 IM 460-3
 IM 463-2

Модульная конфигурация станции S7-400

Локальное расположение до 1,5 м с передачей напряжения питания 5В (IM 461-1)



Локальное расположение до 3 м без передачи напряжения 5В (IM 461-0)



Удаленное расположение до 100 м без передачи напряжения питания 5В (IM 461-3)



Удаленное расположение до 600 м для устройств расширения S5 (IM 314)



Рисунок 1.1 Конфигурация аппаратного обеспечения для S7-300/400

Модули соединены в стойках посредством двух шин: шины входов/выходов (I/O-шина, периферийная шина или P-шина) и коммуникационной шины (или K-шина). I/O-шина (P-шина) разработана для высокоскоростного обмена входными и выходными сигналами, коммуникационная шина (K-шина) – для обмена большими объемами данных. Коммуникационная шина связывает CPU и интерфейс устройства программирования (MPI) с функциональными модулями и коммуникационными процессорами.

1.1.2 Станция S7-300

Централизованная конфигурация

В центральную стойку контроллера S7-300 может быть включено до 8 I/O-модулей. Если эта одноуровневая конфигурация оказывается недостаточной, у вас имеются два варианта для контроллеров, оборудованных CPU 314 или более совершенными процессорами:

- либо выбрать двухуровневую конфигурацию (с IM 365, между стойками до 1 м)
- либо выбрать конфигурацию с количеством уровней до четырех (с IM 360 и IM 361, расстояние между стойками до 10 м).

Вы можете задействовать в стойке максимум 8 модулей. Количество модулей может быть ограничено максимально допустимым током для стойки, который составляет 1,2 А (0,8 А для CPU 312 IFM).

Модули связаны между собой шиной задней панели, сочетающей в себе функции P- и K-шины.

Локальный сегмент шины

Дополнительные возможности при конфигурировании предоставляет использование прикладного модуля FM 356 из семейства компьютеров автоматизации M7-300. FM 356 способен «разделить» шину задней панели модуля и может сам управлять остальными модулями в отделенном «локальном сегменте шины». В этом случае также действуют упомянутые выше ограничения по количеству модулей и потреблению энергии.

SIMATIC и внешние условия

Существуют модули SIMATIC S7-300, применимые в жестких климатических условиях. У них расширенный рабочий диапазон температур от –25 до +60°C, повышенная устойчивость к вибрациям и ударным воздействиям в соответствии с IEC 68 часть 2-6, и они выполняют требования по влажности, конденсации и охлаждению в соответствии с IEC 721-3-3 класс 3 K5, а также требования по вращательному воз-

действию в соответствии с EN 50155 (скоро вступит в действие). Остальные технические характеристики идентичны свойствам стандартных модулей.

1.1.3 Станция S7-400

Централизованная конфигурация

В семействе S7-400 применяются центральные стойки с 18 или 9 слотами (UR1 или UR2); модуль электропитания и CPU также занимают слоты, иногда даже два или более на каждый модуль. Интерфейсные модули IM 460-1 и IM 461-1 предоставляют возможность иметь одну стойку расширения на каждый интерфейс с удалением до 1,5 м от центральной стойки, включая электропитание 5 В. Кроме того, можно задействовать четыре стойки расширения с удалением до 3 м, используя интерфейсные модули IM 360-0 и IM 361-0. И, наконец, могут быть использованы интерфейсные модули IM 360-3 и IM 361-3 для функционирования четырех стоек расширения на расстоянии до 100 м.

К центральной стойке можно подключить максимум 21 расширяющую стойку. Для различия между стойками на кодовом переключателе принимающего IM устанавливаются их номера.

Шина задней панели состоит из параллельной Р-шины и последовательной К-шины. Стойки расширения ER1 и ER2 с 18 или 9 слотами разработаны для «простых» сигнальных модулей, которые не генерируют аппаратных прерываний, не потребляют напряжение 24 В через Р-шину, не требуют резервного питания и не имеют соединения с К-шиной. К-шина находится в стойках UR1, UR2 и CR2 и при использовании их в качестве центральных, и как расширяющих с номерами 1 – 6.

Сегментированная стойка

Специальным элементом является сегментированная стойка CR2. В стойке могут быть размещены два CPU с разделяемым (одним) электропитанием при их независимом функционировании. Оба CPU могут обмениваться данными по К-шине, но иметь полностью независимые Р-шины для их собственных сигнальных модулей.

Мультипроцессорный режим

В S7-400, работающем в мультипроцессорном режиме, могут функционировать до четырех специально разработанных CPU, размещенных в соответствующих стойках. В такой станции каждый модуль со своими адресами и прерываниями назначен только одному CPU. За более подробной информацией обратитесь к параграфам 20.3.6 «Мультипроцессорный режим» и 21.6 «Мультипроцессорное прерывание».

Подключение модулей SIMATIC S5

Интерфейсный модуль IM 463-2 позволяет подключить устройства расширения S5 (EG 183U, EG 185U, EG 186U, а также ER 701-2 и ER 701-3) к S7-400, а также предоставляет возможность централизованного наращивания устройств расширения. IM 314 и устройство расширения S5 управляют связью. Вы можете оперировать всеми аналоговыми и цифровыми модулями, доступными в этих устройствах расширения. S7-400 может содержать четыре интерфейсных модуля IM 463-2; к каждому интерфейсу IM 463-2 могут быть подключены в распределенной конфигурации четыре устройства расширения S5.

Программное резервирование

Используя стандартные компоненты SIMATIC S7-300/400, вы можете создать с помощью программного обеспечения резервированную систему с управляющей процессом главной станцией (master station) и резервной станцией (standby station), принимающей управление в случае сбоя главной станции.

Отказоустойчивость, достигаемая при помощи программного резервирования, применима для медленных процессов, потому что передача в резервную станцию может занять несколько секунд в зависимости от конфигурации программируемых контроллеров. В этот промежуток времени сигналы процесса «заморожены». Затем резервная станция продолжает работу с последними корректными данными главной станции.

Резервирование модулей входов/выходов реализуется с помощью распределенных входов/выходов (ET 200M с интерфейсным модулем IM 153-3 для резервированной PROFIBUS DP). Для конфигурирования используется дополнительная (может не включаться в пакет поставки) программа «Software Redundancy» («Программное резервирование»).

Отказоустойчивый SIMATIC S7-400H

SIMATIC S7-400H является устойчивым к сбоям программируемым контроллером с резервированной конфигурацией, включающим в себя две центральные стойки, в каждой – H-CPU и модуль синхронизации для сравнения данных с передачей через оптико-волоконный кабель. Оба контроллера работают в режиме «горячего резервирования»; в случае сбоя работающий корректно контроллер продолжает функционировать в одиночку в режиме автоматической защищенной передачи.

Входы/выходы могут иметь нормальный доступ (одноканальная, односторонняя конфигурация) или расширенный (одноканальная переключенная конфигурация с ET 200M). Коммуникация осуществляется через простую или резервированную шину.

Пользовательская программа – та же самая, что и для нерезервированного контроллера; функция резервирования управляется исключительно аппаратными средствами

и для пользователя невидна. Для конфигурирования требуется дополнительное программное обеспечение «S7-400H».

1.1.4 Области памяти CPU

Пользовательская память

На рисунке 1.2 показаны области памяти CPU, важные для вашей программы. Сама пользовательская программа находится в двух областях, которые называются загрузочная память (load memory) и рабочая память (work memory).

Загрузочная память может быть интегрированной в CPU или подключаемой (plugin) картой памяти. Вся программа пользователя, включая конфигурационные данные, располагается в загрузочной памяти.

Рабочая память представляет собой высокоскоростную RAM (память со случайным доступом), полностью встроенную в CPU. Рабочая память содержит связанные части пользовательской программы; в основном это программный код и пользовательские данные. «Связанный» - это характеристика существующих объектов и не означает, что определенный блок кода будет обязательно вызван и исполнен.

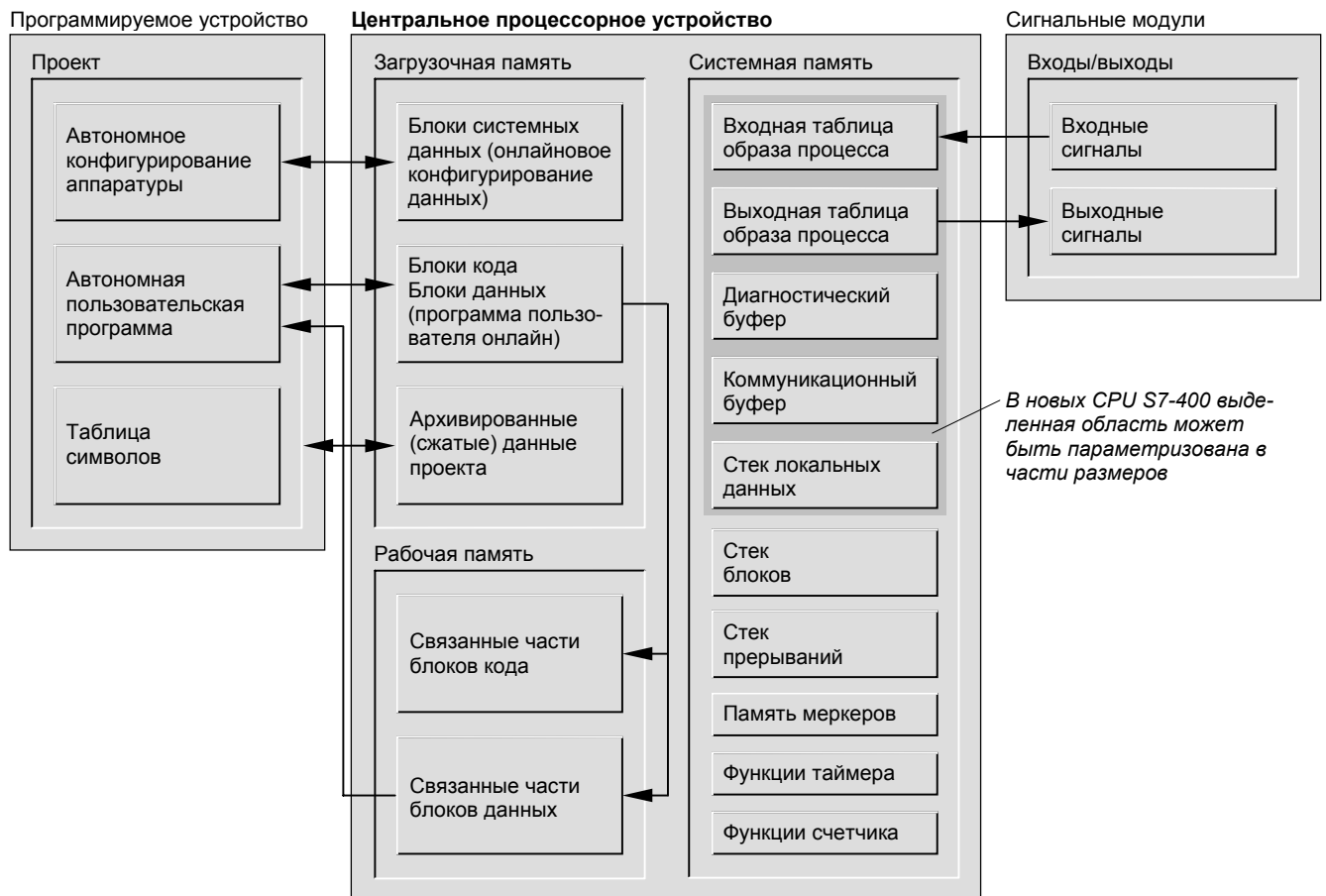


Рисунок 1.2 Области памяти CPU

Программирующее устройство (программатор) переносит всю программу, включая данные конфигурации, в загрузочную память. Затем операционная система CPU копирует «связанный» программный код и пользовательские данные в рабочую память. При обратной загрузке программы в программирующее устройство выбранные из загрузочной памяти блоки дополняются текущими значениями адресов данных из рабочей памяти (подробнее об этом рассказывается в параграфах 2.6.4 «Загрузка пользовательской программы в CPU» и 2.6.5 «Обработка блоков»).

Если загрузочная память состоит из RAM, для хранения программы пользователя требуется резервная батарея. Там, где загрузочная память реализована как интегрированная EEPROM или подключаемая карта флэш-памяти EPROM, CPU может работать без резервной батареи.

В состав загрузочной памяти CPU 3ххIFM входят RAM- и EEPROM-компоненты. Программа перемещается и тестируется в памяти RAM, а затем посредством команды меню протестированная программа может храниться в интегральной (встроенной) памяти EEPROM, защищенной от сбоев питания.

CPU S7-300 (за исключением CPU 318) снабжены встроенной загрузочной памятью RAM, способной хранить всю программу. Карту флэш-памяти EPROM вы можете использовать в качестве носителя данных или защищенной от сбоев электропитания среды хранения для программы пользователя.

В S7-300 текущие значения частей пользовательской памяти (блоки данных – data blocks) и системной памяти (память меркеров, таймеры и счетчики) могут храниться на энергонезависимом элементе памяти. Таким образом, вы можете сохранить и обезопасить ваши данные в случае сбоя питания без резервного источника питания.

Встроенная загрузочная память RAM CPU S7-400 разработана для небольших программ или для модификации отдельных блоков. Если вся программа больше, чем встроенная загрузочная память, для тестирования вам потребуется карта памяти RAM. Можно воспользоваться картой флэш-памяти EPROM в качестве среды данных или защищенной от сбоев питания среды памяти.

На новых CPU S7-400 рабочая память может быть расширена подключаемыми модулями.

STEP 7 с версии 5.1 (V5.1) с применением соответствующим образом разработанных CPU S7-400 позволяет хранить все данные проекта в загрузочной памяти как сжатый архивный файл (обратитесь к параграфу 2.2.2 «Управление, перегруппировка и архивирование»).

1.1.5 Карта памяти

Доступными для применения являются два типа карт памяти: RAM-карты и EPROM-карты.

Если вы хотите только расширить загрузочную память, используйте карту RAM (например, в CPU S7-400). RAM-карта позволяет модифицировать всю пользователь-

скую программу в онлайнном (интерактивном) режиме. Карты памяти RAM при отключении теряют свое содержимое.

Если вы хотите защитить вашу программу, включая конфигурационные данные и параметры модулей, от сбоев электропитания, используйте флэшкарты EPROM. В этом случае загрузите автономно (в режиме офлайн) всю программу на флэшкарту EPROM, подключенную к устройству программирования. С помощью соответствующих CPU также можно загрузить программу в онлайнном режиме, используя подключенную к CPU карту памяти.

1.1.6 Системная память

Системная память (system memory) содержит адреса (переменные), к которым вы обращаетесь в вашей программе. Адреса объединены в области (области адресов), которые содержат определяемое центральным процессорным устройством количество адресов. Адреса могут быть, к примеру, входами, используемыми для сканирования сигнального состояния переключателей мгновенного контакта (кнопок) и переключателей ограничения (конечных выключателей), и выходами, которые можно задействовать для управления контакторами и лампами. Системная память CPU содержит следующие области адресов:

- Входы (Inputs, I)
Входы являются образом («образом процесса») модулей цифрового входа.
- Выходы (Outputs, Q)
Выходы являются образом («образом процесса») модулей цифрового выхода.
- Память меркеров (Bit memory, M)
Хранит информацию, доступную всей программе.
- Таймеры (Timers, T)
Таймеры – это ячейки памяти, используемые для реализации интервалов ожидания и мониторинга.
- Счетчики (Counters, C)
Счетчики – это организуемые на программном уровне ячейки памяти, используемые для ведения счета по возрастанию и убыванию.
- Временные локальные данные (Temporary local data, L)
Ячейки памяти, используемые в качестве динамических промежуточных буферов во время обработки блоков. Временные локальные данные расположены в L-стеке, который динамически используется CPU во время выполнения программы.

Буквы, указанные в скобках после названия, представляют собой аббревиатуры, используемые для различных адресов при написании программы. Вы также можете присвоить символ каждой переменной и затем использовать его вместо идентификатора адреса.

Системная память, кроме того, содержит буферы для коммуникационных служб и системных сообщений (диагностический буфер). В новых CPU S7-400 размеры этих буферов данных, так же как размеры образа процесса и L-стека, могут быть параметризованы.

1.2 Распределенные входы/выходы

PROFIBUS-DP предоставляет стандартизованный интерфейс для передачи преимущественно двоичных данных процесса между «интерфейсным модулем» (центрального) программируемого контроллера и полевыми устройствами. Этот «интерфейсный модуль» называется DP-мастером (DP master), а полевые устройства DP-ведомыми (DP slaves). Распределенный (удаленный) вход/выход обращается к модулям, соединенным посредством PROFIBUS-DP с главным модулем PROFIBUS (мастер-модулем PROFIBUS). PROFIBUS-DP отвечает требованиям EN 50170 и является независимым стандартом для подключения ведомых стандарта DP.

За более подробной информацией о PROFIBUS-DP можно обратиться к параграфу 1.3.2 «Подсети».

DP-мастером и всеми его ведомыми управляет система DP-мастера. В одном сегменте может быть до 32 станций, во всей сети – до 127. Количество управляемых DP-мастером ведомых определяется им самим. Кроме того, вы можете подключить программирующие устройства к сети PROFIBUS-DP так же как, к примеру, устройства для интерфейса человек-машина, устройства ET 200 или DP-ведомые SIMATIC S5.

1.2.1 Система DP-мастера

Одномастерная система

PROFIBUS-DP обычно работает как «одномастерная система», то есть один DP-мастер управляет несколькими DP-ведомыми. DP-мастер является единственным мастером на шине, за исключением временно активируемых программирующих устройств (устройств диагностики и обслуживания). DP-мастер и назначенные ему DP-ведомые формируют систему DP-мастера (рисунок 1.3).

Многомастерная система

Вы также можете установить несколько систем DP-мастера (несколько DP-мастеров) в одной подсети PROFIBUS (многомастерная система). Однако, в отдельных случаях это увеличивает время отклика, потому что после инициализации DP-мастером «своих» DP-ведомых права доступа передаются следующему DP-мастеру, который, в свою очередь, инициализирует «свои» DP-ведомые, и т.д.

Несколько систем DP-мастера в станции

Время отклика может уменьшиться, если система DP-мастера содержит небольшое количество DP-ведомых. В этом случае возможно управление указанными DP-мастерами в одной станции S7, вы можете распределить DP-ведомые станции по нескольким системам DP-мастера. В мультипроцессорном режиме каждый CPU имеет свою собственную систему DP-мастера.

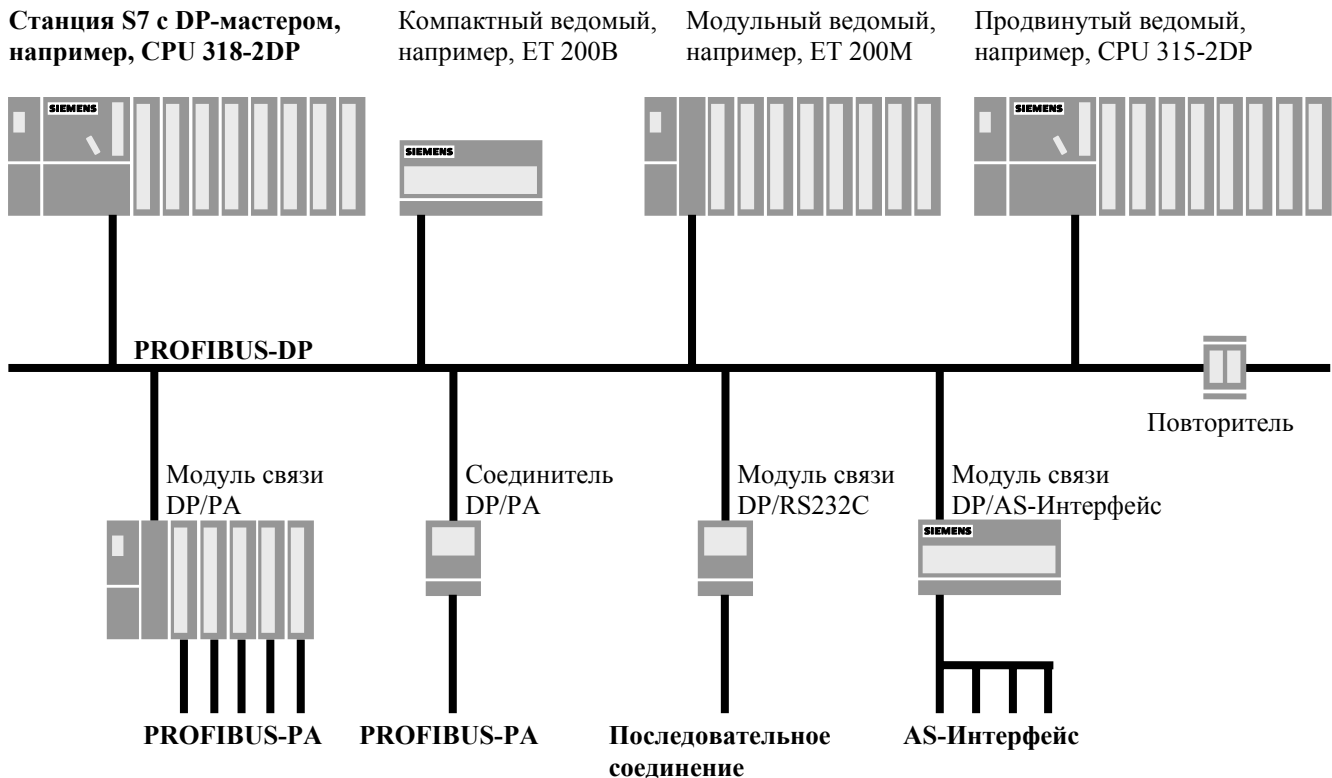


Рисунок 1.3 Компоненты системы мастера PROFIBUS-DP

1.2.2 DP-мастер

DP-мастер является активным узлом в сети PROFIBUS. Он обменивается циклическими данными со «своими» DP-ведомыми. DP-мастером могут быть

- CPU со встроенным интерфейсом DP-мастера или подключенным интерфейсным подмодулем (например, CPU 315-2DP, CPU 417)
- Интерфейсным модулем в сопряжении с CPU (например, IM 467)
- CP в сопряжении с CPU (например, CP 342-5, CP 443-5)

Существуют «мастера класса 1» для обмена данными в работе процесса и «мастера класса 2» для сервиса и диагностики (например, программирующие устройства).

1.2.3 DP-ведомые

DP-ведомые – это пассивные узлы в PROFIBUS. В SIMATIC S7 различают

- Компактные DP-ведомые
Они ведут себя по отношению к DP-мастеру как единичный модуль;

- Модульные DP-ведомые
Они состоят из нескольких модулей (подмодулей);
- Интеллектуальные (с развитой логикой) DP-ведомые (I-slaves)
Они содержат программу, управляющую (собственными) модулями нижнего уровня.

Компактные ведомые PROFIBUS DP

Образцы DP-ведомых включают ET 200В (версия с модулями цифрового входа/выхода или модулями аналогового входа/выхода; степень защиты IP 20; максимальная скорость передачи данных 12 Мбит/с), ET 200С (жесткая конструкция IP 66/67; различные варианты с цифровыми и аналоговыми входами/выходами; скорость передачи данных 1,5 Мбит/с или 12 Мбит/с) и ET 200L-SC (дискретная модуляция со свободно комбинируемыми модулями цифрового входа/выхода и модулями аналогового входа/выхода; степень защиты IP 20; скорость передачи данных 1,5 Мбит/с). Шлюзы шины, такие как модуль связи (Link) DP/AS-i, ведут себя как компактный ведомый в PROFIBUS.

Модульные ведомые PROFIBUS DP

Примером модульного DP-ведомого является ET 200М. Конструкция соответствует станции S7-300 с направляющей рейкой DIN, модулем электропитания, интерфейсным модулем IM 153 вместо CPU и сигнальными (SM) или функциональными (FM) модулями, количество которых может достигать восьми. Скорость передачи данных может составлять от 9,6 Кбит/с до 12 Мбит/с.

ET 200М также может быть спроектирован с активными шинными модулями, если DP-мастер является станцией S7-400. Это означает, что модули входов/выходов S7-300 могут быть подключены и отключены во время работы при включенном питании. Оставшиеся модули продолжают работать. Между подключаемыми модулями теперь могут быть промежутки.

ET 200М может использоваться с интерфейсным модулем IM 153-3 в резервированной шине в качестве ведомого. IM 153-3 имеет два подключения, одно для DP-мастера в главной станции, другое для DP-мастера в резервной станции.

Интеллектуальные ведомые (I-ведомые) PROFIBUS-DP

Примерами DP-ведомых с развитой логикой могут служить станция S7-300, в которой используется CPU, оснащенный DP-интерфейсом, с возможностью переключения в режим ведомого (например, CPU 315-2DP), или станция S7-300 с CP 342-5 в режиме ведомого.

ET 200X с базовым модулем BM 147/CPU может работать как интеллектуальный DP-ведомый. Он имеет базовый модуль и до 7 модулей расширения. В качестве базовых вы можете использовать «пассивные» базовые модули с цифровыми входами

или выходами, или можно применить «интеллектуальный» базовый модуль BM 147/CPU, способный исполнять пользовательскую программу STEP 7. Модули расширения доступны с цифровыми входами/выходами, аналоговыми входами/выходами, а также их можно использовать как нагрузочные фидеры (переключение и защита любой нагрузки трехфазного переменного тока до 5,5 кВт при 400 В). Базовые модули работают на скоростях передачи данных 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 проектирования и использования в областях с должным уровнем безопасности (Ex-область Зона 1), например, в химической промышленности, а также в областях с меньшей степенью безопасности, таких как производство продуктов питания или напитков.

Протокол для PROFIBUS-PA основан на стандарте EN 50170, том 2 (PROFIBUS-DP), а метод передачи – на IEC 1158-2.

Возможны два метода подключения PROFIBUS-DP к PROFIBUS-PA:

- DP/PA-интерфейс, если сеть PROFIBUS-DP может работать на скорости 45,45 Кбит/с
- модуль связи DP/PA, преобразующий скорости передачи данных PROFIBUS-DP в скорость передачи PROFIBUS-PA.

DP/PA-интерфейс

DP/PA-интерфейс предоставляет подключение полевых устройств PA к PROFIBUS-DP. В PROFIBUS-DP DP/PA-интерфейс является DP-ведомым, работающим на скорости 45,45 Кбит/с. К одному DP/PA-интерфейсу может быть подключено до 31 полевого устройства PA. Эти полевые устройства формируют сегмент PROFIBUS-PA со скоростью передачи данных 31,25 Кбит/с. Взятые вместе все сегменты PROFIBUS-PA составляют разделяемую систему шины PROFIBUS-PA.

DP/PA-интерфейс предоставляется в двух вариантах: не-Ex-версия с выходным током до 400 мА и Ex-версия с выходным током до 100 мА.

Модуль связи DP/PA

Модуль связи DP/PA позволяет подключить полевые устройства PA к PROFIBUS-DP при скорости передачи данных от 9,6 Кбит/с до 12 Мбит/с. Модуль связи DP/PA состоит из интерфейсного модуля IM 157 и DP/PA-интерфейсов (до 5 элементов), соединенных вместе посредством коннекторов шины SIMATIC S7. Он занимает сис-

тему шины, включающую все сегменты PROFIBUS-PA, и преобразует ее в ведомого PROFIBUS-DP. К одному модулю связи DP/PA можно подключить максимум 31 полевое устройство PA.

SIMATIC PDM

SIMATIC PDM (Process Device Manager – Менеджер устройств процесса, ранее SIPROM) является независимым инструментом для параметризации, запуска и диагностики интеллектуальных полевых устройств с функциональностью PROFIBUS-PA или HART. Для параметризации HART-преобразователей (Highway Addressable Remote Transducers – магистральные адресуемые удаленные преобразователи) применяется DDL (Device Description Language – язык описания устройств).

Вы можете оперировать SIMATIC PDM как автономной версией под Windows 9x/NT или как инструментом, интегрированным в STEP 7.

1.2.5 Подключение к AS-интерфейсу

Интерфейс привод-датчик

Интерфейс привод-датчик (actuator-sensor interface – AS-i) – это сетевая система для самого низкого уровня процесса в системах автоматизации. AS-i-мастер управляет максимум 31 AS-i-ведомым посредством 2-проводной AS-i-линии, которая передает как управляющие сигналы, так и питающее напряжение. AS-i-ведомыми могут быть приводы или датчики с возможностями шины или модули AS-i, к которым может быть подключено до 8 бинарных («нормальных») приводов или датчиков.

Сегмент AS-i может иметь длину до 100 м; сегмент может быть удлинён до 2x100 м с помощью повторителя (AS-i-ведомые и модули электропитания AS-i на обоих концах) или расширителя (AS-i-ведомые и модуль электропитания AS-i только на линии, направленной к мастеру).

AS-i-мастер

AS-i-мастер обновляет свои данные и данные всех подключенных AS-i-ведомых за время до 5 мс. Вы можете подключить AS-i-шину непосредственно к SIMATIC S7 с использованием CP 342-2 или к PROFIBUS-DP при помощи модуля связи DP/AS-интерфейс (рисунок 1.4).

AS-i-мастер CP 342-2 может использоваться в станции S7-300 или в станции ET 200M. Он поддерживает два режима работы:

В стандартном режиме CP 342-2 ведет себя как модуль входа/выхода. Он занимает 16 входных байт и 16 выходных байт в аналоговой области адресов (из ее 128 байтов). AS-i-ведомые параметризуются с использованием данных по умолчанию, хранящихся в CP.

В расширенном режиме доступна полная документированная функциональность AS-i-мастера. При использовании поставляемого блока FC в дополнение к стандартному режиму из пользовательской программы могут быть произведены вызовы мастера (передача параметров во время работы, тестирование уставки/действующей конфигурации, проверка и диагностика).

Модуль связи DP/AS-интерфейс позволяет подключить AS-i-приводы и AS-i-датчики к PROFIBUS-DP. В PROFIBUS-DP модуль связи представляет собой модульный DP-ведомый, в AS-интерфейсе он является AS-i-мастером, который может контролировать до 31 AS-i-ведомого. С максимальным числом AS-i-ведомых (31) модуль связи DP/AS-интерфейс занимает 16 входных байт и 16 выходных байт. Скорость передачи данных может быть до 12 Мбит/с.

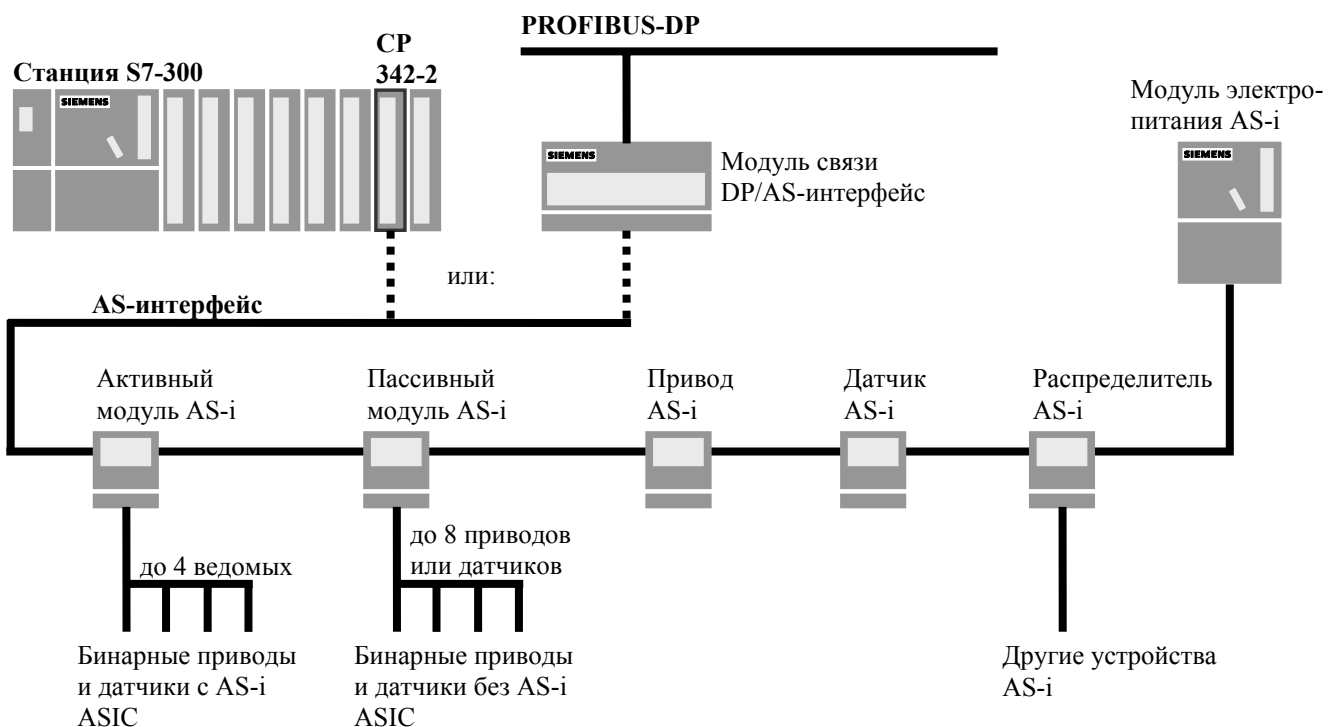


Рисунок 1.4 Подключение системы шины AS-i к SIMATIC S7

Модуль связи DP/AS-интерфейс доступен в двух версиях: усиленный модуль связи DP/AS-интерфейс 65 со степенью защиты IP 66/67 и модуль связи DP/AS-интерфейс 20 со степенью защиты IP 20, которые могут быть установлены с дополнительным командным интерфейсом, что увеличит диапазоны входа и выхода до 20 байт.

1.2.6 Подключение к последовательному интерфейсу

Модуль связи PROFIBUS-DP/RS 232C является преобразователем между интерфейсом RS 232C (V.24) и PROFIBUS-DP. Устройства с интерфейсом RS 232C могут быть подключены к PROFIBUS-DP с помощью модуля связи DP/RS 232C. Модуль

связи DP/RS 232C поддерживает процедуры протоколов 3964R и свободного (free) ASCII.

Модуль связи PROFIBUS-DP/RS 232C подключен к устройству с помощью соединения точка-к-точке (двухточечного, point-to-point). Преобразование к протоколу PROFIBUS-DP происходит в модуле связи PROFIBUS-DP/RS 232C. Данные передаются последовательно в обоих направлениях. В составе одного фрейма передается до 224 байт пользовательских данных.

Скорость передачи данных в PROFIBUS-DP может быть до 12 Мбит/с; RS 232C может работать на скорости до 34 Кбит/с без проверки четности, с проверкой на четность или нечетность, с восемью битами данных и одним стоп-битом.

1.3 Коммуникации

Коммуникации – обмен данными между программируемыми модулями – являются встроенным компонентом SIMATIC S7. Управление почти всеми коммуникационными функциями осуществляет операционная система. Вы можете производить обмен данными между двумя CPU без какого-либо дополнительного аппаратного обеспечения с помощью одного лишь соединительного кабеля. При использовании модулей CP можно организовать мощные сетевые связывающие модули и средства связи с системами других производителей (не Siemens).

СЕТЬ SIMATIC (SIMATIC NET) – это общий термин (термин-зонтик) для коммуникаций SIMATIC. Он представляет обмен информацией между программируемыми контроллерами и между программируемыми контроллерами и устройствами интерфейса человек-машина (HMI-интерфейса). В зависимости от требований к производительности применяются различные пути коммуникаций.

1.3.1 Вводная часть

На рисунке 1.5 изображены наиболее важные коммуникационные объекты. Вы имеете станции SIMATIC или устройства других производителей, между которыми вам нужно наладить обмен данными. Здесь вам потребуются модули с коммуникационными возможностями.

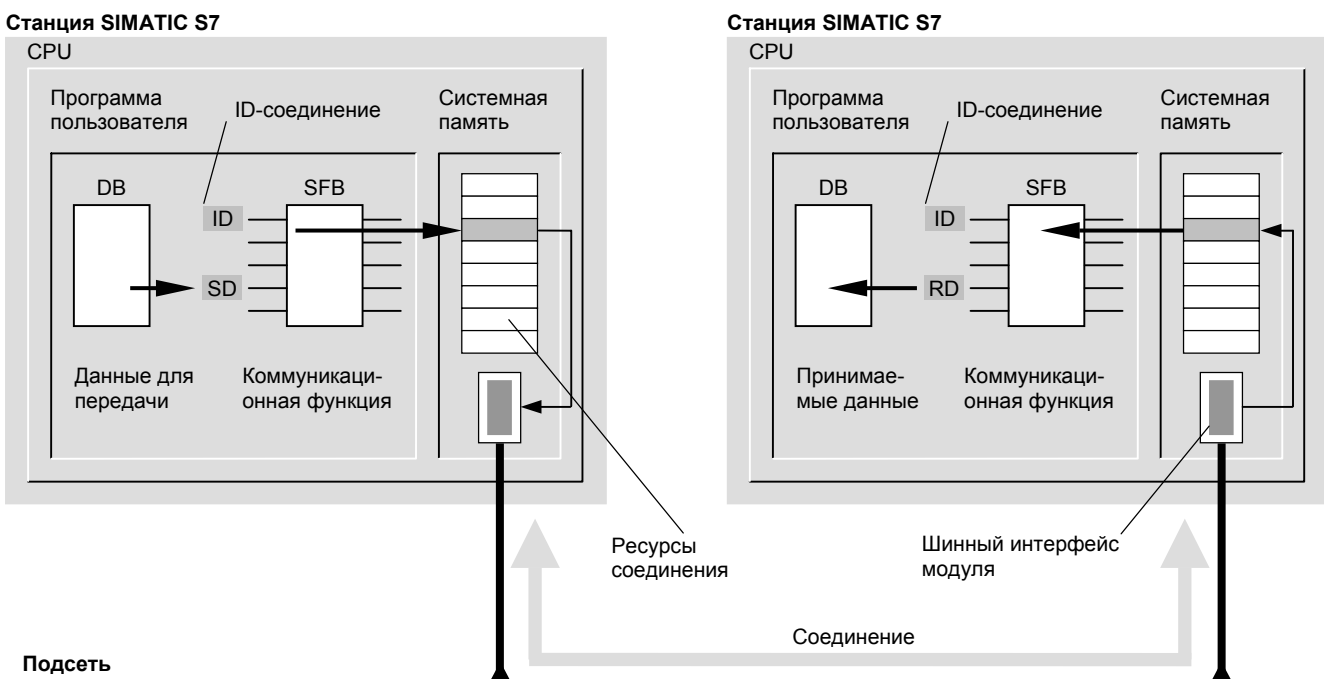


Рисунок 1.5 Обмен данными между двумя станциями SIMATIC S7

Благодаря SIMATIC S7 все CPU оснащены MPI-интерфейсом, через который они могут управлять коммуникациями. Кроме того, имеются коммуникационные процессоры (CP), которые позволяют производить обмен данными на более высоких скоростях передачи и по разным протоколам. Вы должны связать эти модули с помощью сетей. Сеть – это аппаратное соединение между коммуникационными узлами.

Обмен данными происходит через «соединение» в соответствии с особым исполнительным планом («коммуникационный сервис»), который основан, кроме прочего, на определенной координационной процедуре («протокол»). S7-соединение является стандартным, например, между S7-модулями с коммуникационной способностью.

Сеть

Сеть – это соединение между несколькими устройствами для коммуникационных целей. Она состоит из одной или более идентичных или разных подсетей, соединенных вместе.

Подсеть

В подсети все коммуникационные узлы связаны посредством аппаратного соединения с однородными физическими характеристиками и параметрами передачи, такими как скорость передачи данных, и они обмениваются данными с помощью одной (разделяемой) процедуры доступа к сети (передачи). SIMATIC в качестве подсетей принимает MPI, PROFIBUS, промышленный Ethernet (Industrial Ethernet) и соединение «точка-к-точке» (point-to-point – PTP).

Коммуникационный сервис

Коммуникационный сервис (служба) определяет, как происходит обмен данными между коммуникационными узлами, и как данные должны быть обработаны. Это основывается на протоколе, описывающем, наряду с прочим, координационную процедуру между коммуникационными узлами.

В SIMATIC приняты следующие сервисы: S7-функции, PROFIBUS-DP, PROFIBUS-FMS, PROFIBUS-FDL (SDA), ISO-транспорт, ISO-on-TCP и коммуникация глобальных данных (global data communications).

Соединение

Соединение (связь) определяет коммуникационные отношения между двумя коммуникационными узлами. Это логическое назначение двух узлов для выполнения особого коммуникационного сервиса, а также содержит определенные свойства, такие как тип соединения (динамическое, статическое), и способ его установки.

SIMATIC принимает следующие типы соединений: S7-соединение, S7-соединение (отказоустойчивое), FMS- и FDL-соединения, соединение ISO-транспорт, ISO-on-TCP- и TCP-соединение, UDP-соединение и e-mail-соединение.

Коммуникационные функции

Коммуникационные функции – это интерфейс программы пользователя с коммуникационным сервисом. С точки зрения внутренних коммуникаций SIMATIC S7 коммуникационные функции встроены в операционную систему CPU и вызываются через системные блоки. Для коммуникации с устройствами других производителей через коммуникационные процессоры предоставлены загружаемые блоки (loadable blocks).

Обзор коммуникационных объектов

Таблица 1.1 показывает отношения между подсетями, модулями с коммуникационными возможностями и коммуникационными сервисами.

Таблица 1.1 Коммуникационные объекты

Подсеть	Модули	Коммуникационный сервис	Конфигурирование, интерфейс
MPI	Все CPU	Коммуникация глобальных данных	Таблица глобальных данных (GD table)
		Внешние SFC-коммуникации станции	SFC-вызовы
		SFB-коммуникации (активны только у S7-400)	Таблица соединений (connection table), FB-вызовы
PROFIBUS	CPU с DP-мастером	PROFIBUS-DP (мастер, также возможен ведомый)	Hardware configuration (конфигурирование аппаратуры), входы / выходы, SFC-вызовы
		Внутренние SFC-коммуникации станции	SFC-вызовы
	IM 467	PROFIBUS-DP (мастер или ведомый)	Hardware configuration, входы / выходы, SFC-вызовы
		Внутренние SFC-коммуникации станции	SFC-вызовы
	CP 342-5 CP 443-5 Расширенный	PROFIBUS-DPL PROFIBUS-DP (мастер или ведомый)	NCM, таблица соединений, SEND/RECEIVE
		Внутренние SFC-коммуникации станции	SFC-вызовы
		SFB-коммуникации (активны только у S7-400)	Таблица соединений, SFB-вызовы
	CP 343-5 CP 443-5 Базовый	PROFIBUS-FMS PROFIBUS-FDL	NCM, таблица соединений, FMS-интерфейс, SEND/RECEIVE
		Внутренние SFC-коммуникации станции	SFC-вызовы
		SFB-коммуникации (активны только у S7-400)	Таблица соединений, SFB-вызовы

Таблица 1.1 (Продолжение)

Подсеть	Модули	Коммуникационный сервис	Конфигурирование, интерфейс
Промыш- ленный Ethernet	CP 343-1 CP 443-1	Транспортный протокол ISO и TCP/IP	NCM, таблица соединений, SEND/RECEIVE
		SFB-коммуникации (активны только у S7-400)	Таблица соединений, SFB-вызовы
	CP 343-1 IT CP 443-1 IT	Транспортный протокол ISO и TCP/IP, IT-коммуникация	NCM, таблица соединений, SEND/RECEIVE
		SFB-коммуникации (активны только у S7-400)	Таблица соединений, SFB-вызовы

NCM – это конфигурационное программное обеспечение для CP; NCM применимо к PROFIBUS и пром. Ethernet.

1.3.2 Подсети

Подсети – это коммуникационные пути с одинаковыми физическими характеристиками и коммуникационными процедурами. Для коммуникации в SIMATIC-менеджере подсети являются центральными объектами. Подсети отличаются по производительным возможностям:

- MPI
Недорогой метод объединения в сеть нескольких устройств SIMATIC с малыми объемами данных.
- PROFIBUS
Высокоскоростной обмен малыми и средними объемами данных, используемые главным образом с распределенными входами/выходами.
- Промышленный Ethernet
Связь между компьютерами и программируемыми контроллерами для высокоскоростного обмена большими объемами данных.
- Point-to-point (PTP-соединение)
Последовательная связь между двумя коммуникационными партнерами по специальному протоколу.

Начиная с пятой версии STEP 7, можно использовать программирующие устройства, чтобы через подсети связаться со станциями SIMATIC S7, скажем, с целью программирования или параметризации. Шлюзы между подсетями должны располагаться в станциях S7 с «возможностью маршрутизации».

MPI

Каждый CPU оснащен «интерфейсом с возможностью мультиточечности» (мультиточечный интерфейс – multipoint interface, или MPI). Он позволяет организовывать подсети, в которых CPU, устройства интерфейса человек-машина и устройства программирования могут обмениваться данными друг с другом. Обмен данными управляется через патентованный протокол Siemens.

В качестве передающей среды MPI использует либо экранированную витую пару, либо стеклянный или пластиковый оптоволоконный кабель. Длина кабеля в сегменте шины может достигать 50 м. Она может быть увеличена путем добавления повторителей RS485 (до 1100 м) или оптических модулей связи (более 100 км). Обычная скорость передачи данных составляет 187,5 Кбит/с.

Максимальное количество узлов – 32. Каждый узел имеет доступ к шине определенный промежуток времени и может отсылать фреймы данных. По окончании данного временного интервала узел передает права доступа следующему узлу (процедура доступа «передача маркера»).

По сети MPI можно производить обмен данными между CPU с помощью коммуникации глобальных данных, внешних SFC-коммуникаций станции или SFB-коммуникаций. Дополнительных модулей не требуется.

PROFIBUS

PROFIBUS – это сокращение от «Process Fieldbus» («Полевая шина процесса»), и она является независимым стандартом, соответствующим EN 50170, для объединения в сеть полевых устройств.

В качестве передающей среды используется экранированная витая пара либо стеклянный или пластиковый оптоволоконный кабель. Длина кабеля в сегменте шины зависит от скорости передачи данных; она составляет 100 м при наибольшей скорости передачи (12 Мбит/с) и 1000 м при наименьшей (9,6 Кбит/с). Дальность сети может быть увеличена с помощью повторителей или оптических модулей связи.

Максимальное количество узлов – 127; различают активные и пассивные узлы. Активный узел получает права доступа к шине на определенный интервал времени, в течение которого может отсылать фреймы данных. По окончании интервала узел передает права доступа следующему узлу (процедура доступа «передача маркера»). Если активному узлу (мастеру) назначены пассивные узлы (ведомые), мастер, пока обладает правами доступа, обменивается данными с назначенными ему ведомыми. Пассивные узлы прав доступа не получают.

Вы реализуете подключение распределенных входов/выходов через сеть PROFIBUS; соответствующий коммуникационный сервис PROFIBUS-DP подразумевается. Можно использовать либо CPU со встроенным или подключаемым DP-мастером, либо соответствующие CP. Посредством данной сети вы также можете оперировать внутренними SFC-коммуникациями станции или SFB-коммуникациями.

Вы можете передавать данные при помощи PROFIBUS-FMS и PROFIBUS-FDL, используя соответствующие CP. Имеются загружаемые блоки (FMS-интерфейс или SEND/RECEIVE-интерфейс), которые можно применить в качестве интерфейса для пользовательской программы.

Промышленный Ethernet

Промышленный Ethernet – это подсеть, соединяющая компьютеры и программируемые контроллеры, для применения в индустриальной отрасли, определенная международным стандартом IEEE 802.3.

Физическое электронное соединение представляет собой коаксиальный кабель с двойным экранированием или промышленную витую пару, а оптическое соединение выполняется на основе стеклянного оптоволоконного кабеля. Дальность электронной сети составляет до 1,5 км, оптической – до 4,5 км. Скорость передачи данных установлена на уровне 10 Мбит/с.

Более 1000 узлов может быть объединено в сеть с помощью промышленного Ethernet. Перед осуществлением доступа к шине каждый узел проверяет, передает ли в данный момент другой узел. Если это так, то узел ожидает в течение произвольного интервала времени перед следующей попыткой получить доступ (процедура CSMA/CD-доступа). Все узлы имеют равные права доступа.

Вы также можете производить обмен данными через промышленный Ethernet с помощью SFB-коммуникаций, и можно использовать функции S7. Для промышленного Ethernet вам потребуются соответствующие CP, и также вы можете организовать транспортные ISO-соединения или ISO-on-TCP-соединения и управлять с помощью SEND/RECEIVE-интерфейса.

Соединение «точка-к-точке» (point-to-point)

Соединение «точка-к-точке» (PTP) позволяет производить обмен данными через последовательную связь. Двухточечное соединение управляется SIMATIC-менеджером как подсеть и сходным образом конфигурируется.

Передающая среда – электрический кабель с зависящим от интерфейса назначением. В качестве интерфейсов доступны RS 232C (V.24), 20 мА (TTY) и RS 422/485. Скорость передачи данных находится в диапазоне от 300 бит/с до 19,2 Кбит/с для интерфейса 20 мА или составляет 76,8 Кбит/с для RS 232C и RS 422/485. Длина кабеля зависит от физического интерфейса и скорости передачи данных; она достигает 10 м в случае RS 232C, 1000 м с интерфейсом 20 мА при скорости 9,6 Кбит/с и 1200 м с RS 422/485 при 19,2 Кбит/с.

AS-интерфейс

AS-интерфейс (интерфейс привода/датчика, AS-i) объединяет в сеть должным образом спроектированные бинарные датчики и приводы в соответствии со спецификацией AS-интерфейса IEC TG 178. AS-интерфейс в качестве подсети в SIMATIC-менеджере не фигурирует; только AS-I-мастер настраивается с помощью аппаратного конфигурирования (hardware configuration) или сетевого конфигурирования (network configuration).

Средой передачи служит неэкранированная витая пара, которая снабжает приводы и датчики данными и электропитанием (питание требуется). Дальность сети с повторителями может достигать 300 м. Скорость передачи данных установлена на уровне 167 Кбит/с.

Мастер управляет максимум 31 ведомым посредством циклического сканирования, что гарантирует определенное время отклика.

1.3.3 Коммуникационные сервисы

Обмен данными через подсети управляется различными коммуникационными сервисами – в зависимости от выбранного соединения. Используются следующие коммуникационные службы и их свойства:

- Функции программирующего устройства (PG): тестирование, запуск и сервисные функции; используются программирующим устройством, например, для выполнения функции «мониторинг переменных» или «чтение диагностического буфера» или для загрузки программ пользователя.
- HMI-функции: функции интерфейса человек-машина; используются подсоединенными OP, например, для чтения или записи переменных.
- SFB-коммуникации: управляемый событиями сервис для обмена большими объемами данных. В пользовательской программе он запускается SFB-вызовами и предоставляет функции модификации и наблюдения; статические, сконфигурированные соединения.
- SFC-коммуникации: управляемый событиями сервис для обмена данными объемом до 76 байт за транзакцию. В пользовательской программе он запускается SFC-вызовами; динамические, несконфигурированные соединения.

S7-функции могут быть выполнены через подсети MPI, PROFIBUS и промышленный Ethernet.

Коммуникация глобальных данных позволяет производить обмен малыми объемами данных между несколькими CPU без дополнительных усилий пользователя по программированию. Передача может быть циклической или управляемой событиями.

Коммуникация глобальных данных – это широковещательная процедура; прием данных не подтверждается. О состоянии коммуникации сообщается.

Коммуникация глобальных данных по шине MPI или K-шине является единственной возможной.

С помощью **PROFIBUS-DP** происходит обмен данными между мастером и ведомыми через распределенные входы/выходы. Коммуникация прозрачна и стандартизована в соответствии с EN 50170 том 2. С помощью этого сервиса ведомые SIMATIC

S7 и ведомые других стандартов (не SIMATIC) могут быть доступны через сеть PROFIBUS.

PROFIBUS-FMS (Fieldbus Message Specification – спецификация сообщений полевой шины) предоставляет сервисы для передачи структурированных переменных (FMS-переменные) в соответствии с EN 50170 том 2. Коммуникация осуществляется исключительно с использованием статических соединений по подсети PROFIBUS.

PROFIBUS-FDL (Fieldbus Data Link – связывание данных полевой шины) передает данные с использованием функции SDA (Send Data with Acknowledge – пересылка данных с подтверждением), стандартизированной в соответствии с EN 50170 том 2. Коммуникация осуществляется с использованием статических соединений. С помощью PROFIBUS-FDL может быть произведен обмен данными, например, с контроллером SIMATIC S5 по подсети PROFIBUS.

Коммуникационный сервис ISO-транспорт позволяет передавать данные в соответствии с ISO 8073 класс 4. Коммуникация осуществляется через статические соединения. С помощью ISO-транспорта можно обмениваться данными, например, с контроллером SIMATIC S5 по промышленному Ethernet.

Коммуникационный сервис ISO-on-TCP соответствует стандарту TCP/IP с расширением RFC 1006. Коммуникация осуществляется с использованием статических соединений через промышленный Ethernet.

1.3.4 Соединения

Соединение (connection) в зависимости от выбранного коммуникационного сервиса может быть динамическим или статическим. Динамические соединения не конфигурируются; их создание или снятие (очищение) управляется событиями («коммуникации через неконфигурированные соединения»). С коммуникационным партнером может быть только одно неконфигурированное соединение.

Статические соединения конфигурируются в таблице соединений; они создаются при запуске и остаются на протяжении всего исполнения программы («коммуникации через сконфигурированные соединения»). С одним коммуникационным партнером может быть установлено несколько параллельных связей. Для выбора нужного коммуникационного сервиса в конфигурировании сети используется «Connection type» («Тип соединения») (смотрите параграф 2.4 «Конфигурирование сети»).

В случае применения функций S7 вам не нужно с помощью сетевого конфигурирования настраивать соединения для коммуникаций глобальных данных и PROFIBUS-DP или для SFC-коммуникаций. Коммуникационные партнеры для коммуникаций глобальных данных определяются в таблице глобальных данных (global data table); в случае PROFIBUS-DP и SFC-коммуникаций партнеры определяются через адреса узлов.

Ресурсы соединений

Каждому соединению требуются ресурсы соединений на участвующем коммуникационном партнере для концевой точки соединения или точки перехода (транзитной) в модуле CP. Если, к примеру, функции S7 выполняются через MPI-интерфейс CPU, соединение назначено в CPU; те же функции через MPI-интерфейс CP занимают одно соединение в CP и одно соединение в CPU.

Каждый CPU имеет определенное количество возможных соединений. Одно соединение зарезервировано для программирующего устройства и одно – для OP (они не могут быть использованы для других целей).

Ресурсы соединений временно требуются также для «несконфигурированных соединений» в SFC-коммуникациях.

1.4 Адреса модулей

1.4.1 Путь сигнала

При монтаже проводных связей оборудования вы определяете, какие сигналы подключены к программируемому контроллеру, и где осуществлено подключение (рисунок 1.6).

Входной сигнал, например, сигнал от переключателя мгновенного контакта +HP01-S10 (служит для «Включения мотора», «Switch motor on»), идет к модулю входа, где он подключается к определенному терминалу. Терминал имеет «адрес», называемый адресом входа/выхода (I/O-адрес) (например, байт 5, бит 2).

Далее перед каждым стартом исполнения программы CPU автоматически копирует сигнал в образ входа процесса, где он затем доступен как адрес «входа» (I 5.2, например). Выражение «I 5.2» - это абсолютный адрес.

Теперь вы можете дать имя этому входу путем присваивания буквенно-цифрового символа, соответствующего этому входному сигналу (такое как «Включить мотор»), абсолютному адресу в таблице символов (symbol table). Выражение «Включить мотор» - это символический адрес.

1.4.2 Адрес слота

Каждый слот имеет фиксированный адрес в программируемом контроллере (станция S7). Этот адрес слота состоит из номера монтажной стойки и номера слота. При использовании адреса слота модуль имеет уникальное описание («географический адрес»).

Если модуль содержит интерфейсные карты, каждой из них также присваивается адрес подмодуля. Таким образом, каждое подключение двоичного или аналогового сигнала в системе имеет свой уникальный адрес.

Соответственно, у модулей распределенных входов/выходов также есть «географический адрес». В этом случае номер системы DP-мастера и номер станции заменяют номер стойки.

Вы используете программу «Hardware Configuration» пакета STEP 7 для планирования конфигурации аппаратного обеспечения станции S7 согласно физическому расположению модулей. Этот инструмент также позволяет установить стартовые адреса модулей и параметризовать модули (подробнее об это читайте в параграфе 2.3 «Конфигурирование станций»).

1.4.3 Стартовый адрес модуля

Кроме адреса слота, который определяет слот, каждый модуль имеет стартовый адрес, определяющий расположение в пространстве логических адресов (пространстве

адресов входов/выходов). Пространство адресов входов/выходов начинается с адреса 0 и заканчивается верхним пределом, зависящим от CPU.

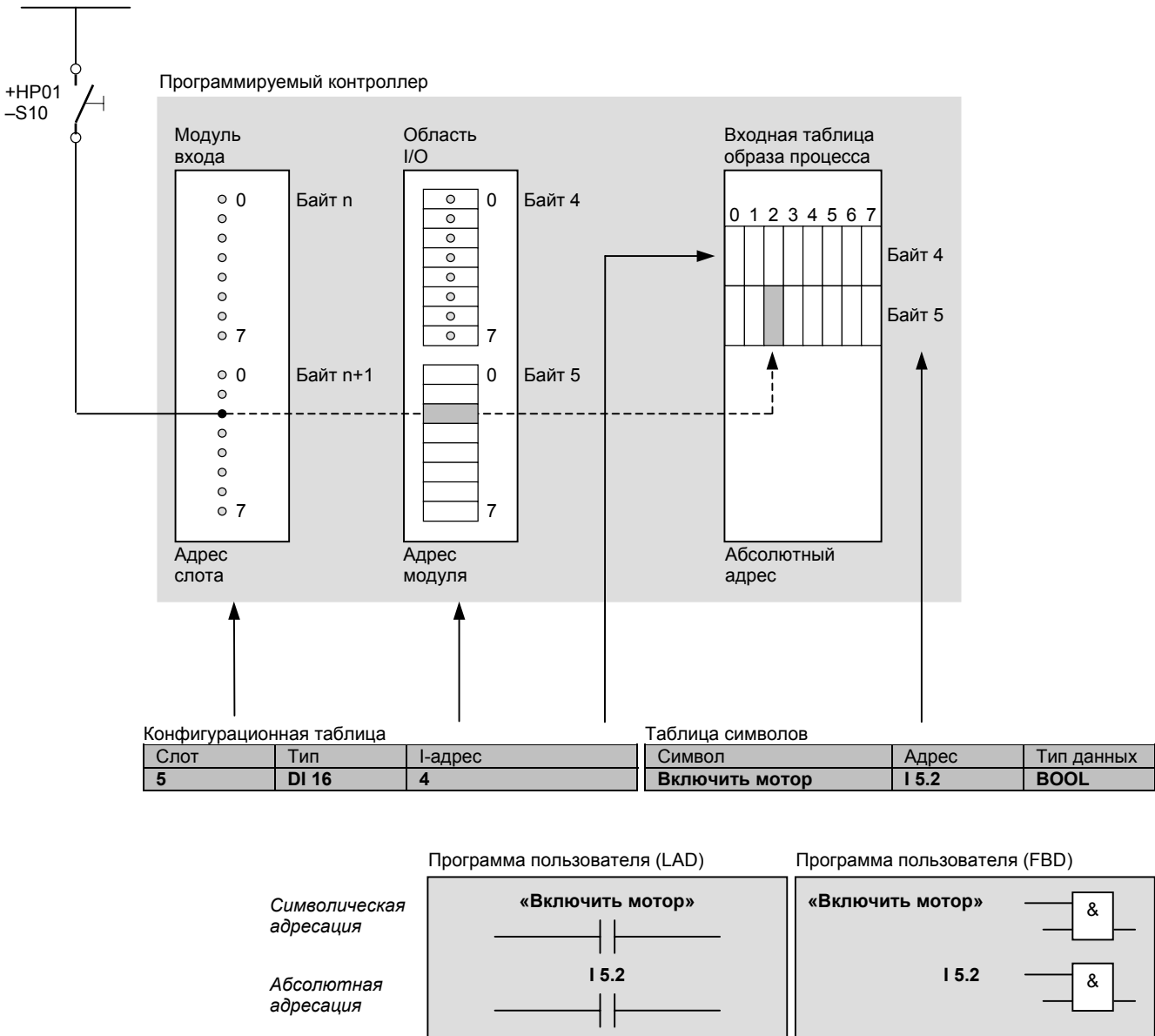


Рисунок 1.6

Зависимость между адресом модуля, абсолютным адресом и символическим адресом (Путь сигнала от датчика к сканированию в программе)

Стартовый адрес модуля определяет, как входные и выходные сигналы адресуются программой (определяется доступ к ним). В случае цифровых модулей отдельные сигналы (биты) собираются в группы по восемь, называемые байтами. Есть модули с одним, двумя или четырьмя байтами. Эти байты имеют соответствующие адреса – 0, 1, 2 и 3; адресация байт начинается со стартового адреса модуля. Пример: в случае цифрового модуля с четырьмя байтами и стартовым адресом 8 отдельные байты дос-

тупны по адресам 8, 9, 10 и 11. В случае аналоговых модулей отдельные аналоговые сигналы (напряжения, токи) называются «каналами», каждый из которых занимает два байта. В зависимости от конструкции применяются аналоговые модули с 2, 4, 8 и 16 каналами, соответствующих 4, 8, 16 или 32 байтам адресного пространства.

При включении питания (если нет конфигурации установки) CPU задает каждому модулю стартовый адрес, который зависит от типа модуля, слота и стойки. Этот стартовый адрес модуля соответствует его первому байту (0-вому). Вы можете увидеть этот адрес в конфигурационной таблице (configuration table).

В S7-3xx со встроенным DP-интерфейсом, системах S7-318 и S7-400 вы можете изменять этот адрес. В вашем распоряжении имеется опция назначения стартовых адресов в пределах допустимого пространства адресов. Также вы можете воспользоваться опцией назначения различных стартовых адресов для входов и выходов в гибридных цифровых или аналоговых модулях. FM и CP обычно резервируют одинаковый стартовый адрес для входов и выходов.

Подобно централизованным модулям модули распределенных входов/выходов (станции) резервируют определенное количество байт в пространстве адресов входов/выходов. Адреса централизованных модулей и модулей распределенных входов/выходов не должны перекрываться.

Соответственно оборудованные DP-ведомые могут быть параметризованы так, что определенное количество байт составляют последовательные (логически построенные) данные для их передач (консистентные данные). Эти ведомые отображают адрес входа/выхода только одного байта, через который они адресуются с помощью системных функций SFC 14 DPRD_DAT и SFC 15 DPWR_DAT.

Цифровые модули обычно упорядочены в соответствии с адресом в образе процесса, поэтому их состояния сигнала могут быть автоматически обновлены, и к ним можно обращаться с использованием адресных областей «Вход» («Input») и «Выход» («Output»). Аналоговые модули, FM и CP получают адрес, которого в образе процесса нет.

1.4.4 Диагностический адрес

Соответствующим образом оснащенные модули могут предоставлять диагностические данные, которые вы можете оценивать в своей программе. Если централизованные модули имеют адрес пользовательских данных (стартовый адрес модуля), то при чтении диагностических данных вы обращаетесь к модулю через этот адрес. Если у модулей нет адреса пользовательских данных (например, модули электропитания), или они являются частью распределенных входов / выходов, то для этой цели существует диагностический адрес.

Диагностический адрес всегда является адресом во входной подобласти области входов/выходов и занимает один байт. Длина пользовательских данных этого адреса равна нулю; если он расположен в образе процесса, что разрешено, то при обновлении образа процесса CPU не берет его во внимание.

STEP 7 автоматически инициализирует отсчет диагностического адреса по убыванию, начиная с наибольшего возможного адреса входов/выходов. Вы можете изменить диагностический адрес с помощью функции Hardware Configuration.

Диагностические данные могут быть прочитаны только специальными системными функциями; доступ к этому адресу с помощью загрузочных операторов эффекта не даст (обратитесь также к параграфу 20.4.1 «Адресация распределенных входов/выходов»).

1.4.5 Адреса для узлов шины

Адреса узлов, номер станции

Каждая DP-станция (например, DP-мастер, DP-ведомый, устройство программирования) в PROFIBUS имеет дополнительный адрес узла, с помощью которого она может быть уникальным образом адресована на шине.

MPI-адрес

Модули, являющиеся узлами в сети MPI (устройства CPU, FM и CP), также имеют MPI-адрес. Этот адрес играет решающую роль при связи с программирующими устройствами, устройствами интерфейса человек-машина и для коммуникаций глобальных данных.

Заметьте, что в случае старших моделей CPU S7-300 модули FM и CP, работающие в одной станции, получают MPI-адрес, производный от MPI-адреса CPU.

В случае более новых CPU S7-300 MPI-адреса модулей FM и CP одной станции могут быть определены независимо от MPI-адреса CPU.

В случае CPU 318 модули с MPI-соединением расположены в своих собственных сегментах, поэтому у них нет MPI-адреса. Они адресуются устройством программирования по номеру стойки и слота.

1.5 Области адресов

Области адресов, имеющиеся в каждом программируемом контроллере, представляют собой

- периферийные входы и выходы,
- образ входов процесса и образ выходов процесса,
- область памяти меркеров,
- функции таймера и счетчика (им посвящены главы 7 «Таймеры» и 8 «Счетчики»),
- L-стек (смотрите параграф 18.1.5 «Временные локальные данные»).

В зависимости от программы пользователя к этому могут добавиться кодовые блоки и блоки данных с локальными (внутриблочными) переменными.

1.5.1 Область пользовательских данных

В SIMATIC S7 каждый модуль может иметь две области адресов: область данных пользователя, которая может быть напрямую адресована с помощью операторов Load (Загрузить) и Transfer (Передать), и область системных данных для передачи записей данных.

При обращении к модулям нет разницы, находятся ли они в стойках с централизованной конфигурацией или используются как распределенные входы/выходы. Все модули занимают одно (логическое) пространство адресов.

Свойства пользовательских данных модуля зависят от типа модуля. В случае сигнальных модулей они являются или цифровыми или аналоговыми входными/выходными сигналами, а в случае функциональных модулей и коммуникационных процессоров они могут быть, например, управляющей информацией или информацией о состоянии. Объем пользовательских данных определяется модулем. Имеются модули, занимающие один, два, четыре или более байт этой области. Адресация всегда начинается с адреса нулевого байта. Адрес нулевого байта – это стартовый адрес модуля; это предусмотрено в конфигурационной таблице.

Пользовательские данные представляют область адресов входов/выходов, подразделенную, в зависимости от направления передачи, на периферийные входы (peripheral inputs, PI) и периферийные выходы (peripheral outputs, PQ). Если пользовательские данные находятся в области образов процесса, то при обновлении образов процесса CPU автоматически управляет передачами.

Периферийные входы

Область адресов периферийных входов (PI) используется при считывании из области пользовательских данных модулей входов. Часть области адресов PI указывает на образ процесса. Эта часть всегда начинается с I/O-адреса 0; размер области определяется моделью CPU.

С помощью операции прямого чтения входов/выходов (Direct I/O Read) вы можете получить доступ к модулям, чьи интерфейсы не соединены с образом входов процесса (к примеру, модули аналоговых входов). Сигнальные состояния модулей, соединенных с образом входов процесса, могут быть также прочитаны с использованием операции прямого чтения (Direct Read). В этом случае сканируются мгновенные сигнальные состояния входных битов. Заметьте, что эти сигнальные состояния могут отличаться от соответствующих входов в образе процесса, поскольку образ входов процесса обновляется в начале программного сканирования.

Периферийные входы могут занимать те же абсолютные адреса, что и периферийные выходы.

Периферийные выходы

Вы используете область адресов периферийных выходов (PQ), когда записываете значения в область пользовательских данных выходного модуля. Часть области адресов PQ указывает на образ процесса. Эта часть всегда начинается с I/O-адреса 0; размер области определяется CPU.

С помощью операции Direct I/O Write (прямая запись входов/выходов) вы можете обратиться к модулям, чьи интерфейсы не связаны с образом выходов процесса (таких, как модули аналоговых выходов). Сигнальные состояния модулей, которые управляются образом выходов процесса, также могут быть подвержены прямому воздействию. Сигнальные состояния битов выхода в этом случае немедленно изменяются. Заметьте, что операция Direct I/O Write также обновляет сигнальные состояния соответствующих модулей в образе выходов процесса! Таким образом, различия между образом выходов процесса и сигнальными состояниями в модулях выходов отсутствуют.

Периферийные выходы могут резервировать те же абсолютные адреса, что и периферийные входы.

1.5.2 Образ процесса

Образ процесса содержит образ модулей цифрового входа и цифрового выхода, и поэтому подразделен на образ входов и образ выходов процесса. Образ входов процесса доступен через область адресов для входов (I), образ выходов процесса – через область адресов для выходов (Q). Как правило, машина или процесс управляется посредством входов и выходов.

Образ процесса может быть подразделен на вспомогательные образы процесса, которые могут обновляться либо автоматически, либо через программу пользователя. Более подробно об этом рассказывается в параграфе 20.2.1 «Обновление образа процесса».

В CPU S7-300, а также в CPU S7-400, выпускающихся с октября 1998 г., вы можете использовать не занятые модулями адреса образа процесса в качестве дополнительной области памяти подобно области памяти меркеров. Это относится и к образу входов и к образу выходов процесса.

В соответствующим образом оснащенных CPU, скажем, CPU 417, размер образа процесса может быть параметризован. При увеличении образ процесса соответственно уменьшается размер рабочей памяти. После изменения размера образа процесса CPU выполняет инициализацию рабочей памяти с тем же результатом, как при холодном рестарте.

Входы

Вход (input) является образом соответствующего бита в модуле цифрового входа. Сканирование входа представляет собой то же самое, что и сканирование самого бита в модуле. Перед исполнением программы в каждом программном цикле операционная система CPU копирует сигнальные состояния из модуля в образ входов процесса.

Использование образа входов процесса имеет много преимуществ:

- Входы могут быть отсканированы, и после этого доступ к ним может осуществляться побитно (биты входов/выходов напрямую не адресуются).
- Сканирование входа происходит намного быстрее, чем доступ к модулю входа (например, не тратится время на переходное (динамическое) восстановление на шине входов / выходов, и время отклика системной памяти становится короче, чем время отклика модуля). Поэтому программа выполняется намного быстрее.
- Сигнальное состояние входа на протяжении всего программного цикла остается неизменным (сохраняется согласованность данных в течение программного цикла). Когда бит модуля входа меняется, изменение сигнального состояния передается на вход на старте следующего программного цикла.
- Входы могут быть также установлены или сброшены, так как они расположены в памяти с произвольным доступом (random access memory). Модули цифрового входа доступны только для чтения. Входы могут быть установлены во время отладки или запуска для моделирования состояний датчиков, упрощая тем самым тестирование программ.

Эти преимущества компенсируют увеличение времени отклика программы (обратитесь также к параграфу 20.2.4 «Время отклика»).

Выходы

Выход (output) является образом соответствующего бита в модуле цифрового выхода. Установка выхода – это то же самое, что и установка бита самого модуля выхода. Операционная система CPU копирует сигнальное состояние из образа выходов процесса в модуль.

Использование образа выходов процесса включает в себе много преимуществ:

- Выходы могут быть установлены и сброшены побитно (прямая адресация битов входов/выходов невозможна).
- Установка выхода намного быстрее, чем доступ к модулю выхода (например, не тратится время на переходное (динамическое) восстановление на шине входов/выходов, и время отклика системной памяти становится короче, чем время отклика модуля). Поэтому программа выполняется намного быстрее.
- Изменение сигнального состояния на выходах во время программного цикла не влияет на биты модуля выхода. Это сигнальное состояние выходов в конце программного цикла передается в модуль.
- Выходы также могут сканироваться (опрашиваться), потому что они расположены в памяти с произвольным доступом. Пока возможна запись в модули цифрового входа, чтение их недоступно. Сканирование и связывание выходов делает дополнительное хранение бита выхода, предназначенного для сканирования, обязательным.

Эти преимущества компенсируют увеличение времени программного отклика. В параграфе 20.2.4 «Время отклика» содержится описание формирования времени отклика программируемого контроллера.

1.5.3 Память меркеров

Область, называемая памятью меркеров, содержит объекты, рассматриваемые как «вспомогательные контакторы» контроллера. Память меркеров используется главным образом для хранения бинарных сигнальных состояний. Биты в этой области могут использоваться как выходы, но таковыми они не являются. Память меркеров расположена в области системной памяти CPU и поэтому доступна в любой момент времени. Количество битов в меркерной памяти определяется CPU.

Память меркеров используется для хранения промежуточных результатов, которые имеют силу вне границ блока и обрабатываются более чем в одном блоке. Помимо данных в глобальных блоках данных для хранения промежуточных результатов доступны:

- Временные локальные данные (temporary local data), доступные во всех блоках, но действующие для вызова только текущего блока;

- Статические локальные данные (static local data), которые доступны только в функциональных блоках, но имеют силу для вызовов множества блоков.

Реманентная (сохраняющаяся) меркерная память

Часть памяти меркеров может быть определена как «сохраняющаяся» (обладающая способностью к запоминанию), это означает, что биты в этой части меркерной памяти сохраняют свои сигнальные состояния даже при отключении питания. Реманентная способность определена с нулевого байта и заканчивается в назначенном месте. Эта способность устанавливается при параметризации CPU. За дополнительной информацией обратитесь к параграфу 22.2.3 «Реманентность».

Тактовый меркер

В контроллере для многих процедур требуется периодический сигнал. Такой сигнал создается при помощи таймеров (генераторов тактовых импульсов), циклических (watchdog) прерываний (контролируемое по времени исполнение программы) или просто с помощью тактового меркера (clock memory).

Тактовый меркер состоит из битов, чьи сигнальные состояния периодически меняются с коэффициентом заполнения (отношением длительностей положительного и отрицательного импульсов) 1:1. Биты объединяются в байт и соответствуют фиксированным частотам (рисунок 1.7). Номер меркерного байта, объявленного тактовым, определяется при параметризации CPU. Обратите внимание, что обновление тактового меркера асинхронно к выполнению главной программы.

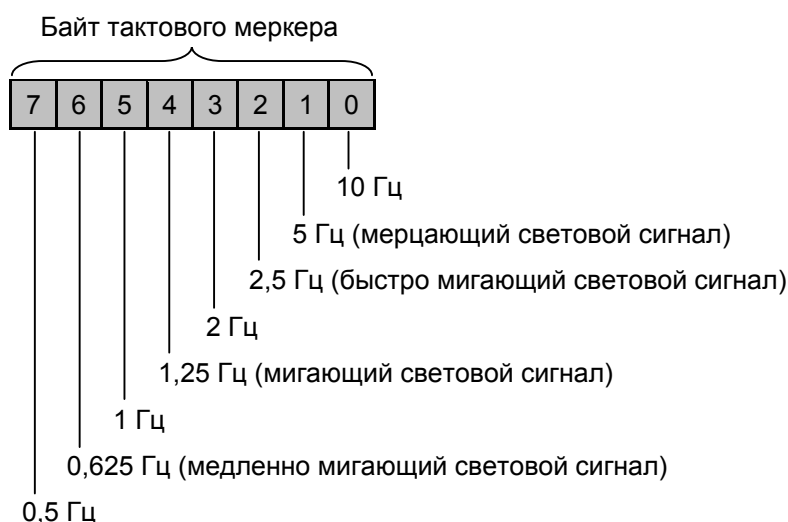


Рисунок 1.7 Содержимое байта тактового меркера

Глава 2
Программное обеспечение STEP 7

Содержание главы 2

2	<u>Программное обеспечение STEP 7</u>	4
2.1	<u>Базовый пакет STEP 7</u>	4
2.1.1	<u>Инсталляция</u>	4
2.1.2	<u>Авторизация</u>	5
2.1.3	<u>SIMATIC-менеджер</u>	5
2.1.4	<u>Проекты и библиотеки</u>	8
2.1.5	<u>Онлайновая помощь</u>	10
2.2	<u>Редактирование проектов</u>	11
2.2.1	<u>Создание проектов</u>	11
2.2.2	<u>Управление, перегруппировка, архивирование</u>	13
2.2.3	<u>Версии проектов</u>	15
2.3	<u>Конфигурирование станций</u>	17
2.3.1	<u>Компоновка модулей</u>	19
2.3.2	<u>Адресация модулей</u>	19
2.3.3	<u>Параметризация модулей</u>	20
2.3.4	<u>Объединение модулей в сеть с помощью MPI</u>	21
2.3.5	<u>Наблюдение и модифицирование модулей</u>	21
2.4	<u>Конфигурирование сети</u>	23
2.4.1	<u>Настройка просмотра сети</u>	25
2.4.2	<u>Конфигурирование системы DP-мастера в Network Configuration</u>	26
2.4.3	<u>Конфигурирование соединений</u>	27
2.4.4	<u>Сетевые переходы</u>	32
2.4.5	<u>Загрузка данных соединения</u>	33
2.5	<u>Создание программы S7</u>	35
2.5.1	<u>Введение</u>	35
2.5.2	<u>Таблица символов</u>	36
2.5.3	<u>Редактор программ</u>	38
2.5.4	<u>Обновление или генерирование исходных файлов</u>	41
2.5.5	<u>Приоритет адреса</u>	42
2.5.6	<u>Ссылочные данные</u>	43
2.5.7	<u>Мультиязыковые комментарии и отображение текста</u>	45
2.6	<u>Онлайновый (интерактивный) режим</u>	48
2.6.1	<u>Подключение PLC</u>	48
2.6.2	<u>Защита программы пользователя</u>	49
2.6.3	<u>Информация CPU</u>	50
2.6.4	<u>Загрузка пользовательской программы в CPU</u>	51
2.6.5	<u>Обработка блоков</u>	52
2.7	<u>Тестирование программы</u>	55
2.7.1	<u>Диагностирование аппаратных средств</u>	55
2.7.2	<u>Определение причины перехода в состояние STOP</u>	56
2.7.3	<u>Наблюдение и модифицирование переменных</u>	56
2.7.4	<u>Принудительная установка переменных (функция Force)</u>	58
2.7.5	<u>Разблокировка периферийных выходов</u>	61
2.7.6	<u>Статус программы LAD/FBD</u>	62

2 Программное обеспечение STEP 7

2.1 Базовый пакет STEP 7

Эта глава содержит описание базового программного пакета STEP 7 версии 5.1 (V5.1). Если первая глава представила обзор свойств программируемого контроллера, то в данной главе рассказывается о том, как устанавливать эти свойства.

В базовый пакет входят языки программирования списка операторов (statement list, STL), контактного плана (ladder logic, LAD) и функционального плана или диаграммы функциональных блоков (function block diagram, FBD). Кроме базового пакета также доступны такие пакеты, как S7-SCL (structured control language – структурированный язык управления), S7-GRAPH (sequence planning – последовательное планирование) и S7-HiGraph (state-transition diagram – диаграммы состояний-переходов).

2.1.1 Инсталляция

STEP 7 V5 – это 32-битное приложение, требующее в качестве операционной системы Windows 95 с Service Pack 1 по крайней мере (версия 4.00.950a), Windows 98 или Windows NT с Service Pack 3 по крайней мере (версия 4.00.1381). Для работы с программным обеспечением STEP 7 под Windows 95/98 вам потребуется программирующее устройство (programming device, PG) или PC с процессором 80486 или выше с минимум 32 МБ RAM; рекомендуется процессор Pentium с 64 МБ. Для Windows NT вам потребуется процессор Pentium и минимум 32 МБ RAM; у вас должна быть авторизация администратора, чтобы установить STEP 7 под Windows NT.

Если вы работаете в STEP 7 над большими проектами, включающими, скажем, несколько станций автоматизации с более чем 100 модулями, вы должны использовать программирующее устройство или PC с современной мощностью обработки.

STEP 7 V5 занимает приблизительно от 200 до 300 МБ на жестком диске с использованием одного языка (например, английского) в зависимости от операционной и файловой систем компьютера. Также необходим файл подкачки (файл swap-out). Размер этого файла составляет примерно от 128 до 256 МБ минус сконфигурированный объем главной памяти.

Вы должны убедиться в том, что на диске, содержащем данные вашего проекта, достаточно памяти. Для некоторых операций, таких как копирование проекта, потребности в памяти могут увеличиться. Если для файла подкачки не хватает пространства, могут возникать такие ошибки, как сбой программы. Не рекомендуется хранить данные проекта на диске, содержащем файл подкачки Windows.

Для инсталляции используется находящаяся на первой дискете программа установки SETUP для Windows 95/98/NT. На устройстве программирования STEP 7 уже установлен производителем.

Кроме STEP 7 V5 компакт-диск содержит также программу авторизации (смотрите ниже), программы NCM для конфигурирования устройств CP и электронные руководства по STEP 7 с Acrobat Reader V3.01.

Для онлайн-соединения требуется MPI-интерфейс. Устройства программирования уже оснащены встроенным многоточечным интерфейсом, но PC должны быть дооборудованы модулем MPI. Если вы хотите использовать карты памяти PC, вам потребуется проммер (prommer).

STEP 7 V5 имеет многопользовательские возможности, заключающиеся в том, что проект, который хранится, скажем, на центральном сервере, может редактироваться одновременно с нескольких станций. Необходимые установки производятся в Панели управления Windows (Windows Control Panel) с помощью программы «SIMATIC Workstation» («Рабочая станция SIMATIC»). В появляющемся диалоговом окне вы можете определить рабочую станцию как однопользовательскую систему или многопользовательскую систему с используемыми протоколами.

2.1.2 Авторизация

Для работы со STEP 7 требуется авторизация (право пользования). Авторизация поставляется на дискете. При выполнении инсталляции STEP 7 вам будет предложено установить вашу авторизацию, если жесткий диск уже не содержит авторизацию. Авторизацию можно поставить позже.

Вы также можете перенести авторизацию на другое устройство путем обратного копирования на (исходную) авторизационную дискету и последующего переноса ее на новое устройство.

Если вы утратили вашу авторизацию по какой-либо причине, например, из-за дефекта жесткого диска, то вы можете воспользоваться экстренной лицензией, также имеющейся на авторизационной дискете. Данная лицензия ограничена по времени действия до получения новой авторизации.

2.1.3 SIMATIC-менеджер

Утилита SIMATIC-менеджер (SIMATIC Manager) является главным инструментом в STEP 7; его пиктограмму (иконку) вы найдете на рабочем столе Windows или в меню Пуск (Start).



SIMATIC Manager

SIMATIC-менеджер запускается двойным щелчком левой клавишей мыши на его иконке.

При первом запуске появляется окно Мастера проектов (Project Wizard). Он может быть использован для упрощенного создания новых проектов. Вы можете деактивировать его при помощи переключателя (флажка, check box) «Display Wizard on starting the SIMATIC Manager» («При запуске SIMATIC-менеджера Отобразить Мастера»), так как при необходимости его можно вызвать по команде меню File → New Project Wizard (Файл → Мастер новых проектов).

Программирование начинается с открытия или создания проекта (project). Предоставляемые примеры проектов являются хорошей основой для ознакомления.

Когда вы откроете пример проекта, выбрав пункт меню File → Open (Файл → Открыть), вы увидите разделенное окно проекта: в левой части расположена структура открытого проекта (иерархия объектов), а справа отображается выделенный объект (рисунок 2.1). Щелчок на квадратике со знаком «плюс» в левом окне приведет к отображению дополнительных уровней структуры; выбор объекта в левой половине окна отобразит его содержимое в правой половине окна.

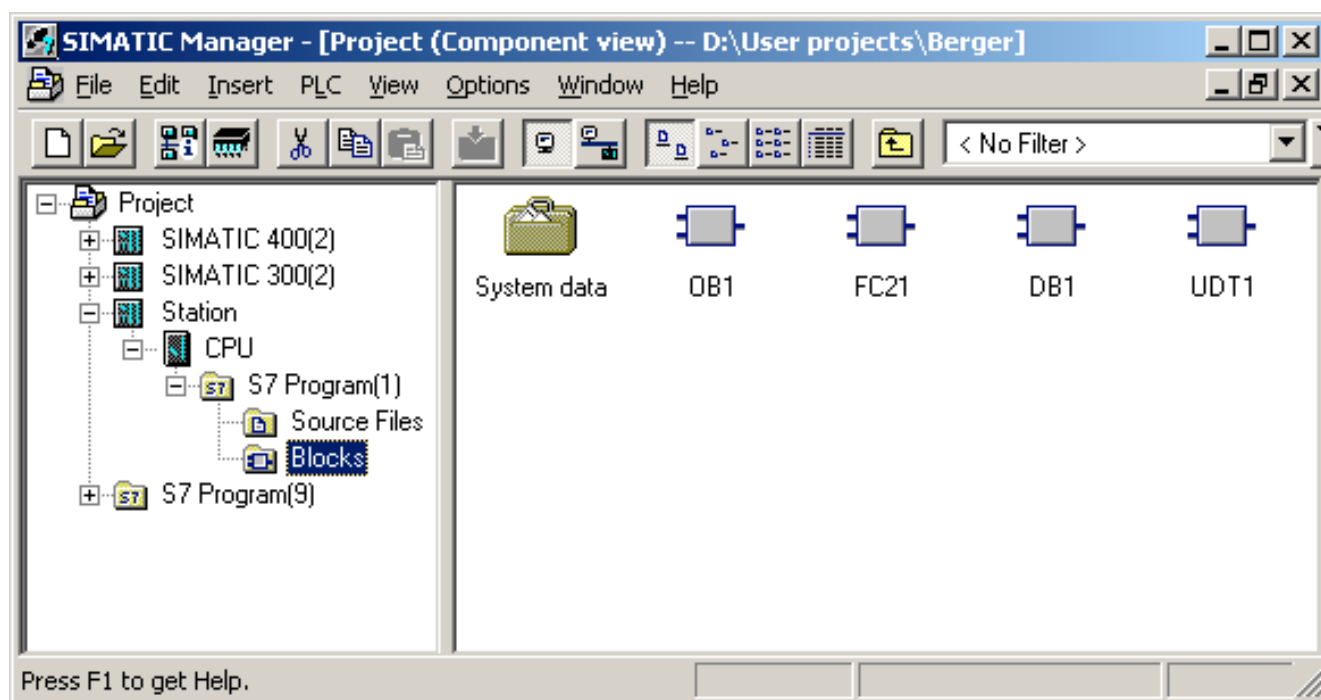


Рисунок 2.1 Пример окна SIMATIC-менеджера

Ваша работа с объектами мира STEP 7 происходит под управлением SIMATIC-менеджера. Данные «логические» объекты соответствуют «реальным» объектам вашего предприятия. Проект содержит предприятие в целом, станция соответствует программируемому контроллеру. Проект может содержать несколько станций, соединенных между собой, к примеру, через подсеть MPI. Станция содержит CPU, а CPU хранит программу, в вашем случае S7-программу. Эта программа, в свою очередь, является «контейнером» («container») для других объектов, таких как Blocks (Блоки), которые содержат, помимо других составляющих, скомпилированные блоки.

Объекты STEP 7 объединены посредством древовидной структуры. На рисунке 2.2 показана самая важная часть древовидной структуры (или «главная ветвь»), когда вы работаете с базовым пакетом STEP 7 для S7-приложений в режиме автономного просмотра. Выделенные (слева) объекты являются контейнерами для других объектов. Все объекты на рисунке доступны для автономного просмотра. Это те объекты, которые находятся на жестком диске устройства программирования. Если ваше программирующее устройство подключено к CPU (обычно целевая система PLC), вы можете переключиться в режим онлайн-просмотра выбором опции меню View → Online (Вид → Онлайн). Кроме того, эта опция отображает окно другого проекта, содержащее объекты удаленного устройства; объекты, выделенные на рисунке, более не отображаются.

Структура	Объект	Описание
Проект	Контейнер для всех объектов проекта	
MPI [PTP, PROFIBUS, Ethernet]	Подсеть	Содержит установки сетевых параметров для подсети
Станция SIMATIC 300/400	Контейнер для всех объектов станции	
Аппаратура	Конфигурационная таблица	Содержит конфигурационные данные для станции и параметры для модулей
CPU xxx	Контейнер для всех объектов CPU	
Соединения	Таблица соединений	Содержит определения коммуникационных соединений между узлами сети
Программа S7	Контейнер для всех объектов программы пользователя	
Символы	Таблица символов	Содержит назначения символов (= имена) абсолютным адресам глобальных данных
Исходники	Контейнер для всех объектов программных исходников	
Исходные файлы	Исходные программы	Содержит исходники для пользовательских программ (например, STL- и SCL-программы)
Блоки	Контейнер для всех скомпилированных объектов программы пользователя	
OB n	Организационные блоки	Содержат откомпилированный код и данные пользовательской программы
FB n	Функциональные блоки	
FC n	Функции	
DB n	Блоки данных	
SFC n	Системные функции	Содержат интерфейс вызовов для системных блоков, интегрированных в CPU
SFB n	Системные функцион. блоки	
Системные данные	Блоки системных данных	Содержат скомпилированные данные для конфигурационной таблицы
UDT n	Типы данных	Содержат определения пользовательских типов данных
VAT n	Таблицы переменных	Содержат переменные для наблюдения (мониторинга) и модифицирования
Программа S7	Контейнер для программы пользователя, не назначенной ни одному CPU (имеет ту же структуру, что и любая S7-программа, назначенная CPU)	

Рисунок 2.2 Иерархия объектов проекта STEP 7

В заголовке окна активного проекта вы можете увидеть, в каком режиме вы работаете – онлайнном или офлайнном (автономном). Для более ясного различия прямоугольник с заголовком и заголовок окна могут быть окрашены в другой цвет, нежели автономное окно. Для этой цели выберите Options → Customize (Опции → Настроить) и измените установки на вкладке «View» («Просмотр»).

Выберите Options → Customize (Опции → Настроить) для того, чтобы изменить основные установки SIMATIC-менеджера, такие как язык сессии (сеанса), программу архивирования и место хранения проектов и библиотек (libraries), а также, чтобы сконфигурировать программу архивирования.

Этапы редактирования

Следующее относится к общим понятиям редактирования объектов:

Выбрать (отметить) объект означает щелкнуть на нем один раз мышью, так чтобы он стал выделенным, подсвеченным (это возможно в обеих частях окна проекта).

Назвать объект означает щелкнуть на имени выбранного объекта (вокруг имени появится рамка, и вы сможете изменить имя в окне) или выбрать пункт меню Edit → Object Properties (Правка → Свойства объекта) и в диалоговом окне изменить имя. В случае некоторых объектов, таких как CPU, вы можете сменить имя только с помощью соответствующего инструмента (приложения), в данном случае – с помощью Hardware Configuration (Конфигурирование аппаратуры).

Чтобы *открыть объект*, дважды щелкните на этом объекте. Если объект является контейнером для других объектов, SIMATIC-менеджер отобразит содержимое объекта в правой половине окна. Если объект расположен на самом нижнем иерархическом уровне, SIMATIC-менеджер запускает соответствующий инструмент для редактирования объекта (например, двойной щелчок на блоке запускает редактор, позволяющий модифицировать блок).

В настоящей книге пункты меню в стандартной линейке меню в верхней части окна описываются как *последовательности действий оператора*. Программисты, имеющие опыт в использовании операторского интерфейса, используют иконки панели инструментов (toolbar). Очень эффективным является использование *правой клавиши мыши*. Единичный щелчок правой клавишей мыши на объекте отобразит меню с текущими опциями редактирования.

2.1.4 Проекты и библиотеки

В STEP 7 главные объекты (main objects) в верхней части иерархии объектов – это проекты и библиотеки.

Проекты (Projects) используются для систематизированного хранения данных и программ, требующихся для решения задачи автоматизации. По существу это

- конфигурационные данные аппаратного обеспечения,
- данные параметризации модулей,
- конфигурационные данные сетевых коммуникаций,
- программы (код и данные, символы, исходники).

Объекты в проекте упорядочены иерархически. Открытие проекта является первым шагом по редактированию всех (подчиненных) объектов, которые этот объект содержит. Следующие разделы посвящены тому, как редактировать эти объекты.

Библиотеки (Libraries) применяются для хранения повторно используемых программных компонентов. Библиотеки организованы иерархически. Они могут содержать программы STEP 7, которые, в свою очередь, могут содержать пользовательскую программу (контейнер для откомпилированных блоков), контейнер для исходных программ и таблицу символов (symbol table). За исключением онлайн-соединений (отладка невозможна) создание программы или раздела программы в библиотеке предоставляет ту же функциональность, что и в проекте.

Поставляемый пакет STEP 7 V5 предоставляет Стандартную Библиотеку (Standard Library), содержащую следующие программы:

- System Function Blocks (системные функциональные блоки)
Содержит интерфейсы вызовов системных блоков, встроенных в CPU, для автономного программирования;
- S5-S7 Converting Blocks (конвертирование блоков S5-S7)
Содержит загружаемые функции для конвертера S5/S7 (замена стандартных функциональных блоков S5 в связи с переходом на другую версию программы);
- TI-S7 Converting Blocks (конвертирование блоков TI-S7)
Содержит дополнительные загружаемые функции и функциональные блоки для конвертера TI-S7;
- IEC Function Blocks (функциональные блоки IEC)
Содержит загружаемые функции для редактирования переменных сложных типов данных DATE_AND_TIME и STRING;
- Communication Blocks (коммуникационные блоки)
Содержит загружаемые функции для управления CP 342-5 (CP-модулями);
- PID Control Blocks (блоки PID-управления)
Содержит загружаемые функциональные блоки для замкнутого управления;
- Organization Blocks (организационные блоки)
Содержит шаблоны (templates) организационных блоков (по существу описание переменных для стартовой информации).

Обзор содержимого этих библиотек вы можете найти в главе 25 «Библиотеки блоков». Например, вы намерены использовать модуль S7 со стандартными блоками, соответствующая инсталляционная программа устанавливает на жесткий диск стандартные блоки как библиотеку. Затем вы можете скопировать эти блоки из библиотеки в ваш проект. Библиотека открывается по команде File → Open (Файл → Открыть) и может редактироваться также как проект. Также вы можете создавать ваши собственные библиотеки.

Пункт меню File → New (Файл → Новый) генерирует новый объект на вершине иерархии объектов (проект, библиотеку). Месторасположение в структуре директории, где SIMATIC-менеджер должен создать проект или библиотеку, должно быть определено в пункте меню Options → Customize (Опции → Настроить) или в диалоговом окне «New» («Новый»).

Меню Insert (Вставка) используется для добавления новых объектов к существующим (например, добавление нового блока к программе). Однако, перед выполнением этих действий вы сначала должны в левой части окна SIMATIC-менеджера выбрать контейнер объектов, в который вы хотите вставить новый объект.

Копирование контейнеров объектов и самих объектов выполняется с помощью опций меню Edit → Copy (Правка → Копировать) и Edit → Paste (Правка → Вставить) или, как обычно в Windows, перетаскиванием выбранных объектов мышью из одного окна в другое. Обратите внимание на то, что в SIMATIC-менеджере нельзя отменить (undo) удаление объекта или контейнера объектов.

2.1.5 Онлайн-помощь

Онлайн-помощь (интерактивная) предоставляет информацию, необходимую вам во время вашего сеанса программирования, не прибегая к обращению к руководствам. Вы можете выбрать тему (topic), по которой требуются данные, в пункте меню Help (Помощь). К примеру, опция онлайн-помощи GETTING STARTED (Начало) предоставляет краткое резюме о том, как работать в SIMATIC-менеджере.

Help → Contents (Помощь → Содержание) запускает центральную функцию Помощи STEP 7 из любого приложения. Здесь содержится вся основная информация.

Help → Context-Sensitive Help F1 (Помощь → Контекстно-зависимая помощь F1) предоставляет контекстно-зависимую помощь, то есть если вы нажмете F1, то получите информацию, касающуюся отмеченного мышью объекта или текущего сообщения об ошибке.

В линейке символов имеется кнопка со стрелкой и знаком вопроса. Если вы нажмете эту кнопку, то к указателю мыши добавится знак вопроса. Теперь с помощью этого информационного указателя мыши вы можете выбрать объект на экране, к примеру, символ или команда меню, щелкнув на нем, и получить соответствующую онлайн-подсказку.

2.2 Редактирование проектов

При настройке проекта вы создаете «контейнеры» для результирующих данных, затем вы генерируете данные и заполняете эти контейнеры. Обычно вы создаете проект, используя соответствующие аппаратные средства, конфигурируете аппаратуру или, по крайней мере, CPU и в ответ получаете контейнеры для пользовательской программы. Кроме того, вы можете поместить S7-программу непосредственно в контейнер проекта без применения какого-либо оборудования. Заметьте, что инициализация модулей (изменение адресов, установки CPU, конфигурирование соединений) возможно только с помощью инструментария Hardware Configuration.

Мы настоятельно рекомендуем, чтобы весь процесс редактирования проекта выполнялся с использованием SIMATIC-менеджера. Создание, копирование или удаление директорий или файлов, а также изменение имен (!) с помощью Windows Explorer (Проводника Windows) внутри структуры проекта могут вызвать проблемы при работе с SIMATIC-менеджером.

2.2.1 Создание проектов

Мастер проектов (Project wizard)

Начиная со STEP 7 V3.2, для оказания помощи в создании новых проектов применяется *STEP 7 Wizard (Мастер STEP 7)*. Вы определяете CPU, а Мастер создает за вас проект со станцией S7 и выбранным CPU, а также контейнер S7-программы, контейнер исходников и контейнер блоков с выбранными организационными блоками.

Создание проекта со станцией S7

Если вы хотите создать проект «вручную», то описание необходимых действий с вашей стороны вы найдете в этом разделе. Общая информация о командах оператора по редактированию объектов содержится в параграфе 2.1.3 «SIMATIC-менеджер».

Создание нового проекта

Выберите пункт меню **File** → **New** (**Файл** → **Новый**), в диалоговом окне введите имя, при необходимости измените тип и место хранения и подтвердите нажатием кнопки «ОК» или клавиши RETURN.

Вставка в проект новой станции

Выберите проект и вставьте станцию с помощью **Insert** → **Station** → **SIMATIC 300 Station** (**Вставка** → **Станция** → **Станция SIMATIC 300**) (в случае S7-300).

Конфигурирование станции

В левой части окна проектов щелкните мышью на квадратике с плюсом, следующим за проектом, и выберите станцию; SIMATIC-менеджер отобразит объект аппаратных средств в правой части окна. Двойной щелчок на *Hardware (Apparatur)* запустит инструмент Hardware Configuration, с помощью которого вы можете отредактировать конфигурационные таблицы (configuration table). Если каталог модулей не выведен на экран, вызовите его командой меню View → Catalog (Вид → Каталог).

Конфигурирование начинается с выбора направляющей рейки с помощью мыши, например, под «SIMATIC 300» и «RACK 300», «удерживая», перетащите ее на свободное пространство верхней части окна станции и «отпустите» (операция drag & drop). После этого вы увидите таблицу с представлением слотов на направляющей рейке.

Затем из каталога модулей выберите требуемые модули и, используя описанную выше процедуру, перетащите и поместите их в соответствующие слоты. Для дальнейшего редактирования структуры проекта для станции требуется, по крайней мере, один CPU, например, CPU 314 в слоте 2. Все остальные модули вы можете добавить позднее. Редактирование аппаратного обеспечения подробно обсуждается в параграфе 2.3 «Конфигурирование станций».

Сохраните и откомпилируйте станцию, затем закройте и вернитесь в SIMATIC-менеджер. В дополнение к конфигурации аппаратных средств теперь открытая станция показывает также CPU.

При конфигурировании CPU SIMATIC-менеджер также создает папку S7 Program (S7-программы) со всеми объектами. Создание структуры проекта теперь завершено.

Просмотр содержимого папки S7 Program

Откройте CPU; в правой части окна проекта вы увидите значки S7-программы и таблицы соединений (connection table) – «S7 Program» и «Connection».

Откройте папку S7-программы; SIMATIC-менеджер в правой половине окна отобразит значки откомпилированной пользовательской программы (Blocks – Блоки), контейнер исходных программ и таблицу символов.

Откройте пользовательскую программу (Blocks); SIMATIC-менеджер в правой половине окна покажет значки откомпилированных конфигурационных данных (System data – Системные данные) и пустой организационный блок для главной программы (OB 1).

Редактирование объектов пользовательской программы

На данный момент мы достигли нижнего уровня иерархии объектов. OB 1 открыт в первый раз, отображается окно со свойствами объектов, и открыт редактор, необходимый для редактирования программы в организационном блоке. Добавьте другой

пустой блок для пошагового редактирования командой меню Insert → S7 Block → ... (Вставка → Блок S7) (*Blocks* – *Блоки* – должен быть подсвечен) и выберите из предоставляемого списка нужный типа блока.

Объект *System data* (*Системные данные*) в раскрытом виде показывает список доступных системных блоков данных. Вы получаете откомпилированные конфигурационные данные. Эти системные блоки данных редактируются с помощью объекта *Hardware* (*Аппаратура*) в контейнере *Station* (*Станция*). Вы можете переместить *System data* (*Системные данные*) в CPU, выбрав опцию меню PLC → Download (PLC → Загрузить), и параметризовать таким образом CPU.

Контейнер объектов *Source Files* (*Исходные файлы*) пуст. Предварительно отметив контейнер *Source Files* (*Исходные файлы*), выберите команду Insert → S7 Software → STL Source File (Вставка → Программное обеспечение S7 → Исходные файлы STL), чтобы вставить пустой исходный текстовый файл. Или вы можете выбрать Insert → External Source File (Вставка → Внешний исходный файл), чтобы переместить исходный текстовый файл, созданный, скажем, в другом редакторе в формате ASCII, в контейнер *Source Files* (*Исходные файлы*).

Создание проекта без станции S7

При желании вы можете сформировать программу без необходимости сначала настроить станцию. Для этого сами сгенерируйте контейнер для вашей программы. Выберите проект и сгенерируйте папку S7-программы путем Insert → Program → S7 Program (Вставка → Программа → S7-программа). Под этой папкой S7-программы SIMATIC-менеджер создаст контейнеры объектов *Sources* (*Исходники*) и *Blocks* (*Блоки*). *Blocks* (*Блоки*) содержит пустой OB 1.

Создание библиотеки

Программу вы также можете создать в виде библиотеки, если вы хотите, к примеру, использовать ее более одного раза. И таким образом, стандартная программа будет всегда доступна, ее можно скопировать полностью или частично в текущую программу.

Заметьте, что в библиотеке вы не можете устанавливать онлайнное соединение, а это значит, что вы можете отлаживать программу STEP 7 только в рамках проекта.

2.2.2 Управление, перегруппировка, архивирование

Утилита SIMATIC-менеджер осуществляет поддержку списка всех известных «главных объектов»: пользовательских проектов, библиотек и примеров проектов. Вы устанавливаете примеры проектов вместе со STEP 7, а пользовательские проекты и ваши собственные библиотеки вы устанавливаете самостоятельно.

При выполнении команды меню File → Rearrange (Файл → Перегруппировка) SIMATIC-менеджер покажет вам список имен и путей всех известных проектов и библиотек. Вы можете удалить из списка проекты или библиотеки, которые вы не хотите отображать («скрыть»), или включить в список новые проекты и библиотеки («показать»).

Когда исполняется File → Rearrange (Файл → Перегруппировка), SIMATIC-менеджер устраняет пробелы, образовавшиеся от удалений, и оптимизирует память данных подобно тому, как программа дефрагментации оптимизирует память данных на жестком диске. Реорганизация может занять некоторое время в зависимости от выполняемого перемещения данных.

Также вы можете заархивировать проект или библиотеку (File → Archive, Файл → Архивировать). В этом случае SIMATIC-менеджер сохраняет выбранный объект (директорию проекта или библиотеки со всеми поддиректориями и файлами) в сжатом виде в архивном файле.

Для архивирования проекта или библиотеки вам нужна программа архивации. STEP 7 содержит архиваторы ARJ и PKZIP 2.50, но вы можете использовать также другие программы-архиваторы (WinZip с версии 6.0, Pkzip с версии 2.04g, JAR с версии 1.02 или LHARC с версии 2.13).

Проекты и библиотеки не могут редактироваться в архивном (сжатом) состоянии. Вы можете распаковать заархивированный объект путем File → Retrieve (Файл → Восстановить) и затем продолжить редактирование. Восстановленные объекты принимаются системой управления проектом или библиотекой автоматически.

Установки для архивирования и восстановления вы можете сделать во вкладке «Archive» («Архив»), выбрав предварительно команду Options → Customize (Опции → Настроить); например, имеется установка целевой директории для архивирования и восстановления или «Generate archive path automatically» («Генерировать путь архива автоматически») (тогда при архивировании не требуется дополнительных определений, потому что имя архивного файла генерируется из имени проекта).

Архивирование проекта в CPU

STEP 7, начиная с версии 5.1, с соответствующим образом построенными CPU S7-400 позволяет вам хранить проект в архивном (сжатом) виде в загрузочной памяти CPU, то есть на карте памяти. Таким образом, вы можете сохранить все данные проекта, требуемые для полного выполнения пользовательской программы, такие как символы или исходные файлы, непосредственно в объекте-машине или объекте-предприятии. При необходимости изменить или дополнить программу вы можете загрузить хранимые локально данные на жесткий диск, скорректировать пользовательскую программу и снова сохранить в CPU обновленные проектные данные.

При загрузке данных проекта на карту памяти, подключенную к CPU, откройте проект, отметьте CPU и выберите PLC → Save Project on Memory Card (PLC → Записать объект на карту памяти). В обратном направлении: переместите сохраненные данные обратно на жесткий диск с помощью PLC → Retrieve Project from Memory Card (PLC

→ Получить объект из карты памяти). Обратите внимание на то, что когда вы записываете на карту памяти, подключенную к CPU, все содержимое загрузочной памяти записывается в CPU, включая системные данные и программы пользователя.

Если вы хотите осуществить обратную выборку проектных данных, сохраненных в CPU, без создания проекта на жестком диске, выберите соответствующий CPU с помощью опции меню PLC → Display Accessible Nodes (PLC → Отобразить доступные узлы). Если карта памяти вставлена в разъем модуля программирующего устройства, то перед передачей выберите карту памяти при помощи команды File → S7 Memory Card → Open (Файл → Карта памяти S7 → Открыть).

2.2.3 Версии проектов

С момента выхода STEP 7 V5 различных версий SIMATIC-проектов стало три. STEP 7 V1 создает проекты версии 1, STEP 7 V2 создает проекты версии 2, и STEP 7 V3/V4/V5.0 может быть использован для создания и редактирования проектов версий 2 и 3. С помощью STEP 7 V5.1 вы можете сформировать и отредактировать V3-проекты и V3-библиотеки.

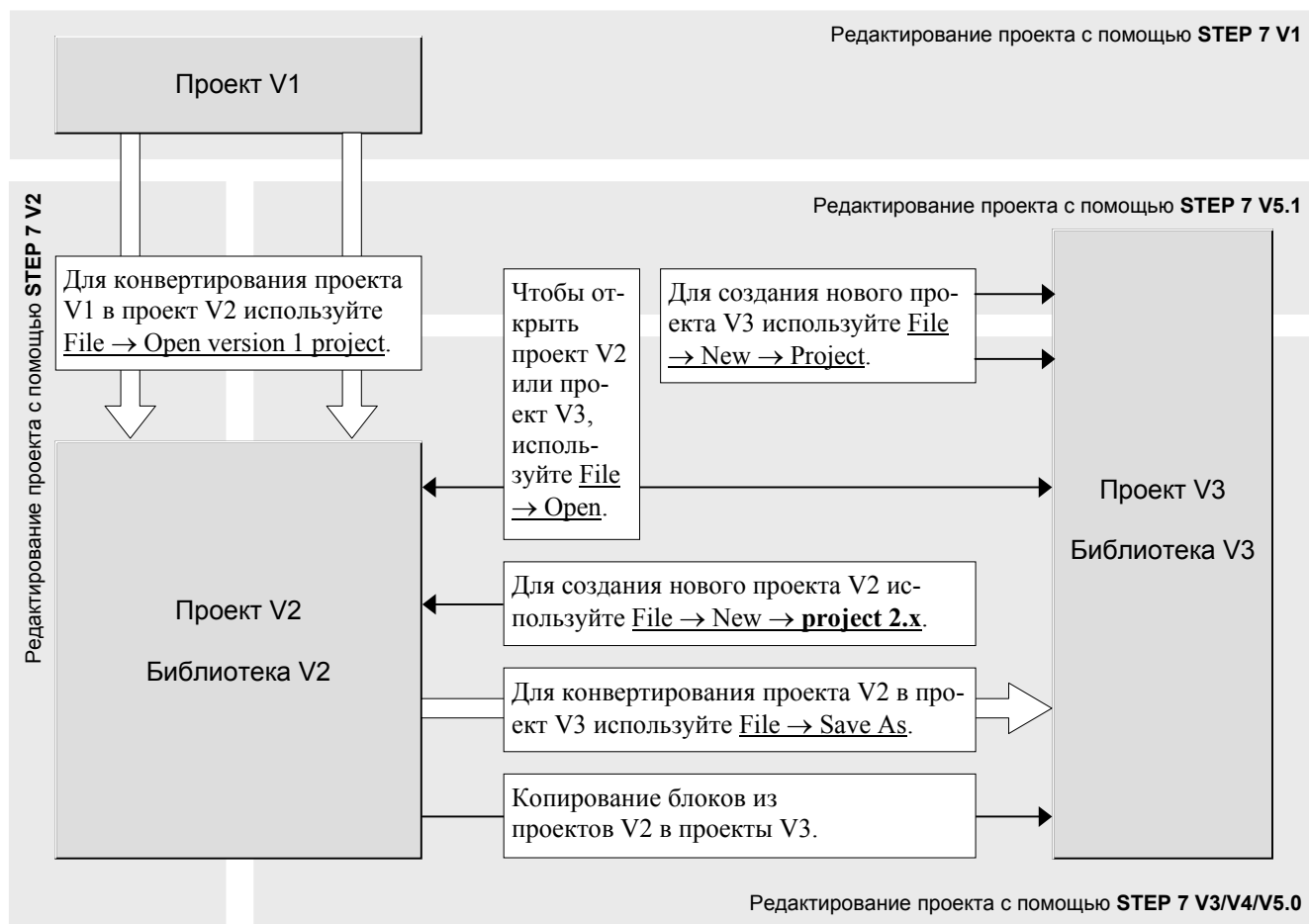


Рисунок 2.3 Редактирование проектов различных версий

Если у вас есть проект версии 1, вы можете преобразовать его в проект версии 2 с помощью File → Open Version 1 Project (Файл → Открыть проект версии 1). Структура проекта с программами, откомпилированными блоками версии 1, исходными STL-программами, таблицей символов и конфигурацией аппаратных средств остаются неизменными.

Вы можете создавать и редактировать проекты версии 2 при помощи STEP 7 версий V2, V3, V4 и V5.0 (рисунок 2.3).

STEP 7 V5.1 работает только с проектами версии 3. Однако, вы можете конвертировать V1-проект в V2-проект с помощью File → Open Version 1 Project (Файл → Открыть проект версии 1), а также открыть V2-проект, выбрав File → Open (Файл → Открыть). Создать V2-проект или записать проект в формате версии 2 невозможно.

2.3 Конфигурирование станций

Для создания конфигурации программируемого контроллера используется инструмент Hardware Configuration. Конфигурирование выполняется автономно без подключения к CPU. Также вы можете использовать этот инструмент для адресации и параметризации модулей. Вы можете создать конфигурацию аппаратного обеспечения на стадии планирования или подождать до установки аппаратуры.

Конфигурирование аппаратного обеспечения начинается с выбора станции, затем следует обратиться к меню Edit → Open Object (Правка → Открыть объект) или дважды щелкнуть мышью на объекте Hardware в открытом контейнере SIMATIC 300/400 Station (станция SIMATIC 300/400). Основные установки производятся с помощью Options → Customize (Опции → Настроить).

Когда конфигурирование окончено, Station → Consistency Check (Станция → Проверка согласованности) проверит введенную вами информацию на наличие ошибок. Команда Station → Save (Станция → Сохранить) записывает на жесткий диск конфигурационные таблицы со всеми данными значений параметров в вашем проекте.

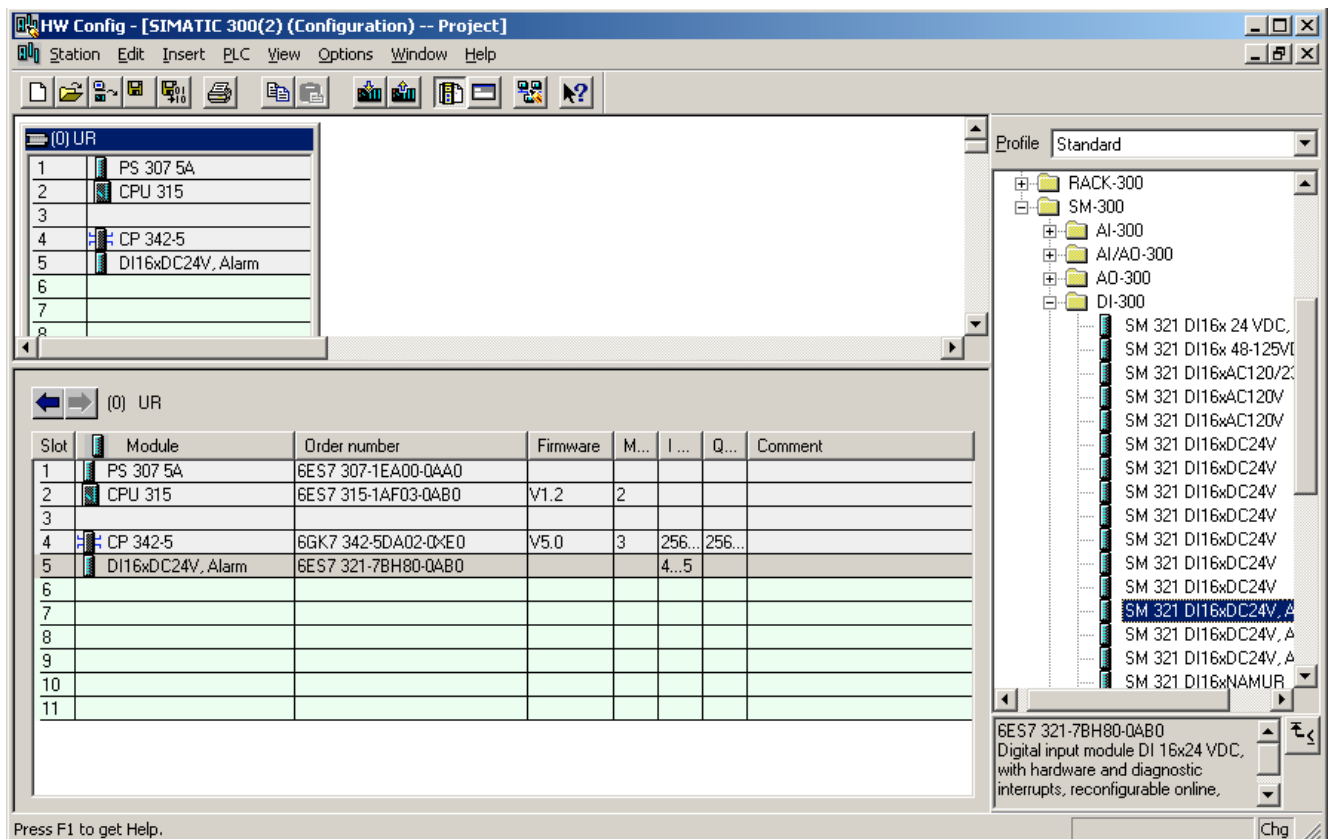


Рисунок 2.4 Пример окна станции в Hardware Configuration

Команда Station → Save and Compile (Станция → Сохранить и скомпилировать) не только сохраняет, но и компилирует конфигурационные таблицы и записывает скомпилированные данные в объект *System data* (*Системные данные*) в автономный контейнер *Blocks* (*Блоки*). После компиляции вы можете переместить конфигурационные данные в CPU с помощью PLC → Download (PLC → Загрузить). Объект *System data* (*Системные данные*) в онлайн-контейнере *Blocks* (*Блоки*) представляет текущие конфигурационные данные в CPU. Вы можете «вернуть» эти данные на жесткий диск с помощью PLC → Upload (PLC → Выгрузить).

Экспортирование данных конфигурации аппаратного обеспечения осуществляется пунктом меню Station → Export (Станция → Экспорт). STEP 7 затем создает файл в формате ASCII, содержащий конфигурационные данные и данные параметризации модулей. Вы можете выбрать между текстовым форматом с данными в виде «читаемых» английских букв и компактным форматом с шестнадцатеричными данными. Предусмотрена возможность импорта соответствующим образом структурированного ASCII-файла.

Контрольная сумма

Hardware Configuration генерирует контрольную сумму (checksum) корректно скомпилированной станции и сохраняет ее в системных данных. Идентичные системные конфигурации имеют такую же контрольную сумму, так что вы можете, к примеру, легко сравнить онлайн-ую и автономную конфигурации.

Контрольная сумма является свойством объекта *System data* (*Системные данные*). Чтобы считать контрольную сумму, откройте в S7-программе контейнер *Blocks* (*Блоки*), выберите объект *System data* (*Системные данные*) и откройте его с помощью Edit → Open Object (Правка → Открыть объект). Пользовательская программа также имеет соответствующую контрольную сумму. Она находится вместе с контрольной суммой системных данных в свойствах *Blocks* (*Блоки*): выберите контейнер *Blocks* (*Блоки*) и затем Edit → Object Properties (Правка → Свойства объектов) на вкладке «Checksums» («Контрольные суммы»).

Окно станции

При работе Hardware Configuration отображает окно станции и каталог аппаратного обеспечения (рисунок 2.4). Увеличьте или максимизируйте окно станции для облегчения редактирования. В верхнем разделе окна показаны монтажные стойки в форме таблиц и DP-станции в виде символов. Если имеются несколько стоек, то вы видите здесь соединения между интерфейсными модулями, и, если используется PROFIBUS, вы видите конфигурацию системы DP-мастера. Нижняя часть окна станции отображает конфигурационную таблицу, в подробностях показывающую стойку или DP-ведомого, отмеченных в верхнем разделе.

Каталог аппаратного обеспечения

Каталог оборудования вы можете вызывать и закрывать с помощью View → Catalog (Вид → Каталог). Он содержит все монтажные стойки, модули и интерфейсные подмодули, известные STEP 7. Выбрав Options → Edit Catalog Profile (Опции → Редактировать профиль каталога), вы можете откомпилировать ваш собственный каталог аппаратных средств, который показывает только те модули, с которыми вы хотите работать, - в выбранной вами структуре. Двойным щелчком клавишей мыши на линейке заголовка вы можете «пристыковать» каталог оборудования к правой границе окна станции или снова его отпустить.

Конфигурационная таблица

Инструмент Hardware Configuration работает с таблицами, каждая из которых представляет монтажную стойку, модуль или DP-станцию. Конфигурационная таблица показывает слоты с расположенными в них модулями или свойства модулей, такие как их адреса и порядковые номера. Двойной щелчок на строке модуля открывает окно свойств модуля и позволяет задать параметры модуля.

2.3.1 Компоновка модулей

Конфигурирование начинается с того, что вы выбираете направляющую рейку из каталога модулей, например, под «SIMATIC 300» или «RACK 300», «удерживая» ее, перетаскиваете мышью в верхнюю часть окна станции и помещаете где-нибудь в этом окне (drag&drop). Отображается пустая конфигурационная таблица для центральной стойки. Затем из каталога модулей выберите требуемые модули и описанным выше способом поместите их в соответствующие слоты. Символ «No Parking» («Парковки нет») информирует вас о невозможности поместить выбранный модуль в намеченный слот.

В случае однорядной конфигурации станции S7-300 слот 3 остается пустым; он зарезервирован для интерфейсного модуля стойки расширения.

Вы можете сгенерировать конфигурационную таблицу для другой стойки путем перетаскивания выбранной стойки из каталога и помещения ее в окно станции. В системах S7-400 неподключенной стойке назначается интерфейс (или точнее: соответствующие стойки получают интерфейсный модуль) на вкладке «Link» («Связь») окна «Properties» («Свойства») модуля Send IM (выберите модуль и нажмите Edit → Object Properties, Правка → Свойства объекта).

Компоновка распределенных I/O-станций описывается в параграфе 20.4.2 «Конфигурирование распределенных входов/выходов».

2.3.2 Адресация модулей

При компоновке модулей инструмент Hardware Configuration автоматически назначает модулям стартовый (начальный) адрес. Вы можете увидеть этот адрес в нижней

части окна станции в свойствах объектов соответствующих модулей. В случае CPU S7-400 и CPU S7-300 со встроенным DP-интерфейсом вы можете изменять адреса модулей. Выполняя это, соблюдайте правила адресации для систем S7-300 и S7-400, а также адресную емкость отдельных модулей.

Существуют модули, имеющие и входы и выходы, для которых вы можете (теоретически) зарезервировать различные стартовые адреса. Тем не менее, внимательно изучайте специальную информацию в руководствах по продукту; для большинства функциональных и коммуникационных модулей требуются одинаковые адреса для входов и выходов.

Когда вы назначаете стартовый адрес модулей в S7-400, вы также можете инициализировать вспомогательный (subsidiary) образ процесса (каждому модулю назначить частичный образ процесса – Part process image). Если в центральной стойке расположено более одного CPU, то автоматически устанавливается мультипроцессорный режим, и вы должны назначить модуль центральному процессору.

С помощью View → Address Overview (Вид → Обзор адресов) вызывается окно, содержащее все адреса модулей, которые используются в настоящий момент выбранным CPU.

Модули на MPI-шине или коммуникационной шине имеют MPI-адрес. Вы также можете изменить эти адреса. Однако, имейте в виду, что новый MPI-адрес становится действующим после передачи в CPU конфигурационных данных.

Символы для адресов пользовательских данных

В Hardware Configuration вы можете назначить входам и выходам символы (имена), которые передаются в таблицу символов.

После того, как вы скомпоновали и адресовали цифровые и аналоговые модули, сохраните данные станции. Затем выберите модуль (строку) и нажмите Edit → Symbols (Правка → Символы). В окне, которое затем откроется, вы можете назначить символ, тип данных и комментарий абсолютному адресу для каждого канала (побитно для цифровых модулей и каждому слову для аналоговых модулей).

Кнопка «Add Symbol» («Добавить символ») вводит абсолютные адреса как символы вместо абсолютных адресов без символов. Кнопка «Apply» («Применить») передает символы в таблицу символов. «ОК» также закрывает диалоговое окно.

2.3.3 Параметризация модулей

При параметризации модулей вы определяете их свойства. Параметризовать модули необходимо только тогда, когда вы хотите изменить параметры по умолчанию. Обязательным условием параметризации является расположение модуля.

Дважды щелкните в конфигурационной таблице на модуле или выделите модуль и нажмите Edit → Object Properties (Правка → Свойства объекта). В диалоговом окне

откроются несколько таблиц с задаваемыми параметрами для этого модуля. Когда вы используете этот метод для параметризации CPU, вы определяете характеристики исполнения вашей (пользовательской) программы.

Некоторые модули позволяют вам устанавливать их параметры во время исполнения через пользовательскую программу с помощью системных функций SFC 55 WR_PARM, SFC 56 WR_DPARM и SFC 57 PARM_MOD.

2.3.4 Объединение модулей в сеть с помощью MPI

Для MPI (подсеть) узлы определяются дополнительно с помощью свойств модуля (Module Properties). В конфигурационной таблице выберите CPU или карту MPI-интерфейса, если CPU оснащен таковой, и откройте выбранный объект, нажав Edit → Object Properties (Правка → Свойства объекта). В появляющемся затем диалоговом окне в секции «Interface» («Интерфейс») вкладки «General» («Общие») есть кнопка «Properties» («Свойства»). Если вы нажмете на эту кнопку, вы переместитесь в другое диалоговое окно с вкладкой «Parameter» («Параметр»), где сможете найти подходящую подсеть.

Здесь также есть возможность установить MPI-адрес, предусмотренный вами для данного CPU. Заметьте, что в более старших CPU S7-300 модули FM или CP с MPI-соединением автоматически получают MPI-адрес от CPU.

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

Совет: если у вас есть несколько станций с одинаковым типом CPU, присвойте CPU различных станций разные имена (идентификаторы). По умолчанию все они имеют имя «CPUxxx(1)», поэтому в подсети они могут различаться по их MPI-адресам. Если вы не хотите присваивать имена самостоятельно, вы можете, например, сменить идентификатор по умолчанию «CPUxxx(1)» на «CPUxxx(n)», где «n» - MPI-адрес.

При назначении MPI-адреса также примите во внимание возможность более позднего подключения к MPI-сети устройства программирования или панели оператора (operator panel, OP) для осуществления поддержки или обслуживания. Установленные на постоянной основе программирующие устройства или OP должны быть подключены к сети MPI напрямую; для подключаемых через шнур (отвод) устройств имеется MPI-коннектор с сокетом (гнездом) с большим сечением и резьбовым соединением. Совет: зарезервируйте адрес 0 для сервисного программирующего устройства, адрес 1 – для сервисной OP и адрес 2 – для резервного CPU (соответствует адресам по умолчанию).

2.3.5 Наблюдение и модифицирование модулей

При помощи Hardware Configuration вы можете провести проверку монтажных соединений (разводки) машины или предприятия без пользовательской программы. Для этого требуется, чтобы программирующее устройство было подсоединено к станции (онлайн) и конфигурация сохранена, откомпилирована и загружена в CPU. Теперь вы можете адресовать любой каждый цифровой и аналоговый модуль. Выбе-

рите модуль, затем PLC → Monitor/Modify (PLC → Наблюдение/модифицирование) и установите операционные режимы наблюдения (Monitor) и модифицирования (Modify) и условия вызова.

Hardware Configuration по нажатию кнопки «Status Value» («Значение статуса») показывает сигнальные состояния значений каналов модулей. Кнопка «Modify Value» («Изменение значения») предназначена для записи в модуль значения, определенного в столбце Modify Value.

Если окошко метки (флажок) «I/O Display» («Отображение I/O») активировано, то периферийные входы/выходы (память модуля) отображаются вместо входов / выходов (образ процесса). Флажок «Enable Perif Outputs» («Разблокировать периферийные выходы») отменяет блокировку модулей выходов, если CPU находится в режиме STOP (Останов) (обратитесь к параграфу 2.7.5 «Разблокировка периферийных выходов»).

Вы можете ознакомиться с другими методами наблюдения и модифицирования входов и выходов в параграфах 2.7.3 «Наблюдение и модифицирование переменных» и 2.7.4 «Принудительная установка переменных».

2.4 Конфигурирование сети

Сетевая система станций S7 является коммуникационной базой SIMATIC-системы. Требуемые объекты – это подсети и модули с коммуникационными возможностями в станциях. Вы можете с использованием SIMATIC-менеджера создать в рамках иерархии проекта новые подсети и станции. Затем с помощью инструмента Hardware Configuration могут быть добавлены модули с коммуникационными возможностями (устройства CPU и CP); одновременно настраиваемой сети назначаются коммуникационные интерфейсы этих модулей. После этого определяются коммуникационные отношения между этими модулями – соединения – с помощью инструмента Network Configuration (Конфигурирование сети) в таблице соединений.

Network Configuration позволяет графически представить сконфигурированные сети и их узлы. Также с помощью Network Configuration вы можете создать все необходимые подсети и станции; затем назначить станции подсетям и параметризовать узловые свойства модулей с коммуникационными возможностями.

Чтобы определить коммуникационные отношения посредством инструмента конфигурирования сети, вы можете действовать следующим образом:

- Откройте стандартную подсеть MPI в контейнере проекта; если она более не доступна, просто создайте новую подсеть с помощью Insert → Subnet (Вставка → Подсеть).
- Используйте Network Configuration для создания необходимых станций и – если требуется – других подсетей.
- Откройте станции и обеспечьте их модулями с коммуникационными возможностями.
- При необходимости измените параметры сети.
- Если требуется, определите коммуникационные соединения в таблице соединений.

Также в рамках Network Configuration можно сконфигурировать коммуникацию глобальных данных: выберите подсеть MPI и затем Options → Define Global Data (Опции → Определить глобальные данные) (обратитесь к параграфу 20.5 «Коммуникация глобальных данных»).

Network → Save (Сеть → Сохранить) сохраняет данные во время работы с Network Configuration. Вы можете проверить согласованность сетевых настроек с помощью Network → Consistency Check (Сеть → Проверка согласованности).

Network Configuration закрывается по нажатию Network → Save and Compile (Сеть → Сохранить и скомпилировать).

Окно Network (окно сети)

Перед запуском утилиты Network Configuration у вас должен быть создан проект. Вместе с проектом SIMATIC-менеджер автоматически создает подсеть MPI.

Двойной щелчок на этой или любой другой подсети запускает Network Configuration. Также можно запустить Network Configuration, если открыть объект Connections (Соединения) в контейнере CPU.

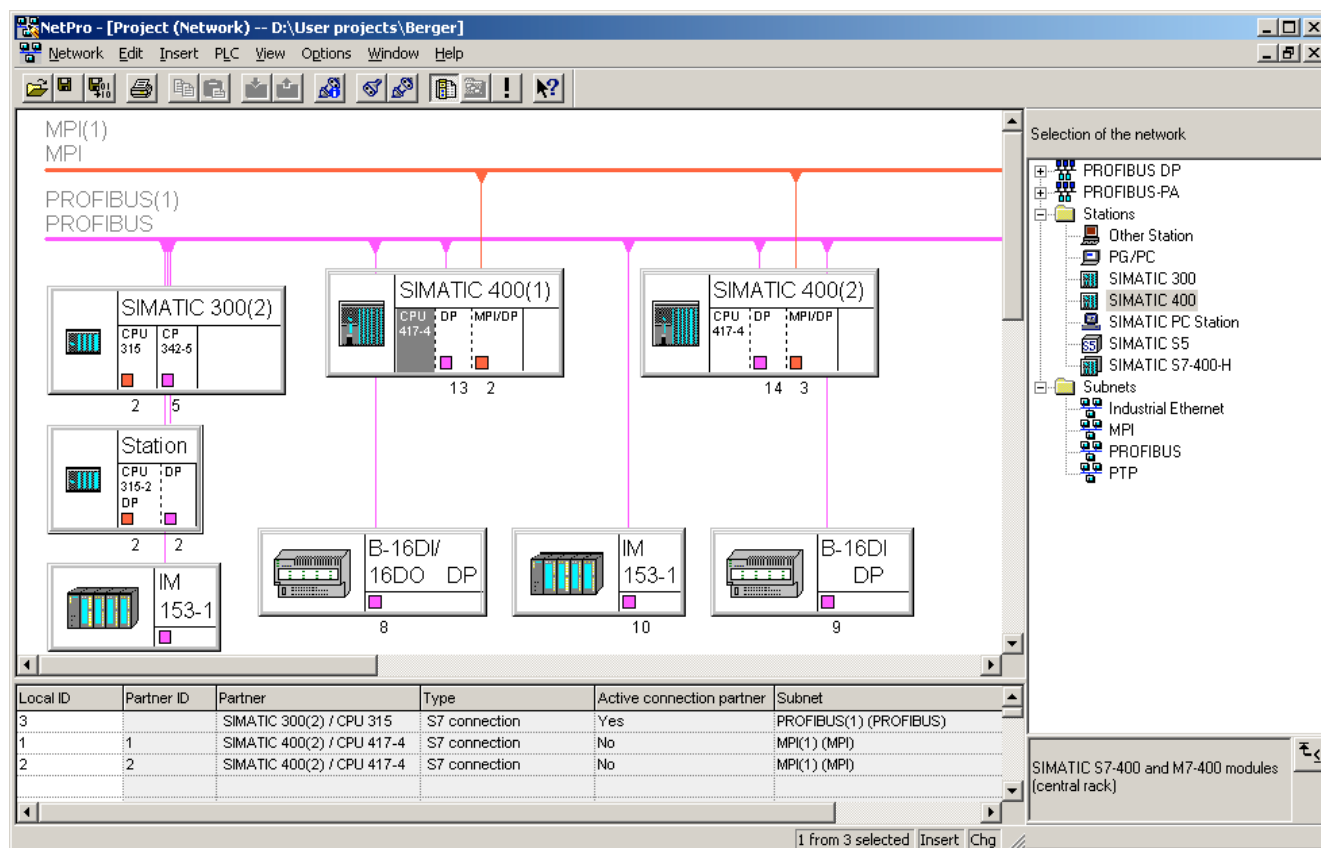


Рисунок 2.5 Пример Network Configuration

В верхней части окна Network Configuration показаны все ранее созданные в проекте подсети и станции (узлы) со сконфигурированными соединениями.

Если в верхней секции окна выбран модуль с «коммуникационными возможностями», например, CPU S7-400, то в нижней части окна отобразится таблица соединений.

Второе окно показывает каталог сетевых объектов (Network Object Catalog) с выбором доступных станций SIMATIC, подсетей и DP-станций. Вы можете скрыть каталог и вывести его на экран с помощью пункта меню View → Catalog (Вид → Каталог), а также «пристыковать» его к правому краю окна сети (двойным щелчком на линейке заголовка).

При помощи команд View → Zoom In (Вид → Увеличить масштаб), View → Zoom Out (Вид → Уменьшить масштаб) и View → Zoom Factor... (Вид → Выбрать масштаб...) вы можете настроить масштаб графического представления.

2.4.1 Настройка просмотра сети

Работа в Network Configuration начинается с выбора подсети. При помощи мыши вы должны выбрать из каталога подсеть, и, удерживая, перетащить ее в окно сети. Подсеть в окне представляется в виде горизонтальной линии. Недопустимые позиции индицируются запрещающим указателем мыши.

Идентичные действия выполняются по отношению к требуемым станциям, для начала без подключения к сети. Станции еще «пустые». Двойной щелчок на станции откроет инструментарий Network Configuration, позволяющий вам сконфигурировать станцию или, по крайней мере, модуль(и) с сетевым соединением. Сохраните станцию и вернитесь к Network Configuration.

Интерфейс модуля с коммуникационной возможностью изображен в Network Configuration в виде маленького квадрата под представлением модуля. Нажмите на этот квадрат, и, удерживая, перетащите его к соответствующей подсети. Соединение с подсетью представлено в виде вертикальной линии.

То же самое проделайте с другими узлами.

В окне сети вы можете передвигать созданные подсети и станции. Таким образом, вы также можете визуально представить вашу аппаратную конфигурацию.

Установка коммуникационных свойств

После формирования графического представления происходит параметризация подсетей: выберите подсеть и нажмите Edit → Object Properties (Правка → Свойства объекта). Окно свойств, которое затем появляется, во вкладке «General» («Общие») содержит ID (идентификатор) подсети S7. ID состоит из двух шестнадцатеричных чисел, номера проекта и номера подсети. Этот ID подсети S7 необходим, если вы хотите работать в онлайн-режиме с программирующим устройством без соответствующего проекта с целью достижения по сети других узлов. Сетевые свойства, например, скорость передачи данных или наибольший адрес узла, устанавливаются во вкладке «Network Settings» («Сетевые установки»).

При выборе сетевого соединения (подключения к сети) узла вы можете определить сетевые свойства узла с помощью Edit → Object Properties (Правка → Свойства объекта), например, адрес узла и сеть, к которой он подключен, или вы можете создать новую подсеть.

На вкладке «Interfaces» («Интерфейсы») свойств станции вы можете увидеть обзор всех модулей с коммуникационными возможностями, содержащий адреса узлов и используемые типы подсетей.

Определение свойств модулей узлов осуществляется тем же способом (теми же действиями оператора, что и в инструменте Hardware Configuration).

2.4.2 Конфигурирование системы DP-мастера в Network Configuration

Network Configuration также можно использовать для конфигурирования распределенных входов/выходов. Выберите опцию меню View → With DP Slaves (Вид → С DP-ведомыми), чтобы отобразить или скрыть DP-ведомых в окне просмотра сети.

Чтобы сконфигурировать систему DP-мастера, вам потребуется следующее:

- Подсеть PROFIBUS (если она еще не доступна, перетащите подсеть PROFIBUS из каталога сетевых объектов в окно сети).
- DP-мастер станции (если он еще не доступен, перетащите станцию из каталога сетевых объектов в окно сети, откройте станцию и выберите DP-мастер с помощью Hardware Configuration либо интегрированный в CPU, либо как автономный модуль).
- Соединение из DP-мастера к подсети PROFIBUS (либо выберите подсеть в Hardware Configuration, либо щелкните на сетевом соединении к DP-мастеру в Network Configuration, «удерживая», перетащите в сеть PROFIBUS).

В окне сети выберите DP-мастера, которому должен быть назначен ведомый. Найдите DP-ведомого в каталоге сетевых объектов под «PROFIBUS» и соответствующим подкаталогом, перетащите его в окно сети и заполните появившееся окно свойств.

Параметризация DP-ведомого осуществляется путем выбора этого объекта и последующего нажатия Edit → Open Object (Правка → Открыть объект). Запускается Hardware Configuration. Теперь вы можете установить адреса пользовательских данных или, в случае модульных ведомых, выбрать модули входов/выходов (обратитесь к параграфу 2.3 «Конфигурирование станций»).

Если DP-ведомый с развитой логикой (I-slave) был вами предварительно создан, то можете только подключить его к подсети (смотрите параграф 20.4.2 «Конфигурирование распределенных входов/выходов»). Тип интеллектуального DP-ведомого вы можете найти в каталоге сетевых объектов в разделе «Already created stations» («Уже созданные станции»); перетащите его, предварительно отметив DP-мастер, в окно сети и заполните появляющееся окно свойств (как в инструменте Hardware Configuration).

С помощью View → Highlight → Master System (Вид → Подсветить → Система мастера) можно выделить назначение узлов системы DP-мастера; сначала выделите мастера или ведомого этой системы мастера.

2.4.3 Конфигурирование соединений

Соединения описывают коммуникационные отношения между двумя устройствами. Соединения должны быть сконфигурированы, если

- вы хотите установить SFB-коммуникации между двумя устройствами SIMATIC S7 («Коммуникация посредством сконфигурированных соединений») или
- коммуникационный партнер не является устройством SIMATIC S7.

Примечание: сконфигурированное соединение для непосредственного онлайн-подключения программирующего устройства к сети MPI для программирования и отладки не требуется. Если вы хотите получить доступ к другим узлам, которые объединены в другую подключенную сеть, с помощью программирующего устройства, то вы должны сконфигурировать соединение программирующего устройства. Для этого в каталоге сетевых объектов выберите двойным щелчком объект *PG/PC* (устройства программирования) под *Stations (Станции)*, в окне сети откройте двойным щелчком *PG/PC* и выберите интерфейс и назначьте его подсети.

Таблица соединений

Коммуникационные соединения конфигурируются в таблице соединений. Требование: у вас есть созданный проект со всеми станциями, между которыми предполагается обмен данными, и вы назначили модули с коммуникационными возможностями сети.

Объект *Connections (Соединения)* в контейнере *CPU* представляет таблицу соединений. Двойной щелчок на *Connections (Соединения)* запускает Network Configuration, также как двойное нажатие на подсети в контейнере проекта.

Чтобы сконфигурировать соединения, выберите в Network Configuration CPU S7-400. В нижней части окна сети размещена таблица соединений (таблица 2.1; если она не видна, поместите указатель мыши на нижнюю границу окна до смены его формы и перетащите границу окна вверх). Ввод нового коммуникационного соединения осуществляется путем Insert → New Connection (Вставка → Новое соединение) или двойным щелчком на пустой строке.

Соединение создается для каждого «активного» CPU. Заметьте, что нельзя создать таблицу соединений для CPU S7-300; CPU S7-300 могут быть только «пассивными» партнерами в S7-соединении.

Выбор коммуникационного партнера осуществляется в диалоговых окнах «Station» («Станция») и «Module» («Модуль») окна «New Connection» («Новое соединение») (рисунок 2.6); станция и модуль должны к этому моменту существовать. В данном окне также определяется тип соединения.

При желании установить другие свойства соединения активируйте флажок «Show Properties Dialog» («Показать диалог свойств»).

Таблица соединений содержит все данные о сконфигурированных соединениях. Для отображения ее в требуемом виде используйте команду View → Display Columns (Вид → Отобразить столбцы) и выберите интересующую вас информацию.

ID (идентификатор) соединений

Количество возможных соединений определяется CPU. STEP 7 определяет ID соединения для каждого отдельного соединения и каждого партнера. Данная спецификация требуется вам при использовании в программе коммуникационных блоков. Вы можете изменить **локальный ID (local ID, ID соединения открытого в настоящий момент модуля)**. Это необходимо, если вы уже запрограммировали коммуникационные блоки и хотите использовать для соединения определенный в них локальный ID.

Новый локальный ID вводится в виде шестнадцатеричного числа. Оно не должно быть уже присвоено и должно оставаться в следующих границах значений, зависящих от типа соединений:

- Область значений для S7-соединений:
от 0001_{hex} до 0FFF_{hex}
- Область значений для PtP-соединений:
от 1000_{hex} до 1400_{hex}

ID партнера (partner ID) изменяется путем обращения к таблице соединений CPU-партнера и изменения (в этих условиях) локального ID: выберите линию соединения и нажмите Edit → Object Properties (Правка → Свойства объекта). Если STEP 7 не вводит ID партнера, то данное соединение является односторонним (смотрите ниже).

Таблица 2.1 Пример таблицы соединений

Local ID / Локаль- ный ID	Partner ID / ID парт- нера	Partner / Партнер	Type / Тип	Active connection partner / Актив- ный партнер со- единения	Send operating mode messages / Посылает сообщения рабочего режима
1	1	Станция 416 / CPU416(5)	S7- соединение	да	нет
2	2	Станция 416 / CPU416(5)	S7- соединение	да	нет
3		Станция 315 / CPU315(7)	S7- соединение	да	нет
4	1	Станция 417 / CPU414(4)	S7- соединение	да	нет

Партнер

Столбец отображает коммуникационного партнера. Если вы хотите зарезервировать соединительный ресурс без наименования устройства-партнера, в диалоговом окне под станцией (Station) введите «unspecified» («неопределено»).

В одностороннем соединении (one-way connection) коммуникация может быть инициирована одним партнером; пример: SFB-коммуникации между CPU S7-400 и S7-300. Хотя функции SFB-коммуникаций не доступны в S7-300, обмен данными может быть осуществлен CPU S7-400 при помощи SFB 14 GET и SFB 15 PUT. В S7-300 для этой коммуникации пользовательская программа не исполняется, а поддержка обмена данными осуществляется операционной системой.

Одностороннее соединение конфигурируется в таблице соединений «активного» CPU. Только после этого STEP 7 назначает «локальный ID». Также загрузка этого соединения происходит только в локальную станцию.

В случае **двухстороннего соединения (two-way connection)** оба партнера активно участвуют в коммуникации; например, два CPU S7-400 с помощью коммуникационных функций SFB 8 SEND и SFB 9 BRCV.

Для одного из двух партнеров конфигурирование двухстороннего соединения происходит только один раз. Затем STEP 7 назначает «локальный ID» и «ID партнера» и генерирует данные соединения для обеих станций. Загрузка каждого партнера должна осуществляться с помощью его собственной таблицы соединений.

Тип соединений

Базовый пакет STEP 7 предоставляет вам в Network Configuration следующие типы соединений:

PtP-соединение (двухточечное соединение) принято для использования в подсети RTP (процедуры 3964(R) и RK 512) с SFB-коммуникациями. PtP-соединение (соединение point-to-point, двухточечное) является последовательным соединением между двумя партнерами. Это могут быть устройства SIMATIC S7 с соответствующими CP или устройство SIMATIC и устройство другого производителя (не Siemens), например, принтер или устройство считывания штрихового кода.

S7-соединение принято к использованию в подсетях MPI, PROFIBUS и промышленного Ethernet с SFB-коммуникациями. S7-соединение – это соединение между устройствами SIMATIC S7 и может включать программирующие устройства и устройства интерфейса человек-машина. Обмен данными происходит через S7-соединение, или исполняются программные и управляющие функции.

Отказоустойчивое S7-соединение принято для подсетей PROFIBUS и промышленного Ethernet с SFB-коммуникациями. Отказоустойчивое S7-соединение создается между устойчивыми к сбоям устройствами SIMATIC S7, а также может быть использовано в соответствующим образом оборудованном PC.

Дополнительные пакеты «NCM S7 for PROFIBUS» и «NCM S7 for Industrial Ethernet» применяются для **параметризации CP**. В зависимости от установленного программного обеспечения NCM можно выбрать дополнительные типы соединений: FMS-, FDL-, транспортное ISO-, ISO-on-TCP-, TCP-, UDP- и email-соединение.

Построение активного соединения

Перед фактической передачей данных соединение должно быть образовано (инициализировано). Если партнеры по соединению имеют такую возможность, то определяется, какое устройство устанавливает соединение. Вы можете это сделать с помощью флажка «Active Connection Buildup» («Построение активного соединения») в окне свойств соединения (выберите соединение, затем нажмите Edit → Object Properties, Правка → Свойства Объекта).

Отправление сообщений рабочего режима

Партнеры по соединению с настроенным двухсторонним соединением могут обмениваться сообщениями рабочего состояния. Если локальный узел будет посылать свои сообщения рабочего состояния, активируйте соответствующий флажок в окне свойств соединения. В пользовательской программе CPU-партнера эти сообщения могут быть получены с помощью SFB 23 USTATUS.

Путь соединения

В качестве пути соединения (connection path) в окне свойств соединения показаны конечные точки соединения и подсети, через которые оно проходит. Если на выбор имеются несколько подсетей, STEP 7 выбирает их в таком порядке:

- 1) промышленный Ethernet;
- 2) промышленный Ethernet/TCP-IP;
- 3) MPI;
- 4) PROFIBUS.

Станция и CPU, через которые проходит соединение, показаны как конечные точки соединения. Список модулей с коммуникационными возможностями приведен в пункте «Interface» («Интерфейс»), определяя номер стойки слот. Если оба CPU расположены в одной стойке (например, процессоры CPU S7-400 в мультипроцессорном режиме), то окно просмотра отображает «PLC-internal» («PLC-внутренний»).

В пункте «Types» («Типы») вы должны выбрать подсеть, по которой будет проходить соединение. Если оба партнера, к примеру, подключены к одной подсети MPI и одной подсети PROFIBUS, то в пункте «Types» («Типы») ставится «MPI». Вы можете изменить данное определение на «PROFIBUS», и STEP 7 автоматически подберет остальные установки. После этого в пункте «Address» («Адрес») вы увидите MPI-адрес или PROFIBUS-адрес.

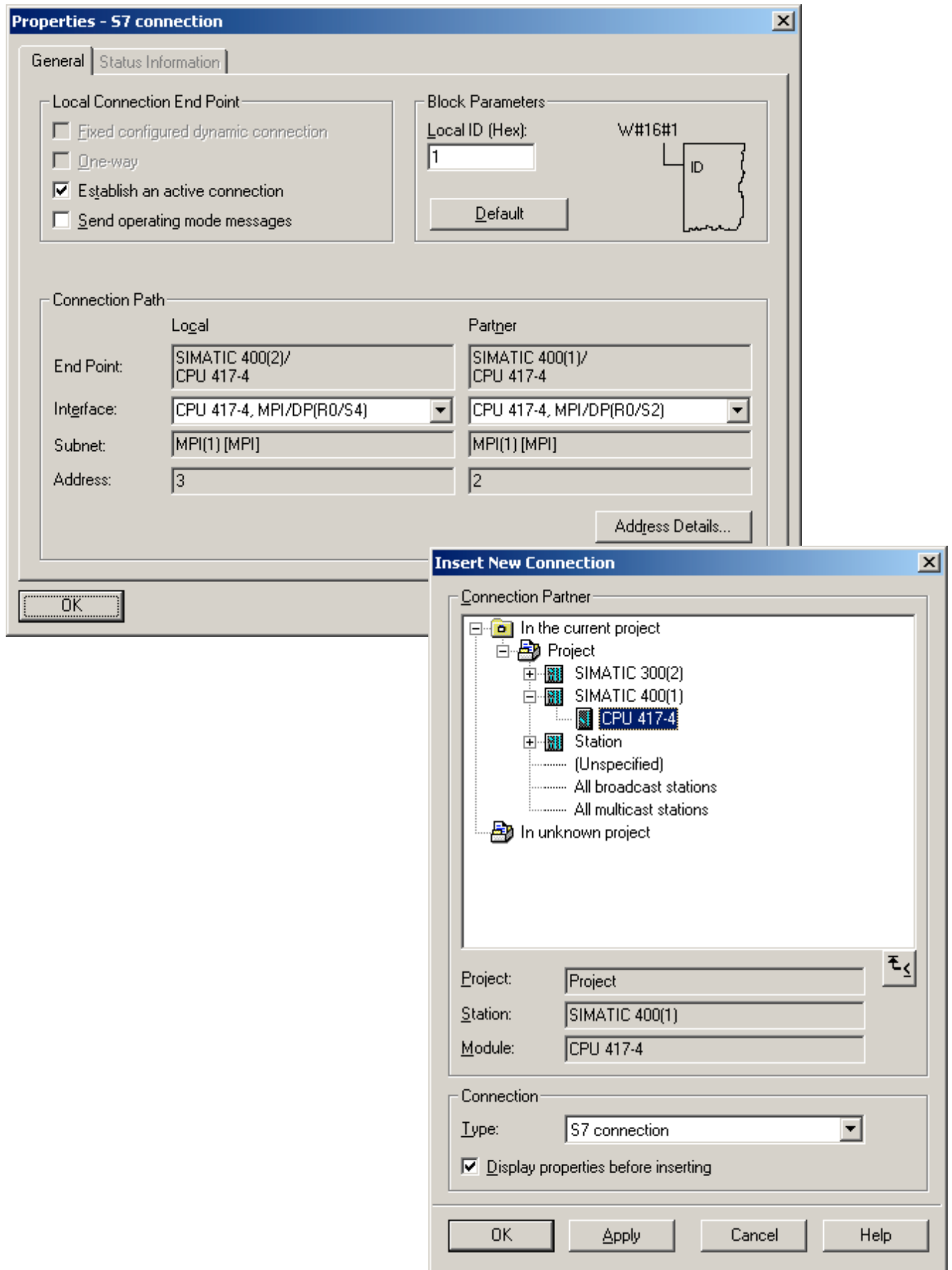


Рисунок 2.6 Конфигурирование коммуникационных соединений

Соединения между проектами

Для обмена данными между двумя модулями S7, принадлежащими различным SIMATIC-проектам, в таблице соединений для партнера по соединению вводится «unspecified» («неопределено») (в локальной станции в обоих проектах).

Убедитесь, что данные соединения в обоих проектах согласованы (STEP 7 не проверяет это). После сохранения и компиляции загрузите данные соединения в локальную станцию каждого проекта.

Подключение к станции, не являющейся станцией S7

В рамках проекта вы также можете определить в качестве партнеров по соединению станции, не являющиеся станциями S7:

- Другие станции (устройства производства не Siemens, а также станции S7 в другом проекте);
- Программирующие устройства/PC;
- Станции SIMATIC S5.

Для конфигурирования соединения требуется, чтобы подобная станция присутствовала в качестве объекта в контейнере проекта, и она была подключена к соответствующей подсети в свойствах станции. Например, выберите станцию в Network Configuration, нажмите Edit → Object Properties (Правка → Свойства объекта) и соедините станцию с нужной подсетью во вкладке «Interfaces» («Интерфейсы»).

2.4.4 Сетевые переходы

Если устройство программирования подключено к подсети, то оно может получить доступ ко всем другим узлам этой подсети. Например, из одной точки соединения вы можете программировать и отлаживать все станции S7, подключенные к подсети MPI.

Если иная подсеть, такая как подсеть PROFIBUS, подключена к станции S7, программирующее устройство также может получить доступ к станциям другой подсети. Для этого требуется, чтобы станция с подсетевым переходом имела возможность маршрутизации, то есть она проведет передаваемые фреймы сообщений.

Когда конфигурация сети откомпилирована, для станций с подсетевыми переходами автоматически генерируются таблицы маршрутизации (routing tables), содержащие всю необходимую информацию.

Все доступные коммуникационные партнеры в сети предприятия в рамках S7-проекта должны быть сконфигурированы и снабжены «знанием» о том, к каким станциям можно получить доступ через какие подсети и подсетевые переходы.

Если вы хотите получить доступ ко всем узлам подсети с использованием программирующего устройства из одной точки соединения, то вы должны сконфигурировать точку соединения. Введите «заполнитель» («placeholder»), PG/PC-станцию из каталога сетевых объектов в конфигурации сети соответствующей подсети. Вы должны сконфигурировать PG/PC-станцию на каждой подсети, к которой вы хотите подключить устройство программирования.

Во время выполнения операций вы должны подключить программирующее устройство к подсети и выбрать команду меню PLC → Assign PG/PC (PLC → Назначить PG/PC). Это адаптирует интерфейсы программирующего устройства к заданным установкам подсети. Перед отключением устройства программирования от подсети выберите PLC → Undo PG/PC Assignment (PLC → Отменить назначение PG/PC).

Если вы работаете в онлайн-режиме с программирующим устройством, не содержащим правильного проекта, для доступа в сеть вам потребуется ID подсети S7. ID подсети S7 включает в себя два номера: номер проекта и номер подсети. Вы можете получить ID подсети в Network Configuration путем выбора подсети и Edit → Object Properties (Правка → Свойства объекта) на вкладке «General» («Общие»).

2.4.5 Загрузка данных соединения

Чтобы активировать соединения, вы должны загрузить таблицу соединений в PLC после сохранения и компиляции (все таблицы соединений во все «активные» CPU).

Требование: вы находитесь в окне сети, и таблица соединений видима. Программирующее устройство является узлом подсети, по которой данные соединений должны быть загружены в модули с коммуникационными возможностями.

Всем узлам подсети присвоены уникальные узловые адреса. Модули, которым данные соединений будут переданы, находятся в режиме STOP.

С помощью опции меню PLC → Download → ... (PLC → Загрузить...) вы можете передать данные соединений и конфигураций доступным модулям. В зависимости от того, какой объект отмечен и какая команда меню выбрана, вы можете выбрать между следующим:

- Selected Stations (→ Отмеченные станции),
- Selected and Partner Stations (→ Отмеченные и станции-партнеры),
- Selected Connections (→ Отмеченные соединения),
- Stations on Subnet (→ Станции в подсети),
- Connections and Routers (→ Соединения и маршрутизаторы).

Откомпилированные данные соединений также являются составляющей частью *System data* (*Системных данных*) в контейнере *Blocks* (*Блоки*). Передача системных данных и последующего запуска центральных процессоров также эффективно передает данные соединений модулям с коммуникационными возможностями.

Для работы в онлайн-режиме через MPI дополнительных аппаратных средств не требуется. Если вы подключаете PC к сети или осуществляете включение программирующего устройства в сеть Ethernet или PROFIBUS, то вам потребуется соответствующий интерфейсный модуль. Параметризация модуля выполняется в приложении «Set PG/PC Interface» («Установка интерфейса PG/PC») панели управления Windows (Windows Control Panel).

2.5 Создание программы S7

2.5.1 Введение

Пользовательская программа создается в папке *S7 Program (S7-программа)*. Вы можете назначить этот объект в иерархии проекта центрального процессора (CPU) или можете создать его независимо от CPU. Он содержит объект *Symbols (Символы)*, контейнеры *Source Files (Исходные файлы)* и *Blocks (Блоки)* (рисунок 2.7).

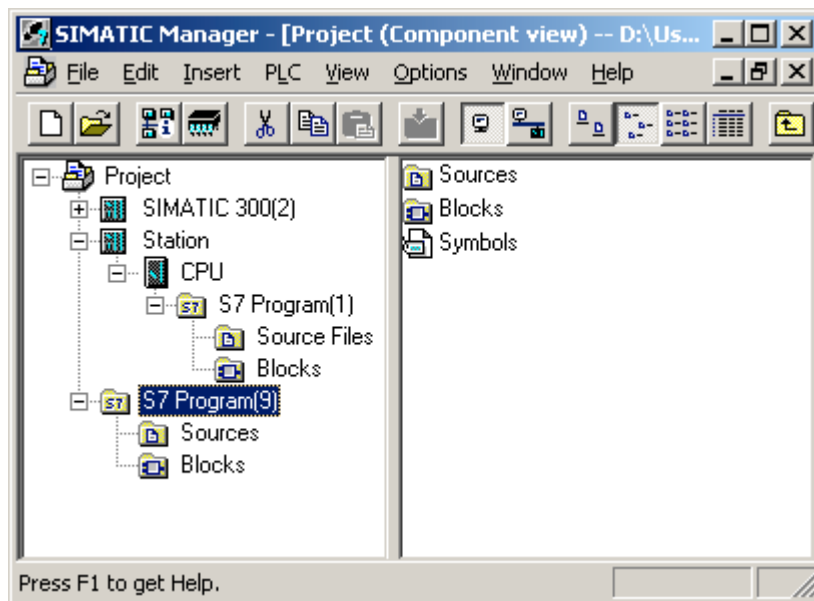


Рисунок 2.7 Объекты для программирования

Используя **пошаговое** создание программ, вы вводите программу непосредственно блок-за-блоком (block-by-block). Синтаксис вводимой информации немедленно проверяется. Блок компилируется одновременно с записью и сохранением его в контейнере *Blocks (Блоки)*. С помощью пошагового программирования вы также можете редактировать блоки в CPU в онлайн-режиме, даже во время выполнения операций. Пошаговое программирование доступно для всех основных языков.

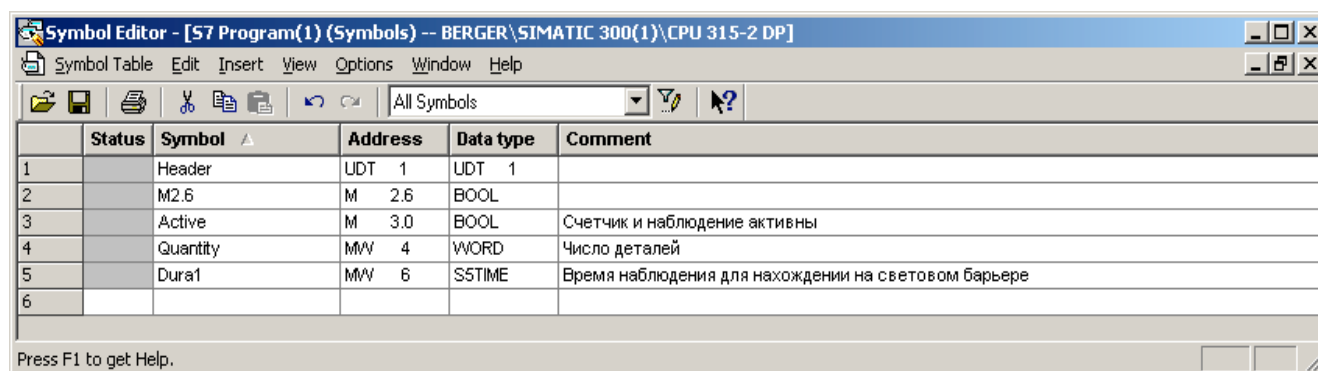
В случае способа создания программ, **ориентированного на источник (исходные файлы)**, вы создаете один или более исходных файлов программы и сохраняете их в контейнере *Source Files (Исходные файлы)*. Исходные программные файлы представляют собой текстовые файлы в формате ASCII, содержащие программные операторы для одного или более блоков, возможно даже для всей программы. Они компилируются, и откомпилированные блоки помещаются в контейнер *Blocks (Блоки)*. Ориентированное на исходные файлы создание программ используется в STL; вы не можете воспользоваться данным способом программирования в LAD или FBD, но программы, созданные с помощью LAD или FBD, могут быть сохранены как исходные файлы.

В программе обрабатываются сигнальные состояния значений адресов. Адресом является, например, вход I 1.0 (*абсолютная адресация – absolute addressing*). С помощью таблицы символов под объектом *Symbols* (*Символы*) вы можете назначить адресу символ (буквенно-цифровое имя, например, «Switch motor on» или «Включить мотор») и затем обращаться к нему по этому имени (*символическая адресация – symbolic addressing*). В свойствах контейнера автономных объектов *Blocks* (*Блоки*) определяется, абсолютный адрес или символ является определяющим для уже откомпилированных блоков при следующем сохранении в случае изменения в таблице символов (*приоритет адреса – address priority*).

2.5.2 Таблица символов

В управляющей программе вы работаете с адресами; это входы, выходы, таймеры, блоки. Вы можете присвоить абсолютные адреса (например, I1.0) или символические адреса (например, Start signal). Символическая адресация использует вместо абсолютного адреса имени. Применяя осмысленные имена, вы можете сделать вашу программу более читаемой.

В символической адресации различают *локальные (local)* и *глобальные (global)* символы. Локальный символ известен только в блоке, в котором он был определен. Вы можете использовать одинаковые локальные символы в разных блоках для различных целей. Глобальный символ известен во всей программе и имеет одинаковое значение во всех блоках. Глобальные символы определяются в таблице символов (объект *Symbols* (*Символы*) в контейнере *S7 Program* (*S7-программа*)).



Status	Symbol	Address	Data type	Comment
1	Header	UDT 1	UDT 1	
2	M2.6	M 2.6	BOOL	
3	Active	M 3.0	BOOL	Счетчик и наблюдение активны
4	Quantity	MW 4	WORD	Число деталей
5	Dura1	MW 6	SSTIME	Время наблюдения для нахождения на световом барьере
6				

Рисунок 2.8 Пример таблицы символов

Глобальный символ начинается с буквы, и его длина может составлять до 24 знаков. Он также может включать в себя пробелы, специальные литеры и национальные буквы, такие как умляут. Исключениями являются знаки 00_{hex}, FF_{hex} и кавычки ("). При написании программ символы со специальными литерами вы должны заключать в кавычки. В скомпилированных блоках все глобальные символы программный редактор отображает в кавычках. Комментарий символа может включать до 80 знаков.

В таблице символов вы можете присвоить имена следующим адресам и объектам:

- Входам I, выходам Q, периферийным входам PI и периферийным выходам PQ;
- Меркерам M, таймерам T и счетчикам C;
- Кодовым блокам OB, FB, FC, SFC, SFB и блокам данных DB;
- Определенным пользователем типам данных UDT;
- Таблице переменных (variable table) VAT.

Адреса данных и блоки данных включены в число локальных адресов; ассоциированные символы определяются в разделе описаний блока данных в случае глобальных блоков данных и разделе описаний функционального блока в случае блоков данных экземпляров.

При создании программы S7 SIMATIC-менеджер создает также пустую таблицу символов *Symbols* (*Символы*). Вы можете ее открыть, определить глобальные символы и назначить их абсолютным адресам (рисунок 2.8). В программе S7 может быть только одна таблица символов.

Тип данных является частью определения символа. Он определяет специфические свойства данных за символом, по существу представление содержимого данных. Например, тип данных BOOL (логический) идентифицирует двоичную переменную, а тип данных INT (целый) обозначает цифровую переменную, чье содержимое представляет 16-битное целое. Обратитесь к параграфу 3.5 «Переменные, константы и типы данных», в нем приводится подробная информация о типах данных.

Используя пошаговое программирование, вы создаете таблицу символов перед вводом программы; также вы можете добавить или откорректировать отдельные символы во время ввода программы. В случае ориентированного на источник программирования завершенная таблица символов должна быть доступна, когда компилируется исходная программа.

Импортирование, экспортирование

Таблицы символов могут быть импортированы и экспортированы. «Экспорт» означает создание файла с содержимым вашей таблицы символов. Вы можете выбрать для этого либо всю таблицу символов, либо ограниченное фильтрами подмножество, либо только отдельные строки. В качестве формата данных вы можете выбрать между чистым ASCII-текстом (расширение *.asc), последовательным списком назначений (*.seq), форматом системных данных (System Data Format, *.sdf для Microsoft Access) и форматом обмена данными (Data Interchange Format, *.dif для Microsoft Excel). Экспортированный файл можно откорректировать в соответствующем редакторе. Таблицу символов в одном из названных выше форматов можно также импортировать.

Свойства специальных объектов

С помощью команды Edit → Special Object Properties → ... (Правка → Свойства специальных объектов → ...) вы можете установить атрибуты для каждого символа в таблице символов. Атрибуты или свойства используются

- В функциях интерфейса человек-машина (HMI-функциях) для наблюдения с использованием WinCC;
- При конфигурировании коммуникаций;
- При конфигурировании сообщений;
- При наблюдении процесса с использованием S7-PDIAG.

Видимыми установки делает команда меню View → Columns O, M, C, R (Вид → Столбцы O, M, C, R). С помощью Options → Customize (Опции → Настроить) вы можете установить, будут ли копироваться свойства специальных объектов, и определить поведение при импортировании символов.

2.5.3 Редактор программ

Для создания пользовательских программ базовый пакет STEP 7 содержит редактор программ (Program Editor) для языков программирования LAD, FBD и STL. С LAD и FBD вы будете программировать пошагово, то есть будете вводить непосредственно исполняемый блок; на рисунке 2.9 показаны возможные действия для этого.

Если вы используете символическую адресацию для глобальных адресов, то в случае пошагового программирования символы уже должны быть присвоены абсолютным адресам; тем не менее, вы можете ввести новые символы или изменить их во время ввода программы.

Блоки LAD/FBD могут быть «декомпилированы», то есть из кода MC7 может быть воссоздан читаемый блок без использования автономной базы данных (вы можете прочитать любой блок из CPU с помощью программирующего устройства без ассоциированного проекта). Кроме того, из любого откомпилированного блока может быть создана исходная программа STL.

Запуск редактора программ

Доступ к редактору программ можно получить при открытии блока в SIMATIC-менеджере, например, по двойному щелчку на автоматически генерируемом символе организационного блока OB 1 или через панель задач Windows, нажав Start → Simatic → STEP 7 → LAD, STL, FBD – Program S7 Blocks (Пуск → Simatic → STEP 7 → LAD, STL, FBD → Программировать блоки S7).

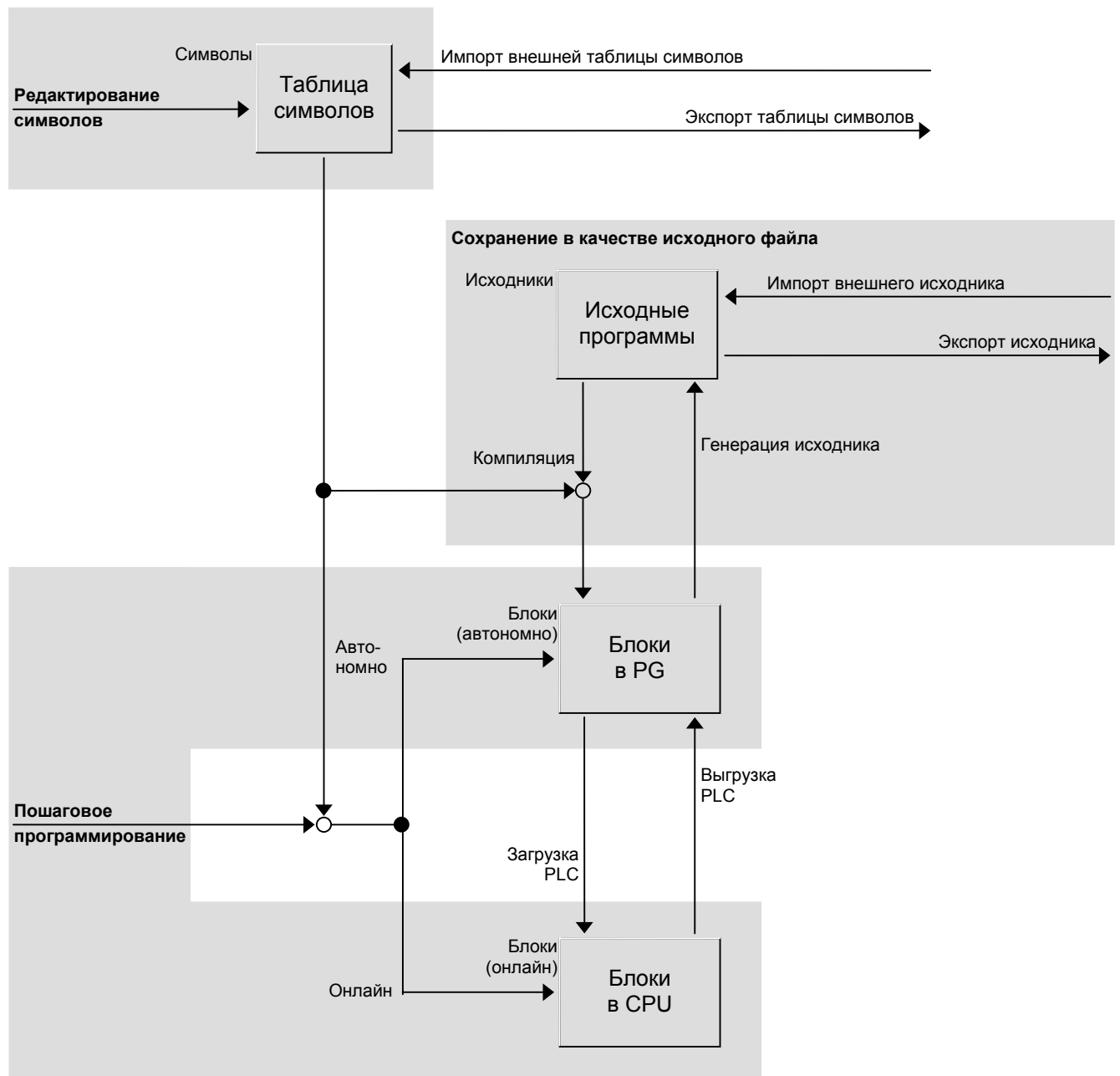


Рисунок 2.9 Написание программ с помощью редактора LAD/FBD

Установить свойства редактора программ можно с помощью **Options → Customize** (**Опции → Настроить**). На вкладках «Editor» («Редактор») и «Create Blocks» («Создание блоков») выберите свойства, с которыми будет сгенерирован и отображен новый блок, к таковым относятся, например, язык создания, предвыбор для комментариев и символов.

При открытии компилированного блока в контейнере *Blocks (Блоки)* (к примеру, двойным щелчком), он открывается для пошагового программирования.

Также возможно вызвать блоки, которые были написаны на другом языке программирования, например, STL. Пользовательская программа создается блок за блоком, и

каждый блок, в конечном счете, содержит исполняемый код МС7 независимо от языка программирования, использованного для его написания.

Используя пошаговое программирование, вы можете выполнять все функции программирования с одним исключением: если вы хотите предусмотреть защиту блоков (KNOW_HOW_PROTECT), сделать это вы сможете только через исходный программный файл (за более подробной информацией по этой теме обратитесь к параграфу 24.1 «Защита блоков»).

Пошаговое программирование

Применяя пошаговое программирование, вы можете редактировать блоки как в автономном, так и в онлайн-режиме в контейнере *Blocks* (Блоки). Редактор проверит введенные вами в пошаговом режиме данные, как только вы выйдете из сети (network). При закрытии блока он немедленно компилируется, так что только не содержащие ошибок блоки могут быть сохранены.

На вкладке «Create Blocks» («Создание блоков»), нажав Options → Customize (Опции → Настроить), можно установить автоматическое обновление ссылочных данных при записи блока.

Блоки могут редактироваться как автономно в базе данных устройства программирования, так и онлайн в CPU, обычно называемом «программируемый контроллер» или «PLC». Для этих целей SIMATIC-менеджер предоставляет автономное и онлайн-новое окно; одно отличается от другого метками в линейке заголовка.

В автономном окне происходит редактирование блоков непосредственно в базе данных PG. Если вы находитесь в редакторе, можете сохранить измененный блок в автономной базе данных с помощью опции меню File → Save (Файл → Сохранить) и передать его в CPU, нажав PLC → Download (PLC → Загрузить). Если вы хотите сохранить открытый блок под другим номером или в другом проекте, или если вы хотите переместить его в библиотеку или в другой CPU, воспользуйтесь командой меню File → Save As (Файл → Сохранить как).

Чтобы редактировать блок в CPU, откройте этот блок в онлайн-режиме. Это переместит блок из CPU в программирующее устройство, и, таким образом, он может редактироваться. Вы можете записать отредактированный блок обратно в CPU, выбрав PLC → Download (PLC → Загрузить). Если CPU находится в режиме RUN (исполнение), то центральный процессор обработает измененный блок в следующем цикле программного сканирования. Если вы хотите сохранить отредактированный в режиме онлайн блок также в автономной базе данных, то это можно выполнить с помощью команды File → Save (Файл → Сохранить).

Параграфы 2.6.4 «Загрузка пользовательской программы в CPU» и 2.6.5 «Обработка блоков» продолжают рассказ об онлайн-режиме программирования. Параграфы 3.3 «Программирование кодовых блоков» и 3.4 «Программирование блоков данных» показывают, как вводить блоки LAD/FBD.

2.5.4 Обновление или генерирование исходных файлов

На вкладке «Source Files» («Исходные файлы»), вызываемой командой Options → Customize (Опции → Настроить), вы можете выбрать опцию «Generate source file automatically» («Автоматически генерировать исходный файл») с тем, чтобы при сохранении (пошагово созданного) блока исходный файл программы обновлялся или создавался, если он еще не был сгенерирован.

Имя нового исходного файла может быть образовано на основе абсолютного адреса или символического адреса. Адреса могут быть переданы в исходный файл в абсолютной или символической форме.

Нажмите кнопку «Execute» («Выполнить») и в появившемся диалоговом окне выберите блоки, из которых вы хотите сгенерировать программный исходный файл.

Вы можете экспортировать исходные файлы из проекта путем выбора в SIMATIC-менеджере Edit → Export Source File (Правка → Экспортировать исходный файл). После этого вы можете произвести дальнейшую обработку ASCII-файлов, например, в другом текстовом редакторе. Исходные файлы также могут быть импортированы обратно в контейнер *Source Files* (*Исходные файлы*) с помощью Insert → External Source File (Вставка → Внешний исходный файл).

Если вы генерируете исходный файл из блока, созданного с помощью LAD или FBD, то вы сможете из этого исходного файла вновь сгенерировать LAD- или FBD-блок. Компиляция исходного файла осуществляется после его открытия в SIMATIC-менеджере по двойному щелчку и выбора в редакторе программ File → Compile (Файл → Компилировать). В контейнере *Blocks* (*Блоки*) создается STL-блок. Откройте этот блок и перейдите к обычному для вас представлению, нажав View → LAD (Вид → LAD) или View → FBD (Вид → FBD). После записи блок сохраняет это свойство.

Если при создании исходного файла вы выбрали установку «Addresses – Symbolic» («Адреса – символические»), для компилирования исходного файла вам потребуется завершенная таблица символов. Таким образом, вы можете определить в таблице символов различные абсолютные адреса и, после компиляции, получить в итоге программу, например, с различными входами и выходами. Это позволяет вам адаптировать программу к различным конфигурациям аппаратных средств. Для этой цели лучше сохранить эти исходные файлы (которые не зависят от аппаратной адресации), например, в библиотеке.

Перемонтаж (переадресация)

Функция *Rewire* (*Перемонтаж*) позволяет вам заменить адреса отдельно в откомпилированных блоках или во всей программе пользователя. Например, вы можете заменить входные биты с I0.0 по I0.7 на входные биты I 16.0, ..., I 16.7.

Допустимыми адресами являются входы, выходы, биты памяти, таймеры и счетчики, а также функции FC и функциональные блоки FB.

В SIMATIC-менеджере вы должны выбрать объекты, в которых вы хотите выполнить перезапись; выбираете один блок, группу блоков, отмечая их мышью при нажатой клавише Ctrl, или всю папку *Blocks* (Блоки) программы пользователя. Нажав Options → Rewire (Опции → Перемонтировать), вы попадете в таблицу, в которой вы можете ввести заменяемые старые адреса и новые адреса. После подтверждения кнопкой «ОК» SIMATIC-менеджер производит обмен адресов.

При «перемонтаже» блоков сначала измените номера блоков, затем выполните переадресацию, которая модифицирует вызовы соответствующим образом.

В отображаемом позже файле указано, в каких блоках сделаны изменения и сколько.

Другие возможные методы переадресации:

- С отдельно откомпилированными блоками можно использовать функцию *Address priority* (Приоритет адреса);
- Если имеется программный исходный файл с символической адресацией, вы должны изменить абсолютные адреса в таблице символов перед компиляцией, и в вашей программе после компиляции не будет «зашитых» адресов, то есть она будет связана с другими адресами.

2.5.5 Приоритет адреса

В окне свойств автономного контейнера объектов *Blocks* (Блоки), во вкладке «Blocks» («Блоки») вы можете установить, абсолютный или символический адрес будет иметь приоритет для уже записанных блоков, когда они отображаются и сохраняются вновь после изменений в таблице символов или в описании (объявлении) или назначении глобальных блоков данных.

Присваиваемое по умолчанию значение – «Absolute address has priority» («Приоритет имеет абсолютный адрес») (так же, как и в предыдущих версиях STEP 7). Это означает, что когда в таблице символов сделано изменение, абсолютный адрес в программе сохраняется, а символ соответственно меняется. Если установлено «Symbol has priority» («Приоритет имеет символ»), меняется абсолютный адрес, символ сохраняется.

Пример:

Таблица символов содержит следующее:

```
I 1.0 "Limit_switch_up"
I 1.1 "Limit_switch_down"
```

В программе уже скомпилированного блока вход I1.0 сканируется:

```
I 1.0 "Limit_switch_up"
```

Если назначения для входов I1.0 и I1.1 в таблице символов изменены на:

```
I 1.0 "Limit_switch_down"
I 1.1 "Limit_switch_up"
```

и уже откомпилированный блок считан, то программа содержит:

```
I 1.1 "Limit_switch_up"
```

если установлено «Symbol has priority» («Приоритет имеет символ») и установлено «Absolute address has priority» («Приоритет имеет абсолютный адрес»), то программа содержит:

```
I 1.0 "Limit_switch_down"
```

Если в результате изменения в таблице символов удалены все связи между абсолютным адресом и символом, то в случае установленного флажка «Absolute address has priority» («Приоритет имеет абсолютный адрес») оператор (statement) будет содержать абсолютный адрес (даже с символическим отображением, потому что символа, разумеется, не будет). Если установлено «Symbol has priority» («Приоритет имеет символ»), оператор отклоняется как ошибочный (потому, что отсутствует обязательный абсолютный адрес).

Если установлено «Symbol has priority» («Приоритет имеет символ»), то пошагово запрограммированные блоки с символической адресацией в случае изменения таблицы символов сохраняют свои символы. Таким образом, уже запрограммированный блок может быть «перешит» сменой назначения адреса.

Заметьте, что эта «перешивка» не происходит автоматически, потому что уже откомпилированные блоки содержат исполняемый MC7-код операторов с абсолютными адресами. Изменения касаются только соответствующих блоков – следующих за соответствующим сообщением – после того, как они были открыты и сохранены снова.

2.5.6 Ссылочные данные

В качестве дополнения к самой программе SIMATIC-менеджер показывает вам ссылочные данные (reference data), которые вы можете использовать как основу для исправлений или тестов. Эти ссылочные данные включают следующее:

- Перекрестные ссылки
- Резервированные адреса (I, Q, M и T, C)
- Структура программы
- Неиспользуемые символы
- Адреса без символов

Чтобы сгенерировать ссылочные данные, выберите объект *Blocks* (Блоки) и команду меню Options → Reference Data → Display (Опции → Ссылочные данные → Отобразить). Представление ссылочных данных может быть установлено отдельно для каждого рабочего окна с помощью View → Filter... (Вид → Фильтр...); вы можете сохранить установки для более позднего редактирования путем выбора опции «Save as Standard» («Сохранить в качестве стандартного»). Вы можете открыть и просмотреть несколько списков одновременно.

С помощью Options → Customize (Опции → Настроить) в редакторе программ на вкладке «Create Blocks» («Создание блоков») задайте, будут ли обновляться ссылочные данные при компилировании программного исходного файла или при сохранении пошагово написанного блока.

Обратите внимание, что ссылочные данные доступны только при автономном управлении данными; автономные ссылочные данные отображаются даже в случае, если функция вызвана в блоке, открытом в режиме онлайн.

Перекрестные ссылки

Список перекрестных ссылок показывает, как используются адреса и блоки в пользовательской программе. Он включает абсолютный адрес, символ (если есть), блок, в котором задействован адрес, как он использовался (чтение или запись) и связанная с языком информация. Для STL столбец с языковой информацией отображает сеть, в которой использовался адрес; для SCL – номер строки и столбца. Щелкните на заголовке столбца, чтобы отсортировать таблицу по содержимому столбца.

Если адрес выбран, команда меню Edit → Go To → Line (Правка → Перейти к → Строка) запускает редактор программ и отображает адрес в программной среде.

Список перекрестных ссылок показывает адреса, выбранные вами при помощи View → Filter... (Вид → Фильтр...), например, память меркеров. По двойному щелчку на адресе редактор открывает блок, отображенный в той строке, в которую занесен адрес. К тому же STEP 7 использует фильтр, сохраненный как «Standard» («стандартный»), каждый раз, когда открывает список перекрестных ссылок.

Преимущество: перекрестные ссылки показывают, были ли также отсканированы или сброшены ссылочные адреса. Кроме того, они показывают, в каких блоках используются адреса (возможно более одного раза).

Назначения

Список ссылок I/Q/M показывает, какие биты в областях адресов I, Q и M назначены в программе. В каждой строке отображается один байт, разбитый на биты. Также показано, как осуществляется доступ – побайтно, по слову или по двойному слову. Список ссылок T/C показывает таймеры и счетчики, задействованные в программе. В строке отображаются десять таймеров или счетчиков.

Преимущество: список демонстрирует, были ли определенные области адресов (неправильно) назначены или где еще есть доступные адреса.

Структура программы

Структура программы показывает иерархию вызовов блоков в пользовательской программе. «Начальный блок» для иерархии вызовов определяется посредством установок фильтра. Вы можете выбрать между двумя различными представлениями: *Древовидная структура* показывает все вложенные уровни вызовов блоков. Управление отображением вложенных уровней осуществляется с помощью квадратиков со знаками «+» и «-». Показаны потребности во временных локальных данных для полного пути, следующего за начальным блоком, и/или на каждый путь вызова. Щелкните правой кнопкой мыши для вызова меню, в котором вы можете открыть блок, переключиться в область вызова или в окно с дополнительной информацией о блоке.

Структура предок/потомок показывает два уровня вызовов с одним вызовом блока. Связанная с языком информация также включена.

Преимущество: какие блоки были задействованы? Все ли запрограммированные блоки были вызваны? Каковы потребности блоков во временных локальных данных? Достаточно ли заданных требований по локальным данным на приоритетный класс (на организационный блок)?

Неиспользуемые символы

Этот список отображает все адреса, которые размещены в таблице символов, но не используются в программе. В списке приведены символ, адрес, тип данных и комментарий из таблицы символов.

Преимущество: какие адреса из списка были нечаянно забыты в начале написания программы? Или они, возможно, являются лишними и не нужны?

Адреса без символов

В этом списке собраны все используемые в программе адреса, которым символы не назначены. В списке приводятся эти адреса и как часто они использовались.

Преимущество: были ли адреса использованы ошибочно (случайно или из-за опечатки)?

2.5.7 Мультиязыковые комментарии и отображение текста

SIMATIC-менеджер позволяет управлять несколькими языковыми версиями комментариев и текста на дисплее.

Выбор языка комментариев и отображаемого текста влияет на тексты, вводимые пользователем. Язык сеанса, назначенный для меню и сообщений об ошибках, например, устанавливается STEP 7 и выбирается в SIMATIC-менеджере во вкладке «Language» («Язык») после нажатия Options → Customize (Опции → Настроить). На

этой вкладке также устанавливается мнемоника, то есть язык, который STEP 7 использует для операндов и операций. Все три установки независимы друг от друга.

Общая процедура

Вы ввели тексты на исходном языке, например, английском и хотите сгенерировать немецкую версию вашей программы. Чтобы это сделать, экспортируйте требуемый текст или типы текстов. Экспортированный файл имеет расширение *.csv, и вы можете отредактировать его, к примеру, в Microsoft Excel. Можно ввести перевод для каждого текста. Импортируйте законченную таблицу перевода обратно в ваш проект. Теперь вы можете переключать языки. Это можно сделать с несколькими языками.

Экспорт и импорт текстов

В SIMATIC-менеджере выберите объекты, содержащие комментарии, которые вы хотите перевести, например, таблицу символов, контейнер блоков, несколько блоков или один блок. Выберите команду меню Options → Manage Multilingual Texts → Export (Опции → Управление мультязыковыми текстами → Экспорт). В появившемся диалоговом окне введите местоположение для сохранения экспортированного файла и требуемый язык. Выберите типы текстов, которые вы хотите перевести (таблица 2.2).

Для каждого текстового типа генерируется отдельный файл, к примеру, файл SymbolComment.csv для комментариев из таблицы символов.

Откройте экспортированный файл(ы) в Microsoft Excel с помощью диалогового окна File → Open (Файл → Открыть) (не двойным щелчком клавишей мыши). Экспортированные тексты отображаются в первом столбце, а перевод вы можете ввести во втором столбце.

Ввести переведенные тексты обратно в проект вы можете с помощью Options → Manage Multilingual Texts → Import (Опции → Управление мультязыковыми текстами → Импорт). Регистрационный файл (log-файл) предоставляет информацию об импортированном файле и любых ошибках, которые могут возникнуть.

Обратите внимание, что имя импортируемого файла не должно изменяться, так как имеется прямая связь между именем и содержащимися в файле текстовыми типами.

Таблица 2.2 Текстовые Типы переводимых текстов (выбор)

Тип текста	Значение
BlockTitle	Заголовок блока
BlockComment	Комментарий блока
NetworkTitle	Заголовок сегмента (сети, схемы)
NetworkComment	Комментарий к сегменту (сети, схемы)
LineComment	Комментарий строки
InterfaceComment	Комментарий в <ul style="list-style-type: none"> ➤ таблице описаний блоков кода ➤ блоках данных ➤ пользовательских типах данных UDT
SymbolComment	Комментарий символа

Выбор и удаление языка

В SIMATIC-менеджере вы можете переключаться на любой импортированный язык при помощи Options → Manage Multilingual Texts → Change Language (Опции → Управление мультиязыковыми текстами → Сменить язык). Смена языка выполняется для объектов (блоков, таблицы символов), для которых были импортированы соответствующие тексты. Эта информация содержится в log-файле.

Импортированные языки можно удалить с помощью команды меню Options → Manage Multilingual Texts → Delete Language (Опции → Управление мультиязыковыми текстами → Удалить язык).

2.6 Онлайнный (интерактивный) режим

Вы создаете конфигурацию аппаратного обеспечения и пользовательскую программу на устройстве программирования, обычно называемом «инженерная система» («engineering system», ES). Программа S7 автономно сохранена здесь на жестком диске, а также в скомпилированном виде.

Чтобы передать программу в CPU, вы должны подключить к нему устройство программирования. Вы устанавливаете «онлайнное» соединение. Этим соединением можно воспользоваться для определения рабочего (операционного) состояния CPU и назначенных модулей, то есть выполнить функции диагностики.

2.6.1 Подключение PLC

Для онлайнного соединения требуется механическое соединение между MPI-интерфейсом программирующего устройства и MPI-интерфейсом CPU. Соединение является единственным, когда CPU – единственный подключенный программируемый модуль. Если в подсети MPI имеется несколько CPU, каждому из них должен быть присвоен уникальный узловой номер (MPI-адрес). Установка MPI-адреса происходит при инициализации CPU. Пред объединением всех CPU в одну сеть, подключите устройство программирования к одному CPU и передайте объект *System Data* (*Системные данные*) из автономного *Blocks* (*Блоки*) пользовательской программы или направьте с помощью редактора Hardware Configuration, используя команду меню PLC → Download (PLC → Загрузить). Это назначит CPU его собственный уникальный MPI-адрес («присвоение имени») наряду с другими свойствами.

MPI-адрес CPU в MPI-сети может быть изменен в любой момент путем передачи в CPU новой записи данных параметризации, содержащей новый MPI-адрес. Обратите особое внимание на то, что новый MPI-адрес вступает в силу немедленно. Так как программирующее устройство сразу устанавливает новый адрес, вы должны адаптировать другие приложения, такие как коммуникации глобальных данных, к новому MPI-адресу.

MPI-параметры сохраняются в CPU даже после сброса памяти. Таким образом, к CPU можно обращаться и после сброса памяти. Устройство программирования всегда может работать в онлайнном режиме в CPU, даже с независимой от модулей программой и без загруженного проекта.

Если нет загруженного проекта, вы должны установить соединение с CPU с помощью PLC → Display Accessible Nodes (PLC → Отобразить доступные узлы). Эта команда отобразит окно проекта со структурой «*Accessible Nodes*» – «Module (MPI=n)» – «Online User Program (*Blocks*)» или «*Доступные узлы*» – «Модуль (MPI=n)» – «Онлайнная программа пользователя (*Блоки*)». При выборе объекта *Module* (*Модуль*) вы можете использовать интерактивные функции, такие как изменение рабочего режима и проверка состояния модуля. Выбор объекта *Blocks* (*Блоки*) отобразит блоки в пользовательской памяти CPU. И вы можете редактировать (изменять, удалять, вставлять) отдельные блоки.

Вы можете считать обратно системные данные из подключенного CPU с целью, скажем, продолжения работы на основе имеющейся конфигурации без наличия соответствующего проекта в системе управления данными программирующего устройства. Создайте новый проект в SIMATIC-менеджере, отметьте проект и нажмите PLC → Upload Station (PLC → Выгрузить станцию). После определения нужного CPU в появляющемся диалоговом окне онлайнные системные данные сгружаются на жесткий диск.

Если в проектном окне имеется **независимая от CPU программа**, создайте соответствующее (ассоциированное) онлайнное окно проекта. Если к MPI-сети подключены и доступны несколько CPU, при отмеченной S7-программе выберите Edit → Object Properties (Правка → Свойства объекта) и на вкладке «Addresses Module» («Адреса модуля») установите номер монтажной стойки и слота CPU.

Если вы в онлайнном окне выберите *S7 Program (S7-программу)*, все онлайнные функции подключенных CPU будут для вас доступны. *Blocks (Блоки)* показывает блоки, расположенные в пользовательской памяти CPU. Если блоки в автономной программе согласованы с блоками онлайнной программы, то вы можете отредактировать блоки в пользовательской памяти, используя информацию из системы управления данными устройства программирования (символический адрес, комментарии).

Когда вы переводите **назначенную CPU программу** в онлайнный режим, вы можете модифицировать ее, как если бы это была независимая от CPU программа. Кроме того, теперь вы можете сконфигурировать SIMATIC-станцию, то есть установить параметры и адрес CPU и параметризовать модули.

2.6.2 Защита программы пользователя

С помощью соответствующим образом оснащенных CPU доступ к программе пользователя может быть защищен паролем. Каждый, кто владеет паролем, имеет неограниченный доступ к пользовательской программе. Для тех, кто пароля не знает, вы можете определить три уровня защиты. Установка уровней защиты производится в Hardware Configuration, на вкладке «Protection» («Защита») во время параметризации CPU.

Уровень защиты 1: блокировочное положение переключателя

Данный уровень защиты установлен по умолчанию (без пароля). При этом пользовательская программа защищена переключателем выбора режимов, который расположен на лицевой стороне центрального процессора. В положениях RUN-P (исполнение-P) и STOP вы имеете неограниченный доступ к пользовательской программе; в положении RUN возможен только доступ для чтения через программирующее устройство. В этом положении вы можете вынуть блокирующий переключатель, так что с помощью переключателя нельзя будет изменить режим.

Вы можете обойти защиту положением RUN блокирующего переключателя путем выбора опции «Can be revoked with password» («Может быть отменен паролем»), к

примеру, если к CPU, а вместе с ним и к блокирующему переключателю доступ затруднен или они находятся на расстоянии.

Уровень защиты 2: защита от записи

На этом уровне защиты программу пользователя можно только прочитать, несмотря на положение блокирующего переключателя.

Уровень защиты 2: защита от чтения/записи

Доступа к пользовательской программе нет, несмотря на положение блокирующего переключателя.

Защита паролем

Если вы выберете уровень защиты 2 или 3 или уровень 2 с «Can be revoked with password», вы получите запрос на задание пароля. Длина пароля может быть до 8 символов.

При попытке получить доступ к пользовательской программе, защищенной паролем, будет выведен запрос на ввод пароля. Перед доступом к защищенному CPU также можно ввести пароль через PLC → Access Rights (PLC → Права доступа). Сначала выберите соответствующий 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 того, чтобы получить доступ ко всем модулям, защищенным тем же паролем.

Авторизация доступа с паролем действует до завершения последнего S7-приложения.

Каждый, кто знает пароль, имеет неограниченный доступ к пользовательской программе в CPU, несмотря на установленный уровень защиты и положения блокировки.

2.6.3 Информация CPU

В онлайн-режиме доступна информация CPU, приведенная ниже. Задействованы следующие команды меню при выбранном модуле (в режиме онлайн и без проекта) или программе S7 (в онлайн-режиме проектного окна).

- PLC → Diagnose Hardware (PLC → Диагностировать аппаратуру)
Обратитесь к параграфу 2.7.1 «Диагностирование аппаратных средств».

- PLC → Module Information (PLC → Информация о модуле)
Общая информация (версия, например), диагностический буфер, память (текущее отображение рабочей и загрузочной памяти, сжатие), время цикла (длительность последнего, самый продолжительный и самый короткий программный цикл), система отсчета времени (свойства часов CPU, синхронизация часов, счетчик рабочего времени), рабочие параметры (конфигурация памяти, размеры адресных областей, число доступных блоков, SFC и SFB), коммуникация (скорость передачи данных и коммуникационные связи), стеки в состоянии STOP (B-стек, I-стек и L-стек).
- PLC → Operating Mode (PLC → Рабочий режим)
Отображает текущий рабочий режим (к примеру, RUN или STOP), изменение рабочего режима.
- PLC → Clear/Reset (PLC → Очистить/Сбросить)
Очистка (сброс) памяти CPU, перевод его в режим STOP.
- PLC → Set Time and Date (PLC → Установка времени и даты)
Установка внутренних часов CPU.
- PLC → CPU Messages (PLC → Сообщения CPU)
Отчет об асинхронных системных ошибках и об определенных пользователем сообщениях, сгенерированных в программе с помощью SFC 52 WR_USMSG, SFC 18 ALARM_S и SFC 17 ALARM_SQ.
- PLC → Display Force Values (PLC → Отобразить принудительные значения), PLC → Monitor/Modify Variables (PLC → Наблюдение/модифицирование переменных)
(Обратитесь к параграфам 2.7.3 «Наблюдение и модифицирование переменных» и 2.7.4 «Принудительная установка переменных»).

2.6.4 Загрузка пользовательской программы в CPU

При передаче пользовательской программы (скомпилированные блоки и конфигурационные данные) в CPU она грузится в загрузочную память CPU. Физически загрузочная память может быть модулем RAM или флэш-EPROM, и либо встроенной в CPU либо на (дополнительной) карте памяти.

Если используется карта памяти флэш-EPROM, то вы можете записывать на нее в программирующем устройстве и использовать ее в качестве среды данных. Подключение карты памяти к CPU происходит в выключенном состоянии; после включения питания и обнуления памяти соответствующие данные карты памяти передаются в рабочую память CPU. С соответствующим образом оснащенными CPU можно также перезаписать карту флэш-EPROM, если она подключена к CPU, но только записав на нее программу полностью.

В случае загрузочной памяти RAM передача всей программы пользователя осуществляется путем переключения CPU в состояние STOP, выполнения сброса памяти и собственно передачи пользовательской программы. Конфигурационные данные так-

же перемещаются. При обнулении памяти или отключении резервной батареи программа в RAM теряется.

Если вы хотите только изменить конфигурационные данные (свойства CPU, сконфигурированные соединения, GD-коммуникации, параметры модулей и так далее), вам потребуется лишь загрузить объект *System Data* (*Системные данные*) в CPU (выберите объект и передайте его с помощью команды меню PLC → Download - PLC → Загрузить). Параметры CPU вступают в силу немедленно; CPU передает параметры оставшимся модулям во время запуска.

Обратите внимание, что конфигурация полностью загружается в PLC с объектом *System Data* (*Системные данные*). Если вы используете PLC → Download (PLC → Загрузить) в приложении, например, в коммуникациях глобальных данных, передаются только данные, редактируемые приложением.

Замечание: для загрузки сжатого архивного файла выберите PLC → Save Project on Memory Card (PLC → Записать проект на карту памяти) (смотрите параграф 2.2.2 «Управление, перегруппировка, архивирование»). Проект в архивном файле не может быть отредактирован ни с помощью устройства программирования, ни из CPU.

2.6.5 Обработка блоков

Передача блоков

В случае загрузочной памяти RAM в дополнение к передаче всей программы в онлайн-режиме вы можете также модифицировать, удалять или перезагружать отдельные блоки.

Передача отдельных блоков в CPU начинается с их выбора в автономном окне и нажатия PLC → Download (PLC → Загрузить). При одновременно открытых автономном и онлайн-окнах вы можете перетащить блоки из одного окна и поместить их в другое.

Особое внимание требуется при передаче отдельных блоков во время работы. Если в блоке вызываются блоки, отсутствующие в данный момент в памяти, то CPU перейдет в состояние STOP, поэтому сначала вы должны загрузить блоки «нижнего уровня». Это также относится к блокам данных, чьи адреса используются в загруженном блоке. Загрузка блока «высшего уровня» происходит в последнюю очередь. Тогда, после вызова, он будет немедленно исполнен в следующем программном цикле.

SIMATIC-менеджер также предоставляет вам возможность в SCL передать отдельные блоки или программу целиком из автономного контейнера *Blocks* (*Блоки*) в CPU. Нет смысла перемещать их обратно из CPU на жесткий диск, так как скомпилированные блоки не могут редактироваться в редакторе SCL. Вы можете лишь отредактировать исходный файл SCL-программы и, сформировав его, сгенерировать скомпилированные блоки.

Модифицирование блоков в онлайнном режиме

Вы можете пошагово редактировать блоки с помощью STL в онлайнной программе пользователя (в CPU), точно так же, как в автономной пользовательской программе. Однако если онлайнная и автономная обработки данных отличаются, это может привести к тому, что в редакторе нельзя будет отобразить дополнительную информацию об автономной базе данных; в таком случае эти данные могут быть потеряны (символические идентификаторы, метки переходов, комментарии, определенные пользователем типы данных).

Блоки, модифицированные в онлайнном режиме, лучше хранить автономно на жестком диске, чтобы избежать несовместимости данных (например, «конфликт временных меток», при котором интерфейс вызванного блока имеет более позднюю метку времени, чем программа в вызывающем блоке).

Удаление блоков

Если загрузочная память состоит исключительно из RAM, блоки можно модифицировать и удалять. Если пользовательская программа расположена на флэш-EPROM, модули также можете редактировать и удалять, что обеспечивается достаточной емкостью дополнительно доступной RAM. Блоки во флэш-EPROM отмечаются как «неверные» («invalid»). Тем не менее, после сброса памяти или включения питания без резервной батареи блоки вновь передаются из загрузочной памяти флэш-EPROM в рабочую память.

Очистить карту памяти флэш-EPROM можно только в устройстве программирования.

Сжатие

Когда вы загружаете новый или измененный блок в CPU, центральный процессор помещает блок в загрузочную память и передает соответствующие данные в рабочую память. Если блок с таким же номером уже имеется, то данный «старый блок» объявляется неверным (после запроса на подтверждение), а новый блок «добавляется в конец» памяти. Даже удаленный блок «только лишь» объявляется неверным, а не удаляется фактически из памяти.

Результатом этого становятся пробелы в пользовательской памяти, которые все больше и больше уменьшают объем доступной памяти. Эти пробелы могут быть заполнены только с помощью функции *Compress* (*Сжатие*). Когда вы производите сжатие в режиме RUN, исполняемые в текущий момент блоки не перемещаются; истинное сжатие без пробелов может быть достигнуто только в режиме STOP.

Текущее распределение памяти может быть отображено в процентах по команде меню PLC → Module Information (PLC → Информация о модуле), на вкладке *Memory* (*Память*). Диалоговое окно, которое затем появляется, также снабжено кнопкой для предварительного сжатия.

Вы можете инициировать сжатие, зависящее от событий, через программу с вызовом SFC 25 COMPRESS.

Блоки данных в режимах офлайн/онлайн

Адреса данных в блоках данных могут быть назначены *начальному значению* (*initial value*) и *фактическому значению* (*actual value*) (смотрите также параграф 3.4 «Программирование блоков данных»). Если блок данных загружен в CPU, начальные значения передаются загрузочную память, а фактические – в рабочую. Каждое изменение значения по отношению к адресу данных, сделанное через программу, соответствует изменению фактического значения.

При загрузке из CPU блока данных его значения берутся из рабочей памяти, в которой находятся все фактические значения. Вы можете просмотреть фактические значения в момент их считывания по команде View → Data View (Вид → Просмотр данных). Если вы изменяете фактическое значение в блоке данных и записываете блок обратно в CPU, измененное значение помещается в рабочую память.

Когда в качестве загрузочной памяти используется карта памяти флэш-EPROM, блоки, находящиеся на карте памяти, передаются в рабочую память после сброса памяти CPU. Блоки данных сохраняют начальные значения, изначально запрограммированные для них. То же происходит при включении питания без резервной батареи. В S7-300 вы можете защитить отдельные области в блоках данных, объявив ее сохраняемыми.

Блок данных, сгенерированный со свойством UNLINKED (несвязанный), в рабочую память не передается; он остается в загрузочной памяти. Блок данных с этим свойством может быть прочитан только при помощи SFC 20 BLKMOV.

2.7 Тестирование программы

После установки соединения с CPU и загрузки пользовательской программы вы можете протестировать (отладить) программу, как всю целиком, так и частично, к примеру, отдельные блоки. Вы должны инициализировать переменные с использованием сигналов и значений, например, с помощью модулей симулятора, и оценить возвращаемую вашей программой информацию. Если CPU переходит в состояние STOP в результате ошибки, то помощь в поиске источника ошибки может оказать, наряду с другим, информация CPU.

Большие программы отлаживаются по частям. Если, к примеру, вы хотите отладить только один блок, загрузите этот блок в CPU и вызовите его в OB 1. Если OB 1 организован таким образом, что программа может быть отлажена раздел за разделом «от начала до конца», то для отладки вы можете выбрать блоки или разделы программы, используя функции перехода, чтобы пропустить вызовы или программные разделы, которые отладки не требуют.

С помощью дополнительного программного обеспечения PLCSIM можно на устройстве программирования смоделировать CPU и таким образом отладить вашу программу без дополнительных аппаратных средств.

2.7.1 Диагностирование аппаратных средств

В случае сбоя вы можете получить диагностическую информацию о сбойных модулях с помощью функции «Diagnose Hardware» («Диагностика аппаратуры»). Программирующее устройство нужно подключить к шине MPI и запустить SIMATIC-менеджер.

Если в базе данных устройства программирования доступен проект, ассоциированный с конфигурацией предприятия, откройте онлайнное окно проекта командой меню View → Online (Вид → Онлайн). В противном случае нажмите PLC → Display Accessible Nodes (PLC → Отобразить доступные узлы) и выберите CPU. Теперь вы можете получить быстрый доступ к просмотру сбойных модулей с помощью PLC → Diagnose Hardware (PLC → Диагностировать аппаратуру) (по умолчанию). Программа Hardware Configuration во время онлайнного просмотра предоставляет подробную диагностическую информацию о модулях; этот режим может быть установлен в SIMATIC-менеджере на вкладке «View» («Просмотр») после выбора пункта меню Options → Customize (Опции → Настроить).

Информацию о статусе и рабочем состоянии модулей, доступных в онлайнном режиме, вы можете получить в виде просмотра проекта (отображение ошибок, сообщенных станцией), просмотра станции (ошибки, сообщенные модулями) и просмотра модуля (отображение доступной диагностической информации).

2.7.2 Определение причины перехода в состояние STOP

Если CPU из-за возникновения ошибки переходит в состояние STOP, первая предпринимаемая мера для определения причины перехода в STOP – вывод диагностического буфера. CPU помещает в диагностический буфер все сообщения, в том числе причину перехода в STOP и ошибки, приведшие к этому. Чтобы вывести диагностический буфер, переключите PG в режим онлайн, отметьте S7-программу и выберите вкладку *Diagnostic Buffer* (*Диагностический буфер*) с помощью команды меню PLC → Module Information (PLC → Информация о модуле). Последнее событие (оно имеет номер 1) вызвало переход в состояние STOP, например, «STOP because programming error OB not loaded» («STOP, потому что не загружен OB программных ошибок»). Ошибка, приведшая к STOP, описывается в предыдущем сообщении, например, «FC not loaded» («Не загружен FC»). Щелкнув на номере сообщения, вы можете вызвать дополнительные комментарии в следующем расположенном ниже поле дисплея. Если сообщения связаны с программными ошибками в блоке, то с помощью кнопки «Open Block» («Открыть блок») вы можете открыть и отредактировать блок.

Если состояние STOP вызвано, к примеру, программной ошибкой, вы можете выяснить сопутствующие обстоятельства на вкладке *Stacks* (*Стеки*). На открытой вкладке *Stacks* (*Стеки*) вы увидите В-стек (стек блоков), который показывает вам путь вызовов всех незавершенных блоков вплоть до блока, содержащего точку прерывания. Воспользуйтесь кнопкой «I-stack» («I-стек») для того, чтобы отобразить стек прерываний, который покажет вам содержимое регистров CPU (аккумуляторов, адресного регистра, регистра блока данных, слова статуса) в точке прерывания в момент обнаружения ошибки. L-стек (стек локальных данных) показывает временные локальные данные блока, который вы выбрали с помощью мыши в В-стеке.

2.7.3 Наблюдение и модифицирование переменных

Превосходным ресурсом для отладки программ пользователя является наблюдение и модифицирование переменных с помощью таблиц переменных VAT. Могут быть отображены сигнальные состояния или значения переменных простых типов данных. Если у вас есть доступ к пользовательской программе, вы также можете модифицировать переменные, то есть изменить сигнальное состояние или присвоить новые значения.

Внимание: вы должны быть уверены в том, что в результате модифицирования переменных не возникнет опасных состояний.

Создание таблицы переменных

Для наблюдения и изменения переменных вы должны создать таблицу переменных VAT, содержащую переменные и ассоциированные форматы данных. Можно сгенерировать до 255 таблиц переменных (от VAT 1 до VAT 255) и присвоить им имена в таблице символов. Максимальный размер таблицы переменных составляет 1024 строки длиной до 255 знаков (рисунок 2.10).

Вы можете сгенерировать VAT автономно, выбрав *Blocks (Блоки)* пользовательской программы и команду меню Insert → S7 Block → Variable Table (Вставка → Блок S7 → Таблица переменных), также можно сгенерировать VAT без имени в онлайн-режиме путем выбора *S7 Program (S7-программа)* и команды меню PLC → Monitor/Modify Variables (PLC → Наблюдение/модифицирование переменных).

Вы можете определить переменные либо по абсолютным, либо по символическим адресам и выбрать тип данных (формат отображения), с использованием которого переменная будет отображена и модифицирована (по команде меню View → Display Format – Вид → Отобразить формат или щелкнув правой кнопкой мыши прямо на «Display Format» или «Формате отображения»).

Используйте строки комментариев, чтобы дать заглавие определенным разделам таблицы. Также можно задать, какие столбцы должны быть отображены. Вы можете изменить переменную или формат отображения, добавить или удалить строки в любое время. Таблица переменных сохраняется по команде меню Table → Save (Таблица → Сохранить) в контейнере объектов *Blocks (Блоки)*.

	Address	Open pl	Symbol comment	Display format	Status value	Modify value
1	I 28.0	"Contact_1"	Пример глобальной переменной	BOOL	false	
2	I 28.1			BOOL	false	
3	Iw 30			DEC	0	
4	Iw 3			HEX	W#16#0000	
5						

Рисунок 2.10 Пример таблицы переменных

Установка онлайн-соединения

Для работы с таблицей переменных, созданной автономно, переключите ее в онлайн-режим с помощью PLC → Connect To → ... (PLC → Подключить к → ...). Вы должны переключить каждую отдельную VAT в онлайн-режим, и вы можете вновь разъединить соединение по команде PLC → Disconnect (PLC → Отключить).

Условия триггера (пуска)

Находясь в таблице переменных, выберите Variable → Trigger (Переменная → Триггер), чтобы установить триггерную точку и условия триггера отдельно для наблюдения и модифицирования. Триггерная точка – это точка, в которой CPU считывает значения из системной памяти или записывает значения в системную память. Вы

должны определить, как будут осуществлены чтение и запись – один раз или периодически.

Если наблюдение и модифицирование имеют одинаковые условия триггера, то наблюдение выполняется перед модифицированием. Если вы для изменения выбираете триггерную точку «Start of circle» («Начало цикла»), переменные модифицируются после обновления входного образа процесса и перед вызовом ОВ 1. Если для наблюдения вы выбираете триггерную точку «End of circle» («Конец цикла»), значения статуса отображаются после завершения ОВ 1 и перед выводом выходного образа процесса.

Наблюдения переменных

Активируйте функцию наблюдения командой меню Variable → Monitor (Переменная → Наблюдение). Переменные в VAT обновляются в соответствии с заданными условиями триггера. Постоянное наблюдение позволяет отслеживать изменения значений на экране. Значения отображаются в формате данных, установленном вами в столбце «Display format» («Формат отображения»). Клавиша ESC завершает постоянную функцию наблюдения.

Variable → Update Monitor Values (Переменная → Обновить наблюдаемые значения) обновляет отслеживаемые значения один раз и немедленно, несмотря на установленные условия триггера.

Модифицирование переменных

Для передачи заданных значений в CPU, зависящей от условий триггера, воспользуйтесь командой Variable → Modify (Переменная → Модифицирование). Значения вводите только в строках, содержащих переменные, которые вы хотите изменить. Вы можете расширить комментарий для значения при помощи «//» или Variable → Modify Values as comment (Переменная → модифицировать значения как комментарий); эти значения не участвуют в модифицировании. Вы должны определить значения в формате данных, заданном в столбце «Display format» («Формат отображения»).

Изменяются только значения, отображенные при запуске функции модифицирования. Клавиша ESC завершает постоянную функцию модифицирования.

Variable → Activate Modify Values (Переменная → Активировать модифицированные значения) передает измененные значения только один раз и немедленно, несмотря на установленные условия триггера.

2.7.4 Принудительная установка переменных (функция Force)

С помощью соответствующим образом оснащенных CPU вы можете задать фиксированные значения для определенных переменных. Пользовательская программа не

сможет изменить эти значения («принудительная установка»). Принудительная установка допустима в любом режиме работы CPU и выполняется немедленно.

Внимание: вы должны быть уверены в том, что в результате принудительной установки переменных не возникнет опасных состояний.

Стартовой точкой для принудительной установки является таблица переменных (VAT). Создайте VAT, введите адреса для принудительной установки значений и установите соединение с CPU. Выбрав Variable → Display Force Values (Переменные → Отобразить принудительные значения), можно открыть окно, содержащее принудительные значения.

Если в CPU принудительные значения уже активизированы, то они выделены в окне принудительных установок значений жирным шрифтом. Теперь вы можете переместить некоторые или все адреса из таблицы переменных в окно принудительных установок или добавить новые адреса. Сохранение содержимого окна принудительных установок в VAT осуществляется по команде меню Table → Save As (Таблица → Сохранить как).

Принудительные значения допустимы для следующих областей адресов:

- Входы I (образ процесса)
(S7-300 и S7-400)
- Выходы Q (образ процесса)
(S7-300 и S7-400)
- Периферийные входы PI
(только S7-400)
- Периферийные выходы PQ
(S7-300 и S7-400)
- Память меркеров M
(только S7-400)

Задание принудительной установки начинается с Variable → Force (Переменная → Присвоить принудительное значение). CPU принимает принудительные значения и изменений принудительно установленных адресов больше не допускает.

Пока активна функция принудительной установки, выполняется следующее:

- Все обращения для чтения к принудительно заданным адресам через пользовательскую программу (например, загрузка) и через системную программу (например, обновление образа процесса) всегда выдают принудительно установленное значение.
- В S7-400 все обращения для записи к принудительно заданным адресам через пользовательскую программу (например, передача) и через системную программу (например, через SFC – последовательные функциональные схемы) остаются

безрезультатными. В S7-300 пользовательская программа может переписать установленные принудительно значения.

Принудительная установка в S7-300 аналогична циклическому модифицированию: после обновления образа входов процесса CPU переписывает входы принудительными значениями; перед выводом образа выходов процесса CPU переписывает выходы принудительными значениями.

Примечание: принудительная установка не завершается закрытием окна принудительных значений или таблицы переменных или прерыванием соединения с CPU! Удалить задание принудительной установки можно только с помощью Variable → Delete Force (Переменная → Снять принудительную установку).

Принудительная установка также снимается путем сброса памяти или в случае сбоя в электропитании, при условии, что CPU не снабжено резервной батареей. По завершению принудительной установки адреса сохраняют принудительные значения до их перезаписи либо программой пользователя, либо системной программой.

Принудительная установка эффективна только на входах/выходах, назначенных CPU. Если после рестарта установленные принудительно периферийные входы и выходы более не назначены (к примеру, в результате повторной параметризации), то соответствующие периферийные входы и выходы не будут принудительно заданы.

Обработка ошибок

Если размер области, к которой обращаются при чтении, больше размера области, устанавливаемой принудительно (например, принудительно установленный байт в слове), неустанавливаемый компонент значения адреса читается как обычно. Если здесь возникает ошибка синхронизации (ошибка доступа или размера области), то считывается «значение, заменяющее ошибочное» («error substitute value»), определяемое программой пользователя или центральным процессором, или CPU переходит в состояние STOP.

Если при записи размер области, к которой происходит обращение, больше размера устанавливаемой принудительно области (например, принудительно установленный байт в слове), неустанавливаемый компонент значения адреса записывается как обычно. Ошибочный доступ при записи оставляет принудительно устанавливаемый компонент адреса без изменений, то есть ошибка синхронизации не отменяет защиты от записи.

Загрузка принудительно установленных периферийных входов выводит принудительно установленное значение. Если размер области, к которой происходит обращение, согласуется с устанавливаемой принудительно областью, то модули входов, в которых произошел сбой, или которые (еще) не подключены, могут быть «заменены» принудительным значением.

Вход I в образе процесса, который принадлежит принудительно установленному периферийному входу PI, принудительно не устанавливается; он предварительно не

назначается и все еще доступен для перезаписи. Когда обновляется образ процесса, вход получает принудительное значение периферийного входа.

Когда принудительно устанавливаются периферийные выходы PQ, соответствующие выходы Q в образе процесса не обновляются и принудительно не устанавливаются (принудительная установка действует «внешне» только на выходы модулей). Выходы Q сохраняются и доступны для перезаписи; чтение выходов выводит записанное значение (не принудительное значение). Если модуль выхода принудительно установлен и если в нем возникает сбой или он удаляется, то он вновь немедленно получит принудительное значение при переподключении.

Модули выходов выдают сигнальное состояние «0» или заменяют значение на сигнал OD (блокировка модулей выходов в режиме STOP, HOLD или RESTART – останов, ожидания или рестарта) – даже если периферийные выходы принудительно установлены (исключение: аналоговые модули без определения OD продолжают выдавать принудительное значение). Если сигнал OD деактивирован, то принудительное значение вновь становится действующим.

Если в состоянии STOP активирована функция *Enable PQ (Активация PQ)*, принудительное значение в режиме STOP также становится действующим (благодаря деактивации сигнала OD). По завершении *Enable PQ (Активация PQ)* происходит обратная установка модулей в «безопасное» («safe») состояние (сигнальное состояние «0» или заменяемое значение); принудительное значение вновь становится действующим на переходе к режиму RUN.

2.7.5 Разблокировка периферийных выходов

В режиме STOP обычно модули выходов деактивированы сигналом OD; с помощью функции разблокировки периферийных выходов вы можете заблокировать сигнал OD и, таким образом, изменять модули выходов, даже если CPU находится в состоянии STOP. Модифицирование осуществляется посредством таблицы переменных. Изменяются только периферийные выходы, назначенные CPU. Возможное применение: тест монтажных соединений выхода в режиме STOP и без программы пользователя.

Внимание: вы должны быть уверены, что разблокировка периферийных выходов не вызовет опасных состояний!

Создайте таблицу переменных и введите периферийные выходы (PQ) и изменяющиеся значения. Переключите таблицу переменных в режим онлайн по команде меню PLC → Connect To → ... (PLC → Подключить к → ...), и, если требуется, остановите CPU, например, командой PLC → Operating Mode (PLC → Рабочий режим) и «STOP».

Блокировка сигнала OD осуществляется выбором пункта меню Variable → Enable Peripheral Outputs (Переменная → Разблокировать периферийные выходы); выходы модуля теперь имеют сигнальное состояние «0», заменяемое значение или принудительное значение. Модифицировать периферийные выходы вы можете с помощью

Variable → Activate Modify Values (Переменная → Активировать модифицированные значения). Вы можете сменить изменяющееся значение и модифицировать снова.

Отключить функцию можно повторным выбором пункта меню Variable → Enable Peripheral Outputs (Переменная → Разблокировать периферийные выходы) или нажатием клавиши ESC. После этого сигнал OD будет разблокирован, выходы модуля установятся в «0», а заменяемое значение или принудительное значение сбрасывается.

Если при активизированной «разблокировке периферийных выходов» действует режим STOP, то все периферийные выходы сняты (удалены), сигнал OD активизирован на переходе к RESTART и деактивирован на переходе к RUN.

2.7.6 Статус программы LAD/FBD

С помощью функции *Program status* (Состояние программы) программный редактор предоставляет дополнительный метод тестирования программы пользователя. Редактор покажет поток двоичного сигнала и цифровых значений в сети (network).

Блок, чью программу требуется отладить, находится в пользовательской памяти CPU, вызывается и редактируется там. Откройте этот блок, к примеру, дважды щелкнув на нем в онлайнном окне SIMATIC-менеджера. Откроется редактор и отобразит программу в блоке.

Отметьте сеть (network), которую вы намереваетесь отладить. Активируйте функцию состояния программы по команде меню Debug → Monitor (Отладка → Наблюдение). Теперь в окне блока можно увидеть поток двоичного сигнала и следить за его изменением. Параметры отображения в программном редакторе (к примеру, цвет) настраиваются на вкладке «LAD/FBD» после вызова Options → Customize (Опции → Настроить). Деактивировать функцию состояния программы можно, вновь нажав Debug → Monitor (Отладка → Наблюдение).

Вы можете установить условия триггера, выбрав команду Debug → Call Environment (Отладка → Параметры вызова). Данная установка потребуется, если отлаживаемый блок вызывается в вашей программе более одного раза. Можно инициировать запись состояния либо путем определения порядка вызовов, либо поставив ее в зависимость от открытого блока данных. Если блок вызывается только один раз, отметьте «No condition» («Без условий»).

Функция состояния программы позволяет модифицировать переменные. Отметьте модифицируемый адрес и выберите Debug → Modify Address (Отладка → Модифицирование адреса).

Для записи информации по статусу программы требуется дополнительное рабочее время в программном цикле. По этой причине для целей отладки вы можете выбрать два рабочих режима: режим отладки и режим процесса. В *режиме отладки* (debug mode) все функции отладки могут использоваться без ограничения. Данный режим можно избрать, например, для отладки блоков без подключения к системе, так как это может значительно увеличить время исполнения цикла. В режиме обработки (process mode) внимание уделяется сведению к минимуму увеличения времени цик-

ла, что приведет к возникновению ограничений отладки, например, в программных циклах (отображаются не все циклические проходы).

В CPU, оснащенных соответствующим образом, установить рабочий режим вы можете во время параметризации CPU на вкладке «Protection» («Защита»). Если режим отладки выбран во время параметризации CPU, то изменить его вы сможете только путем повторной параметризации. В противном случае он может быть изменен в отображаемом диалоговом окне. Установленный режим работы отображается по команде меню Debug → Operation (Отладка → Операция).

Глава 3

Программа SIMATIC S7

Содержание главы 3

3	<u>Программа SIMATIC S7</u>	4
3.1	<u>Обработка программы</u>	4
3.1.1	<u>Методы обработки программы</u>	4
3.1.2	<u>Приоритетные классы</u>	6
3.1.3	<u>Спецификации для обработки программы</u>	8
3.2	<u>Блоки</u>	10
3.2.1	<u>Типы блоков</u>	10
3.2.2	<u>Структура блоков</u>	12
3.2.3	<u>Свойства блоков</u>	14
3.2.4	<u>Интерфейс блоков</u>	17
3.3	<u>Программирование кодовых блоков</u>	21
3.3.1	<u>Генерирование блоков</u>	21
3.3.2	<u>Редактирование элементов LAD</u>	28
3.3.3	<u>Редактирование элементов FBD</u>	31
3.4	<u>Программирование блоков данных</u>	35
3.4.1	<u>Создание блоков</u>	35
3.4.2	<u>Типы блоков данных</u>	35
3.4.3	<u>Окно блока</u>	36
3.5	<u>Переменные, константы и типы данных</u>	38
3.5.1	<u>Общие замечания о переменных</u>	38
3.5.2	<u>Адресация переменных</u>	39
3.5.3	<u>Обзор типов данных</u>	43
3.5.4	<u>Простые типы данных</u>	44
3.5.5	<u>Сложные типы данных</u>	52
3.5.6	<u>Параметрические типы</u>	57
3.5.7	<u>Пользовательские типы данных</u>	57

3 Программа SIMATIC S7

В этой главе рассказывается о структуре пользовательской программы для CPU SIMATIC S7-300/400, начиная с различных приоритетных классов (типов исполнения программы), по ходу обсуждения затрагивая разделы компонентов программы пользователя (блоков) и заканчивая переменными и типами данных. В данной главе основное внимание уделяется описанию программирования блоков средствами LAD и FBD. Также приведено описание типов данных.

Определение структуры пользовательской программы происходит в фазе проектирования при согласовании технологических и функциональных условий; это имеет решающее значение для процесса создания программы, тестирования программы и запуска. Для эффективного программирования необходимо уделять особое внимание структуре программы.

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 описаний, которые осуществляют управление всеми системными ресурсами и процессами, использующими эти ресурсы. Она включает в себя такие функции, как резервирование данных в случае сбоя электропитания, активация приоритетных классов и так далее. Операционная система является компонентом CPU, к которому у пользователя нет доступа в режиме записи. Однако, вы можете перезагружать операционную систему с карты памяти в случае, к примеру, обновления программы.

Пользовательская программа представляет собой совокупность всех инструкций и описаний для обработки сигналов, с помощью которых осуществляется управление предприятием (процессом) в соответствии с определенной задачей автоматизации.

3.1.1 Методы обработки программы

Пользовательская программа может состоять из программных разделов, которые обрабатываются CPU в зависимости от определенных событий. Таким событием может быть запуск системы автоматизации, прерывание или обнаружение программной ошибки (рисунок 3.1). Программы, назначенные для обработки событий, разделяются на *приоритетные классы (priority classes)*, которые определяют порядок обработки программы (система взаимных прерываний – mutual interruptibility), когда происходит несколько событий.

Программой с низшим приоритетом является *главная программа (main program)*, циклически обрабатываемая CPU. События могут прервать главную программу в любом месте, после чего CPU выполнит связанную с прерыванием обслуживающую

программу (процедуру) или программу (процедуру) обработки ошибки и вернет управление главной программе.

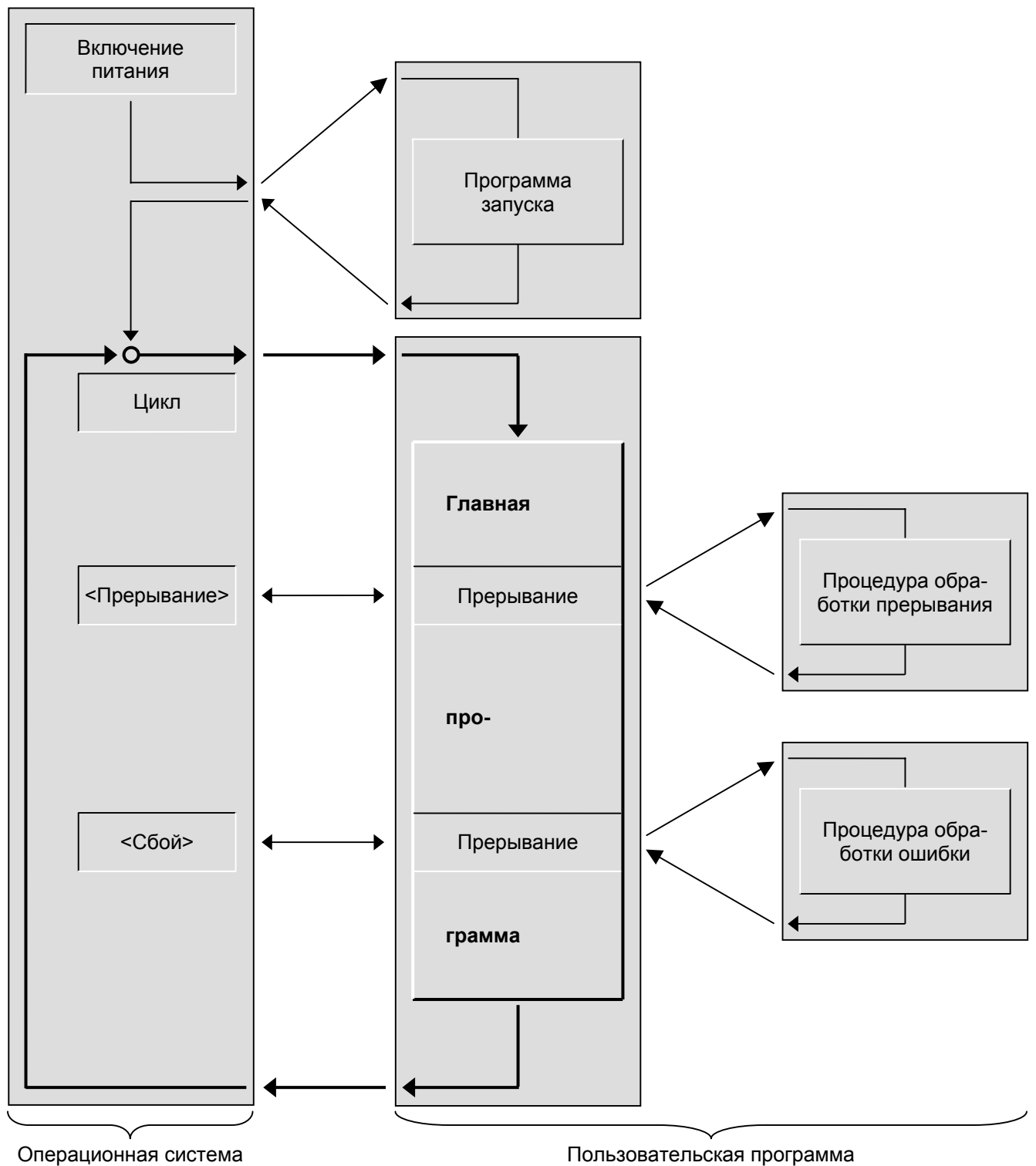


Рисунок 3.1 Методы обработки пользовательской программы

Каждому событию соответствует специальный *организационный блок (organization block – OB)*. Организационные блоки в программе пользователя реализуют механизм приоритетных классов. При возникновении события CPU активизирует назначенный организационный блок. Организационный блок – это часть пользовательской программы, которую вы можете сами написать.

Перед началом обработки главной программы CPU выполняет программу запуска (startup routine). Эта программа может быть запущена включением питания, поворотом переключателя режимов на передней панели CPU или с помощью программирующего устройства. Обработка программы, следующая за исполнением программы запуска, в системах S7-300 начинается всегда с начала главной программы (полный рестарт – complete restart); в системах S7-400 также возможно продолжить сканирование программы с точки, в которой оно было прервано («теплый» рестарт – warm restart).

Главная программа располагается в организационном блоке OB 1, который всегда обрабатывается центральным процессором. Начало пользовательской программы идентично первому сегменту (сети, network) в OB 1. По завершению обработки OB 1 (конец программы) CPU передает управление операционной системе, и после вызова различных функций операционной системы, таких как обновление образа процесса, центральный процессор снова вызывает OB 1.

Событиями, которые могут вмешиваться в работу программы, являются прерывания (interrupts) и ошибки (errors).

Источником прерываний может быть процесс (аппаратные прерывания), или они могут исходить от CPU (циклические прерывания – watchdog interrupts, прерывания по времени суток – time-of-day interrupts и другие).

Что касается ошибок, то различают синхронные и асинхронные ошибки. Асинхронной является ошибка, которая не зависит от выполнения программы, к примеру, отказ электропитания в устройстве расширения или замена модуля. Синхронные ошибки возникают при выполнении программы. К ним относятся, например, обращение к несуществующему адресу или ошибка преобразования типа данных.

Типы и номера регистрируемых событий и соответствующих организационных блоков определяются CPU; не каждый CPU способен обработать все возможные события STEP 7.

3.1.2 Приоритетные классы

В таблице 3.1 приведены доступные организационные блоки SIMATIC S7 с их приоритетами. В некоторых приоритетных классах можно изменять заданный при параметризации CPU приоритет. В таблице показаны возможные низший и наивысший приоритетные классы; для каждого CPU определены свои верхний и нижний приоритеты; в данном обзоре представлены отдельные типы CPU.

Таблица 3.1 Организационные блоки SIMATIC S7

Организационный блок	Вызывается	Приоритет	
		По умолчанию	Возможные изменения
ОВ 1 свободного цикла	Циклически операционной системой	1	Нет
TOD-прерывания ОВ 10 ... ОВ 17	В определенное время суток или через равные промежутки времени (например, ежемесячно)	2	2 ... 24
Прерывания с задержкой времени ОВ 20 ... ОВ 23	По истечении запрограммированного времени, управляется из пользовательской программы	3 ... 6	2 ... 24
Циклические прерывания ОВ 30 ... ОВ 38	Регулярно через запрограммированные интервалы времени (например, каждые 100 мс)	7 ... 15	2 ... 24
Прерывания процесса ОВ 40 ... ОВ 47	По сигналу прерывания от I/O-модуля (модуля входа/выхода)	16 ... 23	2 ... 24
Мультипроцессорное прерывание ОВ 60	Пользовательской программой при возникновении события в мультипроцессорном режиме	25	Нет
Ошибки резервирования ОВ 70 ОВ 72 ОВ 73	В случае потери резервирования из-за I/O-ошибок В случае ошибки резервирования CPU В случае ошибки резервирования коммуникаций	25 28 25	2 ... 26 2 ... 28 2 ... 26
Асинхронные ошибки ОВ 80, ОВ81...ОВ84,86,87 ОВ 85	В случае ошибок, не связанных с выполнением программы (например, ошибка времени – time error, SE-ошибка, диагностическое прерывание, прерывание вставки/удаления модуля, сбой стойки/станции)	26 ²⁾ 26 ²⁾ 26 ²⁾	26 2 ... 26 24 ... 26
Фоновая обработка ОВ 90	Минимальное время продолжительности цикла еще не достигнуто	29 ¹⁾	Нет
Подпрограмма запуска ОВ100, ОВ101, ОВ102	При запуске программируемого контроллера	27	Нет
Синхронные ошибки ОВ 121, ОВ 122	В случае ошибок, связанных с выполнением программы (например, ошибка доступа к I/O)	Приоритет ОВ, вызвавшего ошибку	

¹⁾ см. текст ²⁾ при запуске: 28

Организационный блок ОВ 90 (фоновая обработка) может выполняться вместо ОВ 1 и, как ОВ 1, может быть прерван любыми прерываниями и ошибками.

Процедура запуска может быть расположена в организационном блоке ОВ 100 (полный рестарт) и в ОВ 101 («теплый» рестарт); она имеет приоритет 27. Асинхронные ошибки, возникающие в подпрограмме запуска, имеют приоритетный класс 28. Диагностические прерывания рассматриваются как асинхронные ошибки.

Какие из доступных приоритетных классов будут использоваться – определяется при параметризации CPU. Недействующим приоритетным классам (организацион-

ным блокам) должен быть присвоен нулевой приоритет. Для всех используемых приоритетных классов программируются соответствующие организационные блоки; в противном случае CPU вызовет OB 85 («Program Processing Error» - «Ошибка выполнения программы») или перейдет в состояние STOP.

Для каждого выбранного приоритетного класса должен быть выделен достаточный объем памяти для временных локальных данных (L-стек). Более подробно об этом говорится в параграфе 18.1.5 «Временные локальные данные».

3.1.3 Спецификации для обработки программы

Операционная система CPU обычно использует параметры, установленные по умолчанию. Вы можете изменить эти установки при параметризации CPU (в объекте *Hardware (Аппаратура)*), чтобы настроить систему для соответствия вашим индивидуальным требованиям. Изменить параметры можно в любое время.

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

- **Startup (Параметры запуска)**
Определяет тип запуска («холодный» рестарт, полный рестарт, «теплый» рестарт); наблюдение сигналов Ready («Готов») или параметризация модуля; максимальная продолжительность времени перед «теплым» рестартом (максимальное время между остановом CPU и рестартом).
- **Cycle/Clock Memory (Цикл/Тактовый меркер, синхробайт)**
Включает/выключает циклическое обновление образа процесса; определение продолжительности наблюдения цикла и минимальной длительности цикла; продолжительность времени цикла для коммуникаций в процентах; количество тактовых меркеров; размер образов 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 (организационных блоках); максимальный размер L-стека и число коммуникационных заданий.
- **Interrupts (Прерывания)**
Определение приоритета аппаратных прерываний, прерываний с задержкой времени, асинхронных ошибок и (в скором времени доступных) коммуникационных прерываний.
- **Time-of-Day Interrupts (Прерывания по времени суток)**
Спецификация приоритета, определение времени запуска и периодичности.

- Cyclic Interrupts (Циклические прерывания)
Определение приоритета, времени цикла и сдвига фазы.
- Diagnostic/Clock (Диагностика/Системные часы)
Индикация причины перехода в состояние STOP; тип и интервал синхронизации времени; коэффициент коррекции.
- Protection (Защита)
Определение уровня защиты; задание пароля.
- Multicomputing (Параметры мультипроцессорного режима)
Определение числа CPU.
- Integrated I/O (Интегрированные входы/выходы)
Активирование и параметризация интегрированных входов/выходов.

CPU при запуске производит смену параметров по умолчанию на параметры, заданные пользователем, которые остаются в силе до их замены.

3.2 Блоки

С целью повышения удобочитаемости и понимания программы вы можете разбить ее на произвольное число разделов. Языки программирования STEP 7 поддерживают эту концепцию и предоставляют необходимые функции. Каждая часть программы должна быть независимой и обладать технологическим или функциональным базисом. Эти разделы программы называются «блоками» («Blocks»). Блок – это раздел программы, который определяется собственной функциональностью, структурой или решаемой задачей.

3.2.1 Типы блоков

Язык программирования STL предоставляет для разных задач различные типы блоков:

- Пользовательские блоки (user blocks)
Эти блоки содержат пользовательскую программу и пользовательские данные.
- Системные блоки (system blocks)
Эти блоки содержат системную программу и системные данные.
- Стандартные блоки (standard blocks)
Готовые к непосредственному использованию (созданные заранее) блоки, такие как драйверы для функциональных модулей (FM) и коммуникационных процессоров (CP).

Пользовательские блоки

В случае больших и сложных программ рекомендуется и отчасти является необходимостью «структурирование» (разбиение) программы с выделением блоков. В зависимости от приложения можно выбрать для использования различные типы блоков:

Организационные блоки (Organization blocks - OB)

Эти блоки служат в качестве интерфейса между операционной системой и программой пользователя. Операционная система CPU вызывает организационные блоки при возникновении определенных событий, например, в случае аппаратного прерывания или прерывания по времени суток. Главная программа находится в организационном блоке OB 1. Остальные организационные блоки имеют постоянные номера, назначенные в зависимости от событий, для обработки которых они вызываются.

Функциональные блоки (*Function blocks - FB*)

Эти блоки являются частями программы, вызовы которых могут быть запрограммированы с помощью параметров блока. У них есть область памяти для переменных (*variable memory*), которая расположена в блоке данных. Этот блок постоянно назначен функциональному блоку или, точнее, *вызову (call)* функционального блока. Кроме того, каждому вызову функционального блока можно назначить другой блок данных (с такой же структурой данных, но содержащий другие значения). Подобный постоянно назначенный блок называется экземплярным блоком данных или экземпляром блока данных (*instance data block*), а совокупность вызова функционального блока и экземплярного блока данных называется экземпляром вызова (*call instance*) или просто «экземпляром» («*instance*»). Функциональные блоки могут также хранить свои переменные в экземплярном блоке данных вызывающего функционального блока; такой экземпляр называется «локальным экземпляром» («*local instance*»).

Функции (*Functions - FC*)

Функции используются для программирования часто повторяющихся или сложных функций автоматизации. Для них могут быть назначены параметры. Функции могут возвращать значение (называемое значением функции) в вызывающий блок. Значение функции является необязательным параметром. Кроме него у функций могут быть другие выходные параметры. Функции не сохраняют информацию и не имеют назначенного блока данных.

Блоки данных (*Data blocks - DB*)

Эти блоки содержат данные вашей программы. Программируя блоки данных, вы определяете форму хранения данных (в каком блоке, в каком порядке и какой при этом используется тип данных). Блоки данных используются двумя способами:

- 1) в качестве глобальных блоков данных (*global data blocks*),
- 2) в качестве экземплярных блоков данных (*instance data blocks*).

Глобальный блок данных в пользовательской программе является, так сказать, «свободным» блоком данных и не назначается кодовому блоку. Однако, экземплярный блок данных назначен функциональному блоку и хранит часть локальных данных этого блока.

Количество блоков определенного блочного типа и размер блоков зависит от типа CPU. Число организационных блоков и их номера фиксированы; они назначаются операционной системой CPU. Блокам других типов вы можете самостоятельно назначить номера из определенного диапазона. Также вы можете с помощью таблицы символов назначить каждому блоку имя (символ) и затем обращаться к блокам по присвоенному имени.

Системные блоки

Системные блоки 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 вам доступ к важным системным функциям, таким как управление внутренними таймерами CPU или различные коммуникационные функции.

Вы можете вызвать SFC и SFB, но вы не сможете ни изменить их, ни самостоятельно запрограммировать. Сами блоки не занимают места в пользовательской памяти (user memory); только вызовы блоков и экземплярные блоки данных блоков SFB располагаются в пользовательской памяти.

Блоки SDB содержат информацию о таких вещах, как конфигурация системы автоматизации или параметры модулей. Система STEP 7 сама генерирует эти блоки и управляет ими. Тем не менее, вы можете определять их содержимое, например, при конфигурировании станций. Как правило, блоки SDB располагаются в загрузочной памяти (load memory). Из пользовательской программы доступ к ним получить нельзя.

Стандартные блоки

В дополнение к создаваемым вами функциям и функциональным блокам вы можете использовать готовые к применению блоки (называемые «стандартными блоками»). Они могут поставляться на носителях данных или содержаться в библиотеках, входящих в состав пакета STEP 7 (например, IEC-функции или функции для преобразования S5/S7).

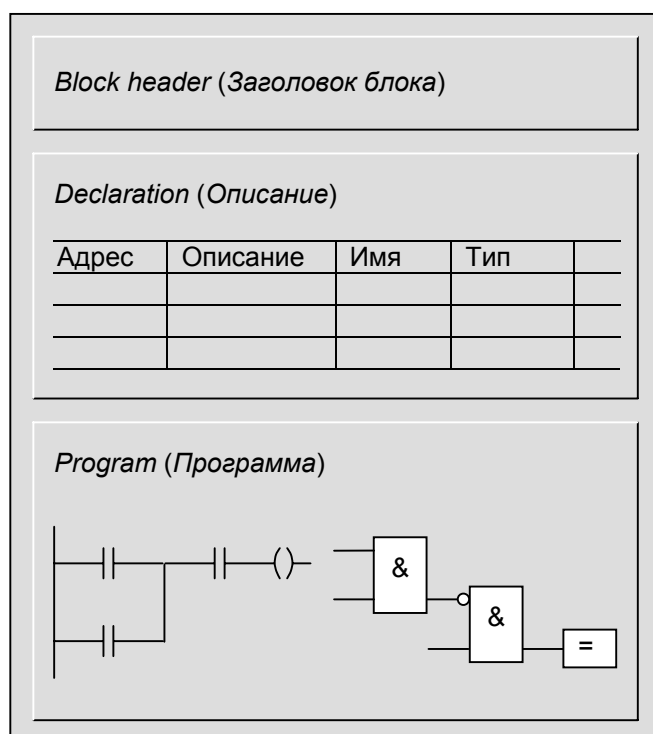
Глава 25 «Библиотеки блоков» содержит обзор стандартных блоков, предоставляемых *Стандартной библиотекой (Standard Library)*.

3.2.2 Структура блоков

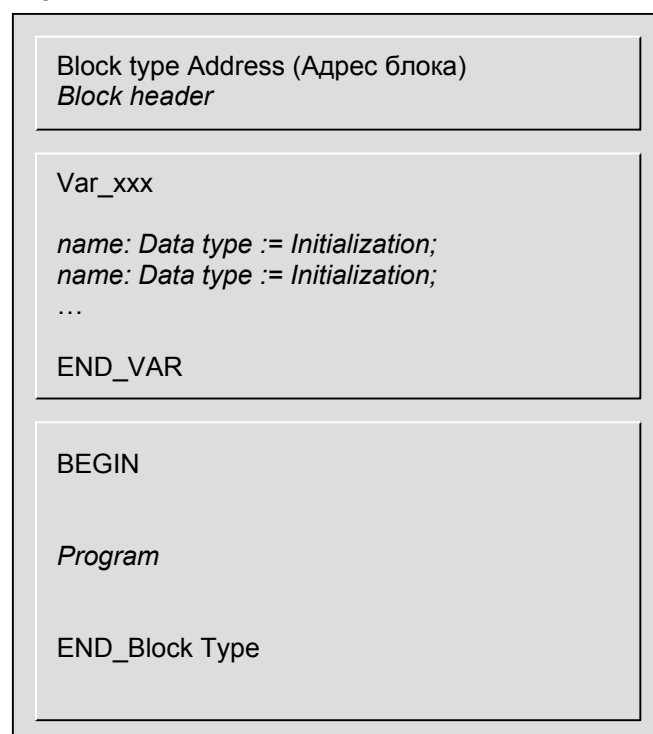
По существу кодовые блоки (code blocks) состоят из трех частей (рисунок 3.2):

- Заголовок блока (block header), который содержит свойства блока, например, имя блока;
- Раздел описаний (объявлений) (declaration section), в котором описаны (определены) локальные переменные блока (внутриблочные переменные);
- Раздел программы (program section), который содержит программу и комментарии к ней.

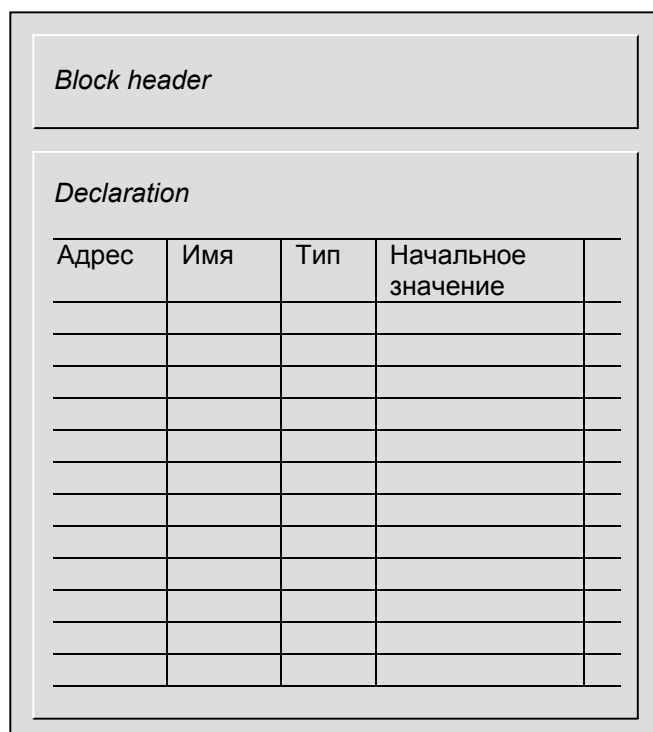
Логический блок, пошаговое программирование



Логический блок, программирование, ориентированное на создание исходных текстов



Блок данных, пошаговое программирование



Блок данных, программирование, ориентированное на создание исходных текстов

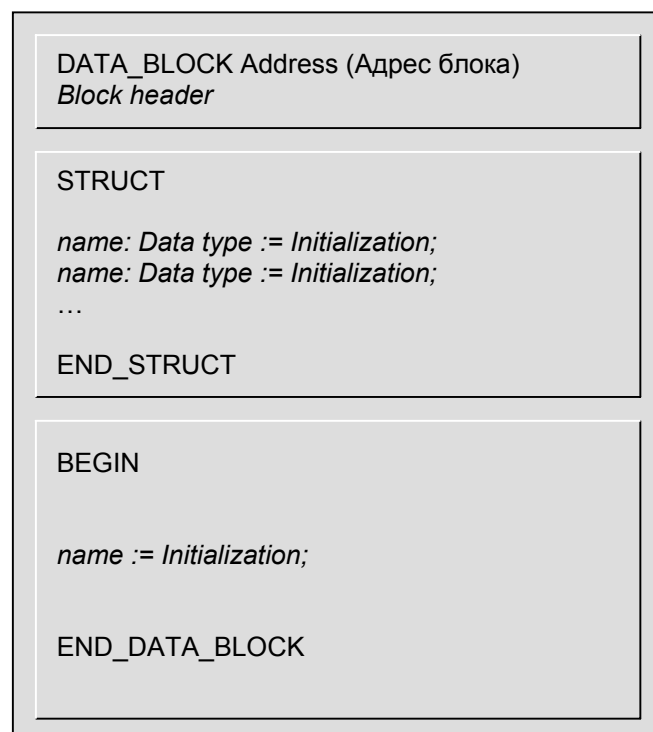


Рисунок 3.2 Структура блока

Блоки данных имеют похожую структуру:

- Заголовок блока (block header) содержит свойства блока;
- Раздел описаний (объявлений) (declaration section) содержит определения локальных переменных блока, в данном случае с адресами данных указываются типы данных;
- Раздел инициализации (initialization section), в котором для отдельных адресов данных могут быть определены начальные значения.

В случае пошагового программирования раздел описаний и раздел инициализации объединены. Вы можете определить адреса данных и их типы в окне просмотра описаний («declaration view») и можете инициализировать отдельно каждый адрес данных в окне просмотра данных («data view») (смотрите ниже).

3.2.3 Свойства блоков

Свойства блоков или атрибуты содержатся в заголовке блока. Вы можете просмотреть и изменить атрибуты блока в редакторе с помощью команды меню File → Properties (Файл → Свойства) (рисунок 3.3).

На вкладке «General – Part 2» («Общие – Часть 2») показано распределение памяти для блока в байтах:

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

Атрибут *KNOW HOW Protection* (Защита ноу-хау) используется для защиты блока. Если этот атрибут блока установлен, программу в таком блоке нельзя просмотреть, распечатать или изменить. Редактор покажет вам только заголовок блока и таблицу описания (объявления) переменных (declaration table) с параметрами блока. При вводе текста в исходный файл вы сами можете защитить каждый блок с помощью ключевого слова `KNOW_HOW_PROTECT`. Если вы установили защиту блока, то никто, даже вы, не сможет просмотреть откомпилированную версию данного блока (сохраните исходный файл в безопасном месте!).

Заголовок любого стандартного блока, поставляемого Siemens, содержит атрибут «*Standard Block*» («Стандартный блок»).

Атрибут «*DB is write-protected in the PLC*» («DB в PLC защищен от записи») используется только для блоков данных. Установка его означает, что вы сможете только считывать из этого блока данных в вашей программе. При попытке записи в такой

блок данных выведется сообщение об ошибке. Это свойство защиты от записи не следует путать с защитой блока. Блок данных с защитой блока может быть считан и записан в программе пользователя, но его данные не будут доступны для просмотра на программаторе или устройстве наблюдения оператора.

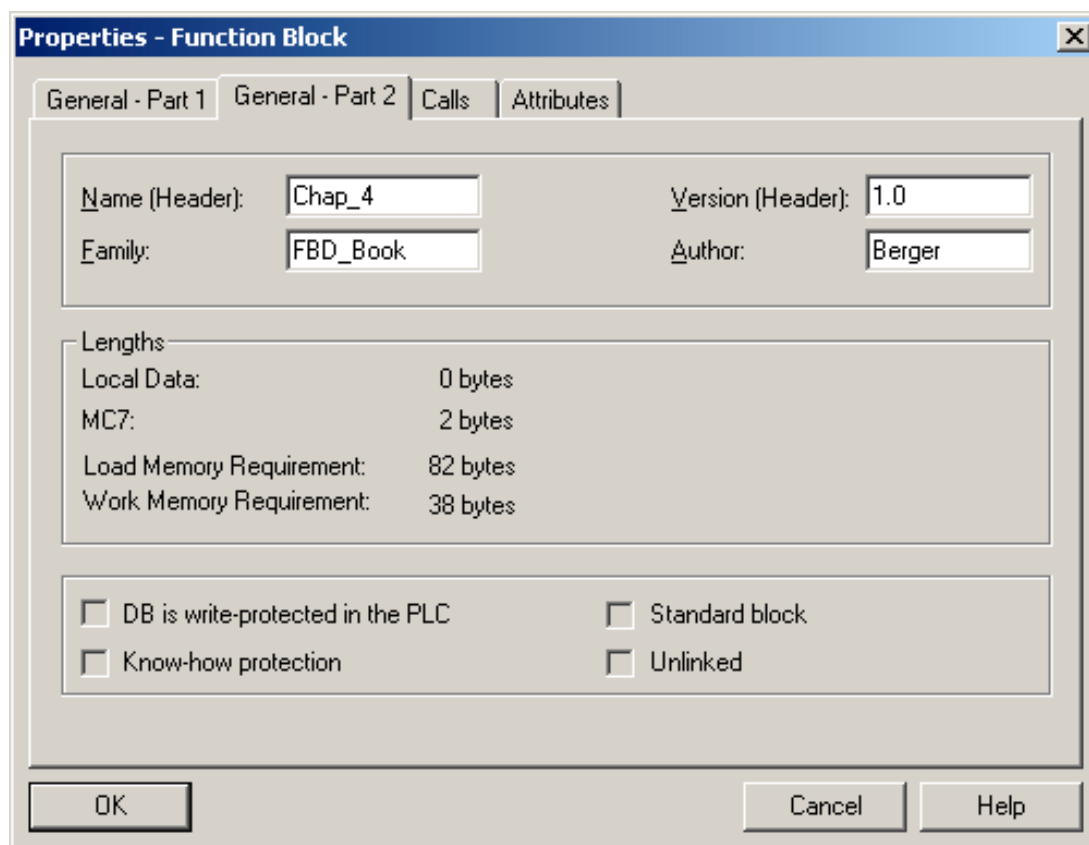


Рисунок 3.3 Свойства блока

Блок данных, имеющий атрибут *Unlinked* (*Неприсоединенный*) располагается только в загрузочной памяти; он не относится к исполняемым. Записывать в блоки данных, находящиеся в загрузочной памяти, нельзя, а чтение их можно произвести только с помощью системной функции SFC 20 BLKMOV.

Ниже приведены остальные спецификации вкладки «General – Part 2» («Общие – Часть 2»).

Атрибут *Name* (*Имя*) идентифицирует блок; это не то же самое, что и символическое имя. Различные блоки могут иметь одинаковые имена.

С помощью атрибута *Family* (*Семейство*) можно назначить общее свойство группе блоков. Имя блока и принадлежность к семейству отображаются при вставке блоков, и когда осуществляется выбор блоков в диалоговом окне каталога программных элементов (program elements catalog).

Атрибут *Author* (*Автор*) указывает на создателя блока.

Последние три атрибута могут состоять из восьми символов. Допускаются буквы, цифры и знак подчеркивания.

Атрибут *Version* (*Версия*) вводится дважды с использованием чисел от 0 до 15.

На вкладке «General – Part 1» («Общие – Часть 1») редактор записывает дату модификации блока в две временные отметки (*time stamps*): для программного кода и для интерфейса, то есть для параметров блока и статических локальных данных. Обратите внимание на то, что дата модификации интерфейса должна быть той же или более ранней, чем дата изменения программного кода в вызывающем блоке. В противном случае при выводе на экран вызывающего блока редактор выдаст сообщение о конфликте меток времени («*time stamp conflict*»).

Блоки могут быть созданы или скомпилированы согласно версии 1 или версии 2. Это имеет практическое значение только для функциональных блоков. Если активировано свойство «*multi-instance capability*» («возможность мультиэкземпляльности, несколько экземпляров DB для функционального блока»), как это обычно и происходит, то мы имеем дело с блоком версии 2. Если свойство «*multi-instance capability*» выключено, то вы не сможете самостоятельно вызвать блок как локальный экземпляр, вызвать другой функциональный блок из данного блока в качестве локального экземпляра также не представляется возможным. Преимущество блока версии 1 заключается в ограничении использования экземпляра блока данных в случае косвенной адресации (имеет значение только при программировании на STL).

На вкладке «Calls» («Вызовы») вы можете увидеть список всех блоков, вызываемых в данном блоке, с отметками времени для кода и интерфейса.

Блоки могут иметь системные атрибуты. На вкладке «Attributes» («Атрибуты») представлены такие атрибуты, которые управляют и координируют функции разных приложений, к примеру, в системе управления SIMATIC PCS7.

Размер программы, требуемый объем памяти

Требуемый объем памяти для скомпилированного блока отображается в свойствах блока. Выберите блок в SIMATIC-менеджере, вызовите команду меню Edit → Object Properties (Правка → Свойства объекта) и в появившемся окне выберите вкладку «General – Part 2» – «Общие – Часть 2». Вы увидите требуемый объем загрузочной и рабочей памяти для данного блока.

Длина пользовательской программы занесена в свойства (Properties) автономного контейнера *Blocks* (Блоки). Выберите *Blocks* (Блоки) и затем команду меню Edit → Object Properties (Правка → Свойства объекта). На вкладке «Blocks» («Блоки») вы найдете информацию «Size in work memory» («Размер области в рабочей памяти») и «Size in load memory» («Размер области в загрузочной памяти»).

Примите во внимание то, что конфигурационные данные (системные блоки данных) не учитываются при указании размера программы в загрузочной памяти. Открыв контейнер *Blocks* (Блоки), вы можете увидеть требования к загрузочной памяти для системных данных в деталях (представленных в виде таблицы). В своей строке состояния SIMATIC-менеджер указывает суммарный объем памяти для всех блоков, которые вы выберете с нажатой клавишей Ctrl.

С помощью программатора, подключенного интерактивно (в онлайн-режиме), и SIMATIC-менеджера вы можете просмотреть текущее назначение памяти CPU на вкладке «Memory» («Память»), предварительно выбрав опцию меню PLC → Module Information (PLC → информация о модуле).

Контрольная сумма (Checksum)

Редактор программ (Program Editor) генерирует контрольную сумму для всех блоков пользовательской программы и сохраняет ее в объектных свойствах контейнера *Blocks* (Блоки). Идентичные программы имеют одинаковую контрольную сумму, при каждом изменении программы меняется и контрольная сумма. Контрольная сумма генерируется также для системных данных. Вы можете найти контрольные суммы, выбрав контейнер *Blocks* (Блоки) и опцию меню Edit → Object Properties (Правка → Свойства объекта).

3.2.4 Интерфейс блоков

Таблица описания переменных содержит интерфейс блока (block interface) с остальной программой. Он состоит из параметров блока (вход, выход и входные и выходные параметры), а также – в случае функциональных блоков – статических локальных данных. Временные локальные данные не входят в интерфейс блока. Интерфейс блока определяется в таблице описания переменных и инициализируется вместе с ними при вызове блока (обратитесь к главе 19 «Параметры блоков»).

Программный редактор проверяет соответствие инициализации параметров вызываемого блока и интерфейса вызываемого блока. Для этого редактор использует метки времени: интерфейс вызываемого блока должен быть старше (иметь более раннюю временную метку), чем код вызывающего блока, то есть последнее изменение интерфейса должно быть сделано до вызова блока. Программный редактор обновляет временную метку интерфейса при изменении числа параметров, или когда меняется тип данных или значение по умолчанию.

Конфликт временных меток

Конфликт временных меток (Time stamp conflict) возникает, когда интерфейс вызываемого блока имеет более позднюю временную метку по сравнению с кодом вызывающего блока. Такой конфликт временных меток произойдет, если вы откроете скомпилированный блок. Программный редактор помечает некорректный вызов блока красным цветом. Конфликт временных меток может быть вызван, например, при модифицировании интерфейсов блоков, уже вызванных из других блоков, если в

новой программе комбинируются блоки из разных программ, или при повторной компиляции раздела полной программы из исходного файла.

Наряду с этим, конфликт интерфейса, обычно описываемый как «конфликт временных меток», может также иметь другие вызывающие его причины. Он также возникает, если вызываемый блок или адресуемый блок «младше» (имеет более позднюю временную метку), чем вызывающий блок. Примерами, при которых возможны конфликты временных меток, могут служить следующие случаи:

- Интерфейс вызываемого блока «младше», чем код вызывающего блока;
- Инициализация интерфейса не согласована с интерфейсом блока;
- Функциональный блок имеет более позднюю временную метку по сравнению с его экземплярным блоком данных (экземплярный блок данных генерируется на основе описания интерфейса функционального блока и, таким образом, должен быть «младше» функционального блока или их временные метки должны совпадать);
- Локальный экземпляр «младше» вызывающего экземпляра (для функциональных блоков);
- Пользовательский тип данных UDT «моложе» блока, чьи переменные имеют тип UDT; это может быть любой блок, включая блок данных, или другой UDT.

Корректировка неправильных вызовов блоков

Редактор программ обеспечивает поддержку корректировки неверных обращений или UDT-приложений. Данная функция вызывается по команде меню Edit → Block Call → Update (Правка → Вызов блока → Обновить). В большинстве случаев при одинаковых именах, типах данных или позициях редактор может найти правильные назначения; иначе вы должны произвести коррекцию вручную. В любом случае нужно проверить скорректированный вызов.

Проверка согласованности блоков

Когда вы открываете блок с конфликтом временных меток, редактор программ лишь сообщает о данном событии. Если вы хотите проверить всю программу, то можно воспользоваться функцией «Check block consistency» («Проверка согласованности блоков»). Эта функция снимает большинство конфликтов интерфейса и указывает на места в программе, требующие редактирования.

Чтобы выполнить проверку согласованности отметьте контейнер *Blocks* (*Блоки*) и затем выберите опцию меню Edit → Check Block Consistency (Правка → Проверка согласованности блоков). Редактор программ генерирует данные, необходимые для проверки согласованности, начиная с версии 5 SP3 пакета STEP 7. Если программа пользователя скомпилирована в более ранней версии или содержит блоки, которые откомпилированы в более ранней версии (эта ситуация распознается по тому факту,

что в окне «Check block consistency» («Проверка согласованности блоков») не отображены соответствующие зависимости), то в этом окне выберите Program → Compile (Программа → Компилировать).

Редактор программ отображает ход выполнения проверки согласованности и ее результат в информационном окне «1:Compile» («1:Компиляция»). Проверка согласованности не может быть применена к программам в библиотеках.

Для вызванных или адресуемых блоков зависимости отображаются в виде древовидной диаграммы (рисунок 3.4). Вы можете выбрать представление из двух следующих.

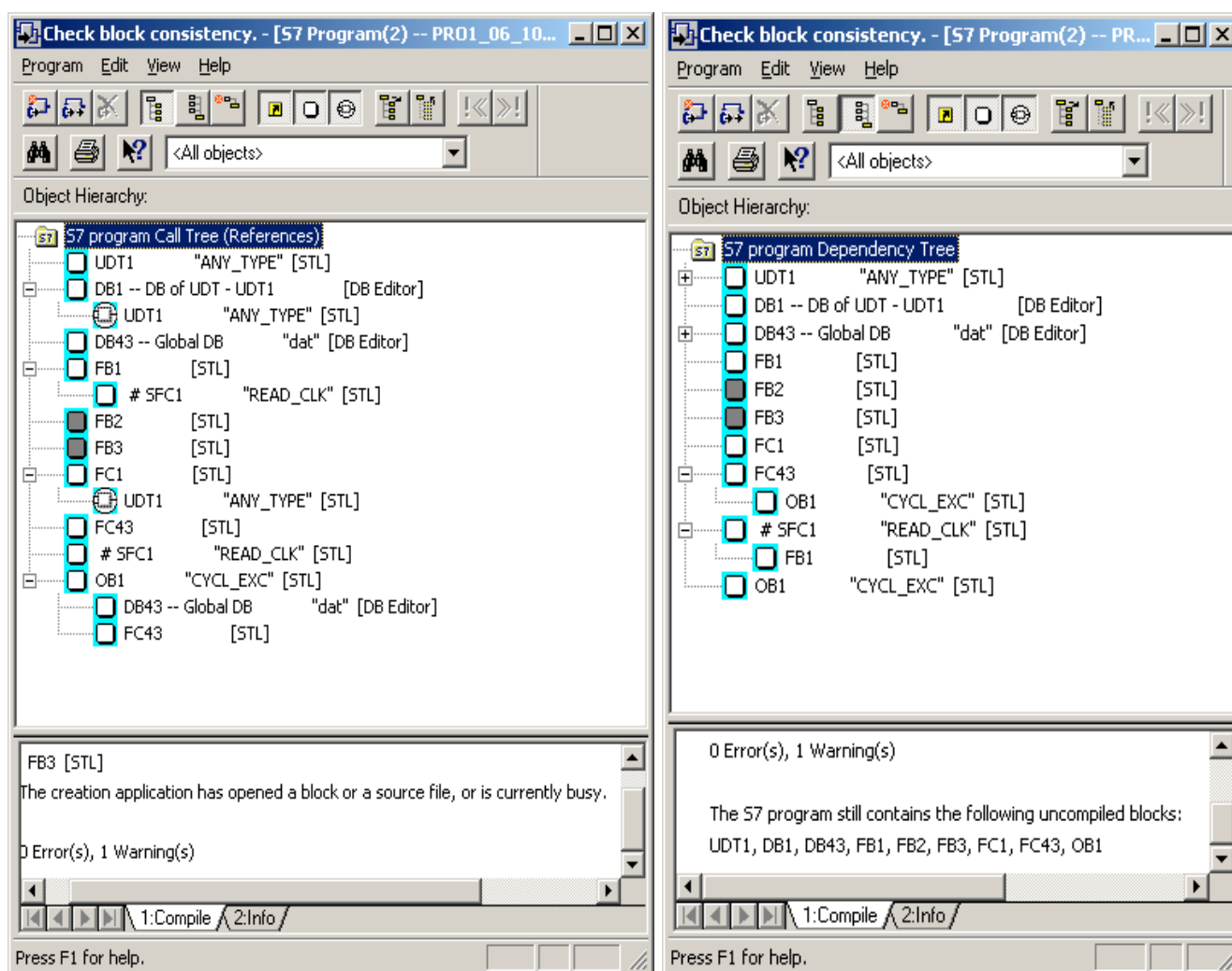


Рисунок 3.4 Пример представления зависимостей в окне проверки согласованности блоков

Дерево ссылок (reference tree) отображает отношения подобно программной структуре: слева показаны вызывающие блоки, правее отображены вызванные ими блоки. Пример: экземпляр DB 20/FB 20 вызывается из OB 1, а локальные экземпляры FB 21 и FB 22 вызываются в FB 20.

Дерево зависимостей (dependency tree) демонстрирует зависимости, начиная со всех вызванных и адресованных блоков. Они расположены в левом столбце, а справа приведены вызывающие блоки. Пример: FB 32 хранит свои данные в экземпляре DB 20/FB 20, который вызывается из OB 1. Он также имеет свой собственный DB 29, и он вызывается как локальный экземпляр из FB 20.

В обоих случаях знак восклицания, например, информирует о том, что соответствующий блок должен быть скорректирован и повторно откомпилирован. Белый крестик на красном фоне показывает (вызванный или адресованный) блок, который выдал ошибку.

Если вы отметите блок в древовидной диаграмме или в открытом окне, то с помощью команды меню Edit → Open Block (Правка → Открыть блок) сможете отредактировать его, к примеру, исправить некорректный вызов.

3.3 Программирование кодовых блоков

Глава 2.5 «Создание программы S7» содержит введение в процесс создания программ и использование редактора программ.

3.3.1 Генерирование блоков

Программирование блока начинается с его открытия. Открыть блок можно либо двойным щелчком на нем в окне проекта SIMATIC-менеджера, либо в редакторе по команде меню File → Open (Файл → Открыть). Если блок еще не создан, то сгенерировать его можно следующими средствами:

- В SIMATIC-менеджере в левой половине окна проекта выбирается объект *Blocks* (*Блоки*), и с помощью команды Insert → S7 Block → ... (Вставка → Блок S7 → ...) генерируется новый блок. Вы увидите окно свойств (*Properties*) блока. На вкладке этого окна «General – Part 1» («Общие – Часть 1») задайте номер блока и выберите язык программирования – «LAD» или «FBD». Оставшиеся атрибуты вы можете ввести позже.
- В редакторе выбирается команда меню File → New (Файл → Новый), которая отобразит диалоговое окно с перечнем параметров заголовка блока (номер блока, язык, атрибуты блока). После закрытия диалогового окна вы можете вводить программу этого блока. Редактор программ использует язык, установленный на вкладке «Create Block» («Создание блока»), доступной по команде меню Options → Customize (Опции → Настроить).

Информацию для заголовка блока вы можете ввести при генерировании блока, или ввод атрибутов блока можно осуществить позже в редакторе, открыв блок и выбрав опцию меню File → Properties (Файл → Свойства).

Окно блока

После открытия блока на экран будет выведено окно, состоящее из трех областей (рисунок 3.5). К ним относятся:

- Сверху расположена таблица описания (объявления) переменных (*variable declaration table*).
Здесь определяются локальные переменные блока.
- Ниже расположено окно программы (*program window*).
Здесь осуществляется ввод программы блока.
- Каталог программных элементов (*program element catalog*).
В STL этот каталог содержит доступные для использования блоки.

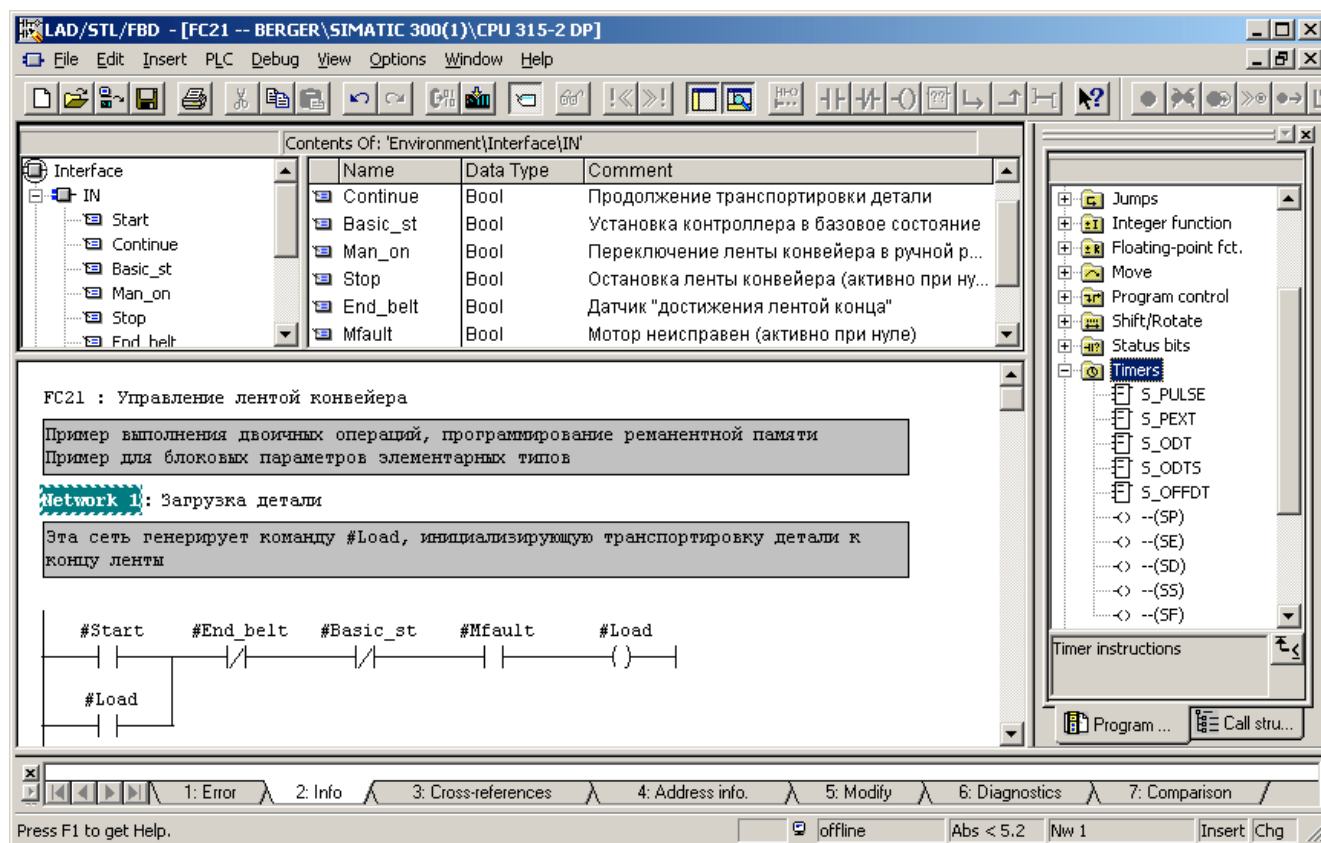


Рисунок 3.5 Пример открытого блока LAD

Таблица описания (объявления) переменных

Окно таблицы описания переменных расположено над окном программы. Если таблица не видна, позиционируйте указатель мыши на верхней границе окна программы и, когда он изменит свою форму, нажмите левую клавишу мыши и перетащите границу вниз. Вы увидите таблицу описания переменных. В этой таблице вами определяются локальные переменные блока (смотрите таблицу 3.2).

Таблица 3.2 Типы переменных в разделе описаний

Тип переменных	Объявление	Может использоваться в типах блоков		
Input parameters (Входные параметры)	in	-	FC	FB
Output parameters (Выходные параметры)	out	-	FC	FB
In-out parameters (Входные/выходные параметры)	in_out	-	FC	FB
Static local data (Статические локальные данные)	stat	-	-	FB
Temporary local data (Временные локальные данные)	temp	OB	FC	FB

Некоторые типы переменных (*variable types*) могут использоваться не во всех видах кодовых блоков. Если вы какой-то тип переменных не используете, то соответствующая строка не заполняется.

Описание переменных состоит из имени, типа данных, значения по умолчанию (если есть) и комментария (необязательный элемент). Значение по умолчанию может быть назначено не всем переменным (например, значения по умолчанию не могут присваиваться временным локальным данным). Более подробно значения по умолчанию для функций и функциональных блоков описаны в главе 19 «Параметры блоков».

Порядок описаний в кодовом блоке фиксирован (как показано в таблице выше), в то же время порядок следования типов переменных произволен. Вы можете экономно использовать память, если будете группировать двоичные переменные в блоки по 8 или 16 штук, а переменные типа BYTE (байт) - парами. Редактор сохраняет (новые) переменные типов BOOL (логический) или BYTE (байт), выделяя им байтовые участки памяти. Для всех остальных типов переменных выделяются участки памяти по одному и более машинных слов (начиная с байта с четным адресом).

Окно программы

В окне программы в зависимости от установок редактора по умолчанию вы увидите поля для заголовка блока и комментария к нему и, если это первый сегмент (*network*), поля для заголовка сегмента (сети) и его комментариев, а также поле для ввода программы. В разделе программы кодового блока вы можете настроить отображение комментариев и символов с помощью команд меню View → Comment (Вид → Комментарий), View → Symbolic Representation (Вид → Представление символов) и View → Symbol Information (Вид → Информация о символе). Можно изменить размер отображения, для этого используются опции меню View → Zoom In (Вид → Увеличить масштаб), View → Zoom Out (Вид → Уменьшить масштаб) и View → Zoom Factor (Вид → Коэффициент масштабирования).

Программирование сегментов (сетей)

Вы можете разделить LAD/FBD-программу на сегменты, каждый из которых представляет контактный план или логическую операцию (функциональный план). Редактор автоматически нумерует сегменты, начиная с 1. Каждый блок может вмещать до 999 сегментов. Для каждого сегмента можно задать заголовок (*network title*) и комментарии (*network comment*). Редактируя, вы можете выбирать каждый сегмент непосредственно при помощи команды меню Edit → Go To → ... (Правка → Перейти к → ...). Сегментирование проводить необязательно.

Для ввода кода программы щелкните один раз ниже окна комментариев сегмента или, если вы установили опцию «Display with comments» («Отображать с комментариями»), щелкните ниже затененной области комментариев сегмента. Появится пустое окно с рамкой. Начать ввод программы вы можете в любом месте окна. Ниже даётся представление о контактном плане (*current path*) LAD и функциональном плане (*logic operation*) FBD.

Начать программирование нового сегмента можно с помощью команды меню Insert → Network (Вставка → Сегмент). При этом редактор вставит пустой сегмент после отмеченного в данный момент сегмента.

Вам нет необходимости завершать блок специальным оператором, просто закончите ввод программы. Однако, вы можете вставить в последний (пустой) сегмент заголовок «Block End» («Конец блока»), тем самым визуально обозначив конец блока (упрощение, в частности, в случае исключительно больших блоков).

Шаблоны сегментов (Network templates)

Подобно тому, как вы храните блоки в библиотеках для их повторного использования в других программах, вы также можете сохранять шаблоны сегментов, чтобы копировать их, к примеру, в другие блоки.

Чтобы сохранить шаблон сегмента, создайте библиотеку, содержащую, по крайней мере, одну S7-программу и контейнер *Source Files* (*Исходные файлы*).

Сегменты, которые вы хотите использовать в качестве шаблонов, программируются вполне «обычно» в (любом) блоке. Затем вы должны заменить адреса, которые будут изменяться, псевдосимволами %00 ... %99. Таким же образом вы можете заменить заголовок сегмента и его комментарии.

Псевдосимволы, заменяющие адреса, представлены в красном цвете, потому что блок не может быть сохранен в таком виде. Это неважно, так как после сохранения шаблона(ов) сегмента этот блок может быть отклонен (закройте блок без сохранения).

После ввода псевдосимволов отметьте сегмент, щелкнув на номере сегмента вверху слева перед заголовком сегмента. Также вы можете объединить несколько сегментов для создания одного шаблона; удерживайте нажатой клавишу Ctrl при выборе следующих номеров сегментов.

Теперь выберите опцию меню Create → Network Template... (Создать → Шаблон сегмента). В появившемся диалоговом окне вы можете составить осмысленные комментарии для всех псевдосимволов. В следующем диалоговом окне вы должны присвоить шаблону имя и определить местоположение для хранения (контейнер *Source Files* (*Исходные файлы*) в библиотеке).

Чтобы воспользоваться шаблонами сегментов, откройте в каталоге программных элементов (Program Elements Catalog) соответствующую библиотеку и выберите затем требуемый шаблон сегмента (дважды щелкните или перетащите в окно редактора). Автоматически будет выведено диалоговое окно, в котором вы произведете замену псевдосимволов достоверными данными. Шаблон сегмента вставляется после выделенного сегмента.

Абсолютная адресация (Absolute addressing)

При абсолютной адресации обращение к адресам и параметрам блока происходит с помощью идентификатора (ID) адреса и битового/байтового адреса. Если в сегменте на месте адресов и параметров указано три красных знака вопроса, то вы должны заменить их достоверным адресом. Если указаны три черные точки, замена необязательна.

Программный редактор производит проверку корректности типов данных адресов и параметров. Вы можете деактивировать некоторые из этих проверок. Для этого предназначена опция «Type check for address» («Проверка типа адреса») на вкладке «LAD/FBD», доступной по команде меню Options → Customize (Опции → Настроить).

Символическая адресация (Symbol addressing)

Если вы в пошаговом программировании хотите применить для глобальных операндов символические имена, эти имена уже должны быть присвоены абсолютным адресам в таблице символов. В процессе написания программы в программном редакторе вы можете вызвать таблицу символов и внести в нее изменения. Для этого выберите опцию меню Options → Symbol Table (Опции → Таблица символов), после чего можно модифицировать символы и добавлять новые.

Активируйте отображение символических адресов с помощью команды меню View → Display → Symbolic Representation (Вид → Отобразить → Символическое представление). Пункт меню View → Display → Symbol Information (Вид → Отобразить → Информация о символе) предоставляет (для каждого сегмента) список назначений всех символов сегмента абсолютным адресам.

В ходе ввода символов вы можете просмотреть список всех символов, занесенных в таблицу символов, с помощью выбора команды Insert → Symbol (Вставка → Символ), или щелкнув правой кнопкой мыши и выбрав пункт Insert → Symbol (Вставка → Символ) в появившемся контекстном меню, после чего вы сможете произвести необходимые действия с выбранным символом. Список отображается автоматически, если вы установили View → Display → Symbol Selection (Вид → Отобразить → Выбор символа).

Декомпилирование

Когда редактор открывает скомпилированный блок, он выполняет «декомпиляцию» в форму представления LAD/FBD. При этом он использует не соответствующие исполняемой программе программные разделы из базы данных программатора, чтобы вывести подобные символы, комментарии и метки переходов. Если на стадии декомпилирования информации в базе данных программатора недостаточно, редактор использует подстановочные символы.

Сегменты, которые не могут быть декомпилированы в представление LAD/FBD, приводятся к виду STL.

Адаптация вызовов блоков

Если вызов блока не соответствует интерфейсу блока в выводе программы (program output), например, потому, что параметр блока был изменен, или интерфейс блока «младше» вызова («конфликт временных меток»), вы можете адаптировать вызов блока с помощью пункта меню Edit → Call → Update (Правка → Вызов → Обновить). Точно так же вы можете адаптировать некорректное использование определенных пользователем типов данных UDT.

При помощи опций меню Edit → Call → Change to Multi-Instance Call (Правка → Вызов → Изменить на мультиэкземплярный вызов) и Edit → Call → Change to FB/DB Call (Правка → Вызов → Изменить на вызов FB/DB) можно изменить вызовы функциональных блоков на вызовы локальных экземпляров или вызовы с блоками данных. После модификации вызовов блоков вы должны заново сгенерировать затрагиваемые экземпляры блоков данных.

Каталог программных элементов

Если каталог программных элементов не показан, то его можно отобразить по команде меню View → Catalog (Вид → Каталог) или Insert → Program Elements (Вставка → Программные элементы).

Каталог программных элементов расположен в специальном окне, которое вы можете перемещать или пристыковать к правой границе окна редактора (в каждом случае требуется двойное нажатие мышью на линейке заголовка окна каталога).

Каталог программных элементов поддерживает программирование на LAD и FBD, предоставляя имеющиеся графические элементы, а также блоки, уже расположенные в автономном контейнере Blocks (Блоки), запрограммированные мультиэкземпляры и доступные библиотеки (рисунок 3.6).

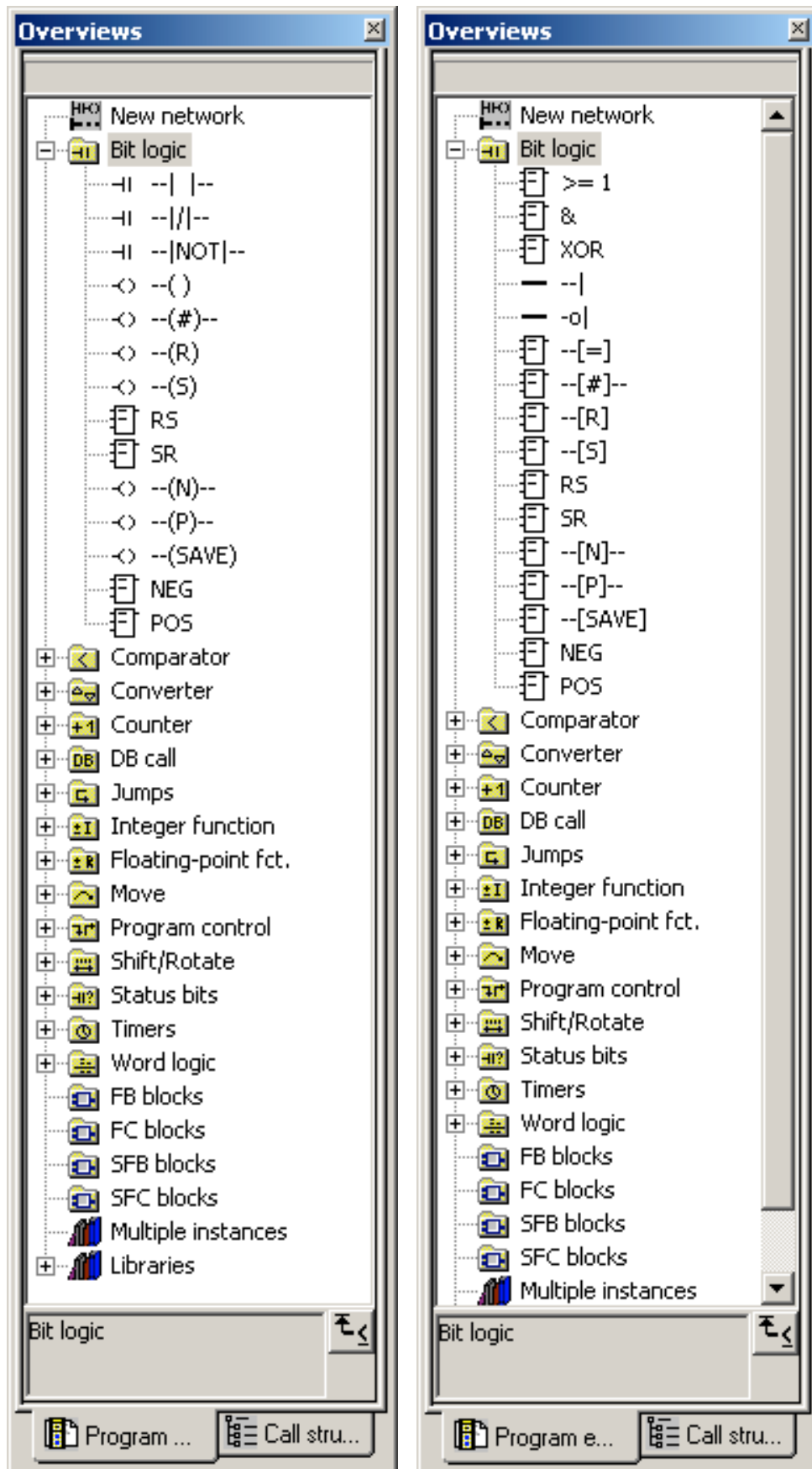


Рисунок 3.6 Каталог программных элементов для LAD и FBD

3.3.2 Редактирование элементов LAD

Программирование в общих чертах

Программа состоит из отдельных элементов LAD, соединенных последовательно или параллельно один по отношению к другому. Контактная схема подобна электрической цепи. Программирование контактного плана (current path) или звена (rung) начинается на левой несущей или левой питающей шине (power rail). Вы должны выбрать место в звене, куда следует вставить элемент, затем выбирается требуемый программный элемент

- при помощи нажатия соответствующей функциональной клавиши (например, клавиша F2 используется для нормально разомкнутого (normally open - NO) контакта),
- при помощи нажатия соответствующей кнопки на функциональной линейке,
- из каталога программных элементов (пункт меню Insert → Program Elements or View → Catalog (Вставка → Программный элемент или вид → Каталог)).

Завершается звено катушкой (coil) или прямоугольным блочным элементом (box). Об этом рассказывается в текущем параграфе 3.3 «Программирование кодовых блоков».

Большинству программных элементов должны быть назначены ячейки памяти (переменные). Самый простой способ сделать это – сначала выстроить все программные элементы, затем назначить им метки (label).

Контакты (Contacts)

Бинарные адреса, такие как входы (inputs), сканируются с использованием контактов. Сканируемые сигнальные состояния комбинируются в соответствии с компоновкой контактов в последовательной или параллельной топологии.

«Ток течет» через нормально разомкнутый контакт (normally open contact), если сканируемый бинарный адрес имеет сигнальное состояние «1» (контакт активирован); «ток течет» через нормально разомкнутый контакт (normally closed contact), если сканируемый бинарный адрес имеет сигнальное состояние «0» (контакт не активирован). Кроме того, вы можете сканировать биты состояния (слово статуса) или инвертировать результат логической операции (контакт NOT (NE)).

Катушки (Coils)

Катушки используются для управления бинарными адресами, такими как выходы (outputs). Простая катушка устанавливает бинарный адрес, когда в катушке течет ток, и сбрасывает его при отключении тока.

Имеются катушки с дополнительными метками, например, катушки установки (Set coil) и сброса (Reset coil), которые выполняют специальные функции. Катушки также применяются для управления таймерами и счетчиками, вызова блоков без параметров, выполнения переходов в программе и так далее.

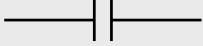
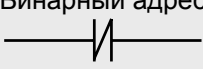
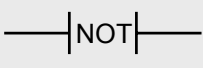
Прямоугольные блочные элементы (Boxes)

Прямоугольные блочные элементы представляют элементы LAD со сложными функциями. STEP 7 предоставляет «стандартные блочные элементы» двух различных типов:

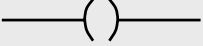
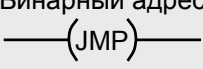
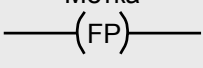
- без механизма EN/ENO, например, функции работы с памятью, функции таймера и счетчика, блочные элементы с функцией сравнения;
- с EN/ENO, например, MOVE (Переместить), арифметические и математические функции, преобразование типов данных.

Когда вы вызываете кодовые блоки (блоки FC, FB, SFC и SFB), LAD представляет вызовы также в виде блочных элементов с EN/ENO. Кроме того, LAD предоставляет «пустой блочный элемент» (Empty box), в который при программировании можно ввести требуемую функцию.

Контакты (Contacts)

Контакт NO	Бинарный адрес 
Контакт NC	Бинарный адрес 
Контакт со специальной функцией (например, отрицание)	

Катушки (Coils)

Простая катушка	Бинарный адрес 
Катушка с дополнительной функцией (например, установка, сброс, оценка уровня или функция перехода)	Бинарный адрес 
	Метка 

Прямоугольные блочные элементы (Boxes)

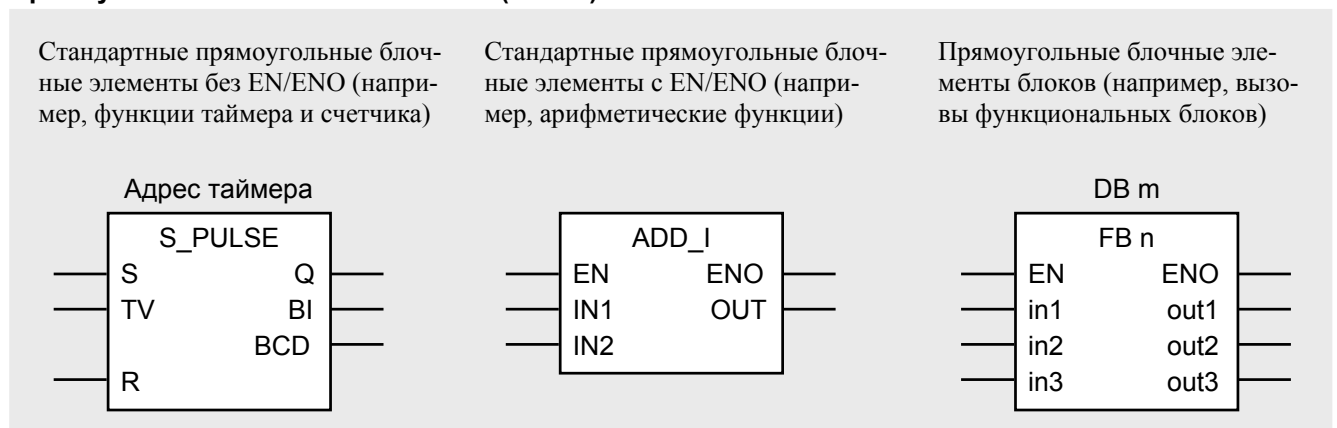


Рисунок 3.7 Примеры программных элементов LAD

Network 2: Parts ready to remove

When the parts have reached the end of the belt, they are ready for removal.

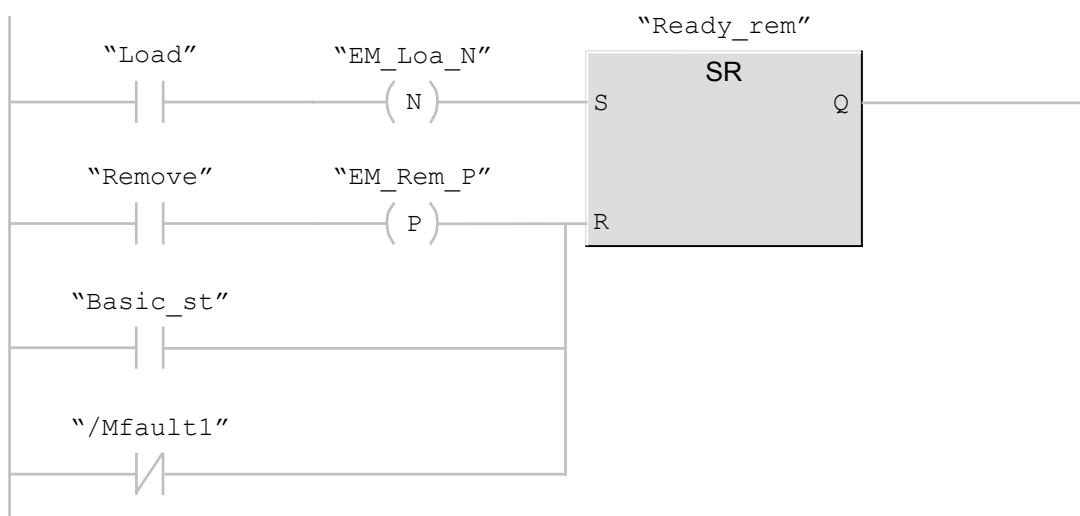


Рисунок 3.8 Пример сегмента LAD в «трехмерном» представлении

Ограничения компоновки

Редактор LAD определяет сегмент в соответствии с принципом «главного звена». Это самая верхняя ветвь, которая начинается непосредственно на левой несущей (питающей шине) и должна завершаться катушкой или блочным элементом. В этом звене могут быть расположены все элементы LAD. В параллельных ветвях, которые не берут начало на левой несущей (питающей шине), иногда действуют ограничения в зависимости от тех или иных программных элементов.

Дополнительные ограничения определяют следующее: элемент LAD не может быть «замкнут накоротко» с «пустой» параллельной ветвью, и «ток» не может протекать через элемент справа налево (параллельная ветвь должна быть замкнута на ветвь, в которой она была разомкнута). Все остальные правила, применяемые к компоновке особых элементов LAD, обсуждаются в соответствующих главах.

При использовании блочных элементов в качестве программных элементов вы можете

- запрограммировать единственный в сегменте блочный элемент;
- скомпоновать блочные элементы в Т-ветви в ветвях, которые начинаются на левой несущей (питающей шине);
- сгруппировать блочные элементы в последовательности путем подключения выхода ENO блочного элемента к входу EN следующего блочного элемента;
- включить блочные элементы параллельно в ветвях на левой несущей (питающей шине) посредством выхода ENO.

С помощью компоновки блочных элементов вы можете определять сигнальные состояния выходов ENO: если выходы ENO закрываете (заделываете) катушкой, то «ток» течет в катушке при условии, что все блочные элементы сработали без ошибок в последовательном соединении, или один из блочных элементов закончил обработку без ошибок в случае параллельного соединения (обратитесь также к параграфу 15.4 «Использование бинарного результата»).

3.3.3 Редактирование элементов FBD

Программирование вообще

Программа состоит из отдельных программных элементов, соединенных посредством потока бинарного сигнала с целью формирования логических операций (функциональных схем, планов) или сегментов. Программирование функциональной схемы начинается с выбора элементов программирования слева от функциональной схемы

- с помощью функциональной клавиши (к примеру, F2 отвечает за функцию AND (И)),
- посредством меню (Insert → FBD Element → AND Box – Вставка → Элемент FBD → Блочный элемент AND) или
- из каталога программных элементов при помощи опций меню Insert → Program Elements (Вставка → Программные элементы) или View → Catalog (Вид → Каталог).

Завершается бинарная логическая операция в простейшем случае блочным элементом присваивания (assign box). Об этом рассказывается в текущем параграфе 3.3 «Программирование кодовых блоков».

Для большинства программных элементов должны быть отведены ячейки памяти (переменные). Самым легким способом осуществить это является следующий: сначала скомпоновать все программные элементы, затем назначить им метки (labels).

Бинарные функции

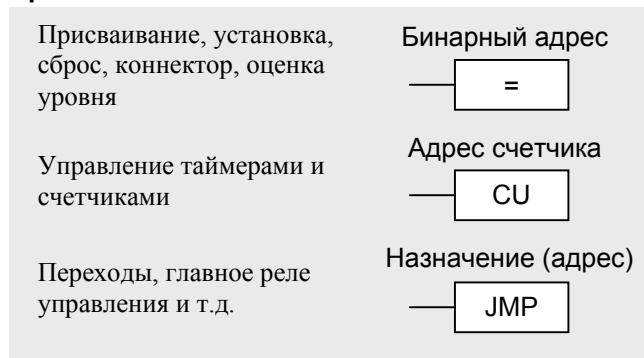
Бинарные (двоичные) адреса, такие как входы, сканируются, и сканируемые сигнальные состояния комбинируются с использованием двоичных функций AND (И), OR (ИЛИ) и Exclusive OR (Исключающее ИЛИ). Каждый бинарный вход блочного элемента также сканирует бинарный адрес на входе.

Результат опроса адреса может быть инвертирован, так что результат сканирования «1» может быть получен из нулевого состояния адреса. Вы также можете сканировать биты статуса или результат функциональной схемы в рамках схемы.

Бинарные функции



Простые блочные элементы



Сложные блочные элементы

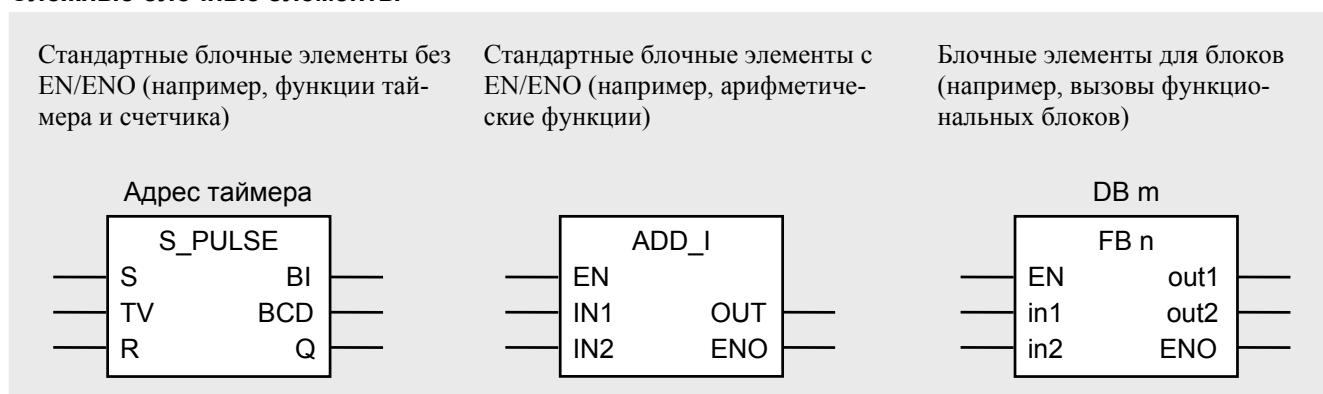


Рисунок 3.9 Примеры программных элементов FBD

Простые блочные элементы

Управление бинарными адресами, такими как выходы, осуществляется с помощью простых блочных элементов. Простые блочные элементы в общем случае имеют только один вход и могут обладать дополнительной меткой.

Есть простые блочные элементы для управления двоичным адресом, оценки уровня (фронта), управления адресами таймеров и счетчиков, вызова блоков без параметров, выполнения переходов в программе и так далее.

Сложные блочные элементы

Сложные блочные элементы представляют программные элементы со сложными функциями. STEP 7 предоставляет «стандартные блочные элементы» в двух вариантах:

- без механизма EN/ENO (такие как функции по работе с памятью, таймеры и счетчики, блочные элементы с функцией сравнения) и
- с EN/ENO (такие как MOVE, арифметические и математические функции, преобразование типов данных).

Если вы вызываете кодовые блоки (FC, FB, SFC и SFB), FBD представляет вызовы также в виде блочных элементов с EN/ENO.

Кроме того, FBD предоставляет «пустой блочный элемент» (Empty box), в который при создании программы вы можете ввести необходимую функцию.

Network 2: Parts ready to remove

When the parts have reached the end of the belt, they are ready for removal.

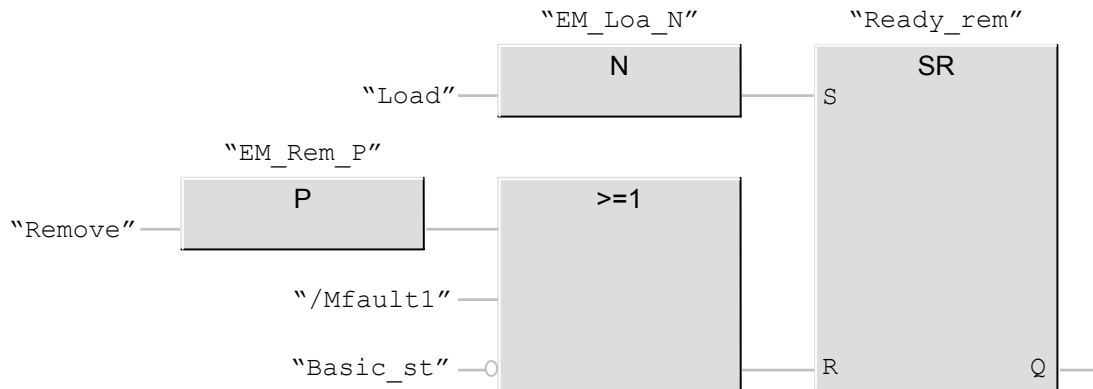


Рисунок 3.10 Пример сегмента FBD в «трехмерном» представлении

Ограничения компоновки

Редактор FBD определяет сегмент слева направо и сверху вниз. Слева входы ведут к функциональным элементам, а выходы отходят справа.

Функциональный план всегда имеет «завершающую функцию». В своей простейшей форме это присваивание результата функционального плана бинарному адресу.

С помощью Т-ветви функциональной схемы вы можете программировать другие «завершающие функции» этой схемы («множественный выход»). Однако, выбор программируемых элементов после Т-ветви ограничен. Например, нельзя выстроить блочные элементы оценки уровня и вызова после Т-ветви. Все следующие правила, применимые к компоновке особых элементов FBD, рассматриваются в соответствующих главах.

Используя блочные элементы в качестве программных элементов, вы можете

- запрограммировать единственный в сегменте блочный элемент;
- скомпоновать блочные элементы в Т-ветви в ветвях, которые начинаются на левой несущей (питающей шине);
- построить блочные элементы в последовательности путем подключения выхода ENO одного блочного элемента к входу EN следующего элемента;

- блочные элементы AND или OR через выход ENO.

В случае блочных элементов, составленных в последовательности, вы можете осуществлять групповое управление ими (обратитесь к параграфу 15.4 «Использование бинарного результата»). Оценка сообщений об ошибках блочных элементов происходит путем комбинирования выходов ENO: объединение по операции AND выходов ENO выполняется, если все блочные элементы обработаны без ошибок, и объединение по операции OR выходов ENO выполняется, если (хотя бы) один из блочных элементов сработал безошибочно.

3.4 Программирование блоков данных

Глава 2.5 «Создание программы S7» содержит введение в процесс создания программ и использование редактора программ. Блоки данных (data blocks) программируются тем же способом в LAD и FBD.

3.4.1 Создание блоков

Программирование блока начинается с его открытия. Открыть блок можно либо двойным щелчком на нем в окне проекта SIMATIC-менеджера, либо в редакторе по команде меню File → Open (Файл → Открыть). Если блок еще не создан, то сгенерировать его можно следующими способами:

- В SIMATIC-менеджере: в левом разделе окна проекта выбирается объект *Blocks* (Блоки), и с помощью команды Insert → S7 Block → Data Block (Вставка → Блок S7 → Блок данных) создается новый блок данных. Вы увидите окно свойств (Properties) блока. На вкладке этого окна «General – Part 1» («Общие – Часть 1») введите номер блока. Язык программирования задается в поле «DB». Оставшиеся атрибуты вы также можете ввести позже.
- В редакторе: выбирается команда меню File → New (Файл → Новый), которая отобразит диалоговое окно. В нем в разделе «Object Name» («Имя объекта») вы можете ввести желаемый блок. После закрытия диалогового окна вы можете вводить программу этого блока.

Информацию для заголовка блока вы можете ввести после его генерирования, или же ввод свойств блока можно осуществить позже. Дополнительные сведения вы можете задать позже в редакторе, открыв блок и выбрав опцию меню File → Properties (Файл → Свойства).

3.4.2 Типы блоков данных

Когда вы впервые открываете новый блок данных, вы увидите окно «New Data Block» («Новый блок данных»); вы должны решить, какого типа будет данный блок.

С помощью щелчка мыши выбирается один из следующих трех вариантов:

- «Data block» («Блок данных»)

Создается глобальный блок данных; в этом случае при программировании блока данных вы объявляете (описываете) адреса данных;
- «Data block with assigned user-defined data type» («Блок данных с назначенным пользовательским типом данных», «Блок со структурой UDT»)

Создается блок данных пользовательского типа данных; в данном случае вы описываете структуру данных пользовательского типа данных UDT (User defined data type);

- «Data block with assigned function block» («Блок данных с назначенным функциональным блоком»)

Создается экземпляр блока данных; здесь объявляется для пересылки структура данных, описанная при программировании соответствующего функционального блока.

3.4.3 Окно блока

На рисунке 3.11 показано открытое окно блока. Вы можете выбрать один из двух видов отображения информации:

- окно просмотра описаний (declaration view), в этом окне вы можете вводить адреса данных, определять их тип данных и задавать начальное значение;
- окно просмотра данных (data view), в нем вы можете установить фактическое значение.

Address	Name	Type	Initial value	Comment
0.0		STRUCT		
+0.0	Number_1	INT	0	
+2.0	Number_2	INT	0	
+4.0	Sum	INT	0	
+6.0	FirstVol	STRUCT		Пример переменной типа STRUCT
+0.0	FirstWid	INT	0	
+2.0	FirstLen	INT	0	
+4.0	FirstHei	INT	0	
+6.0	FirstTime	TIME_OF_DAY	TOD#0:0:0.0	
=10.0		END_STRUCT		
+16.0	Meas1	STRUCT		Пример вложенной структуры
+0.0	MeasTime	TIME	T#0MS	
+4.0	Volumel	STRUCT		
+0.0	Width1	INT	0	
+2.0	Length1	INT	0	
+4.0	Height1	INT	0	
+6.0	Meastime1	TIME_OF_DAY	TOD#0:0:0.0	
=10.0		END_STRUCT		
=14.0		END_STRUCT		
+30.0	Vorname	STRING[8]	'Hans'	Пример переменной типа STRING
+40.0	Last_name	STRING[14]	'Berger'	Пример переменной типа STRING
+56.0	Header_data	"Header"		Использование UD1 "Header"
=64.0		END_STRUCT		

Рисунок 3.11 Пример открытого блока данных (окно просмотра описаний)

При программировании глобального блока данных вы можете задать для каждого адреса данных начальное значение. Переменным может быть присвоено по умолчанию стандартное нулевое значение, наименьшее значение или пустое в зависимости от типа данных.

Экземплярный блок данных, сгенерированный для функционального блока, в качестве начального принимает значение по умолчанию из раздела описаний функционального блока.

Начальными значениями блока данных со структурой UDT являются значения инициализации (значения по умолчанию) UDT.

Редактор отображает блок данных двумя способами: в режиме отображения описаний и в режиме отображения данных.

В режиме *просмотра описаний* (*Declaration view*), активируемом по команде View → Declaration View (Вид → Вид описаний), вы можете определять все адреса данных и увидеть переменные в том виде, в котором вы их определили, например, массив или тип данных пользователя (UDT) как одну переменную.

В режиме *просмотра данных* (*Data view*), вызываемом опцией меню View → Data View (Вид → Вид данных), редактор по отдельности отображает каждую переменную и каждый компонент или элемент массива или структуры. Вы увидите дополнительный столбец с фактическими значениями. Фактическое значение – это значение, которое имеет или приобретает адрес данных в рабочей памяти CPU. Обычно редактор в качестве фактического значения принимает начальное значение.

Вы можете модифицировать фактические значения отдельно для каждого адреса данных: генерируются несколько экземплярных блоков данных для одного функционального блока. Однако, для каждого вызова функционального блока (для каждой пары FB/DB) вам необходимо только небольшое отличие в предустановках отдельного экземпляра блока данных. Вы сможете отредактировать каждый блок данных с помощью команды меню View → Data View (Вид → Вид данных) и ввести корректные для этого блока данных значения в столбце Actual value (фактическое значение). По команде меню Edit → Initialize Data Block (Правка → Инициализировать блок данных) редактор заменит фактические значения начальными.

3.5 Переменные, константы и типы данных

3.5.1 Общие замечания о переменных

Переменная (variable) – это величина определенного формата (рисунок 3.12). Простые переменные состоят из адреса (например, вход 5.2, где 5 – номер байта, 2 – номер бита в нем). Также можно осуществить доступ к адресу или переменной символически, присвоив адресу имя (символ) в таблице символов.

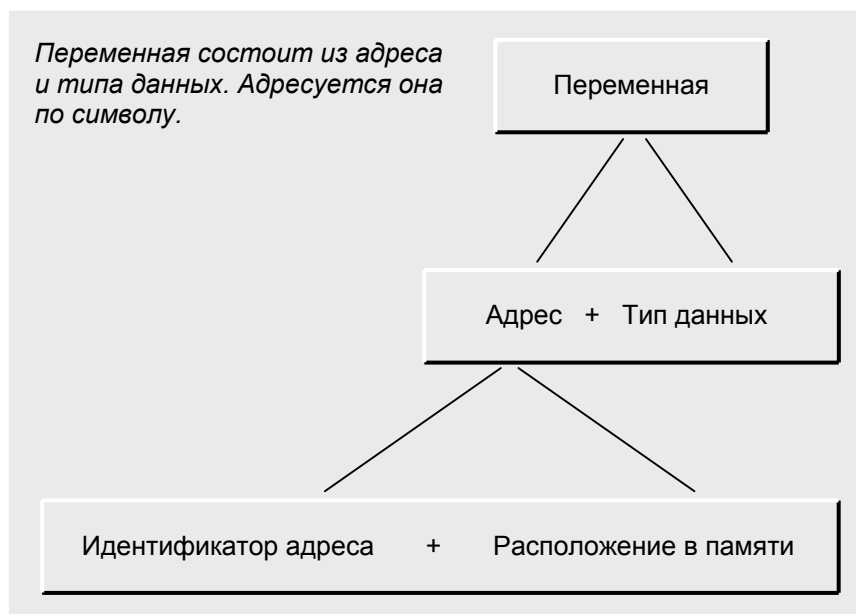


Рисунок 3.12 Структура переменной

Бит данных типа BOOL (логический) называют *двоичным адресом (binary address)* или *двоичным операндом (binary operand)*. Адреса, содержащие один, два или четыре байта или переменные соответствующих типов называются *численными операндами (digital operand)*.

Переменные, которые объявляются внутри блока, называются локальными (внутриблочными) переменными. К ним относятся параметры блока, статические и временные локальные данные и даже адреса данных в глобальных блоках данных. Когда эти переменные являются переменными простого типа данных, они также могут быть доступны как операнды (например, статические локальные данные – как DI-операнды, временные локальные данные – как L-операнды, а данные в глобальных блоках данных – как DB-операнды).

Наряду с этим, локальные переменные могут быть также сложных типов данных (таких как структуры или массивы). Переменные таких типов требуют более 32 бит памяти, поэтому они не могут быть загружены, например, в аккумулятор. И по этой же причине они не могут быть адресованы с помощью «нормальных» STL-операторов. Для обработки этих переменных имеются специальные функции, такие как IEC-функции, которые поставляются со STEP 7 в составе стандартной библиотеки (вы

можете создавать переменные сложного типа данных в параметрах блока того же типа).

Если переменные сложного типа данных содержат компоненты простого типа, то эти компоненты могут обрабатываться, как если бы они были отдельными переменными (например, вы можете загрузить компонент массива, состоящего из 30 целочисленных значений, в аккумулятор и затем обработать его).

Константы (Constants) используются для присваивания переменным фиксированных значений. Константа имеет особый префикс в зависимости от типа данных.

3.5.2 Адресация переменных

При адресации переменных вы можете выбрать один из ее способов: абсолютная адресация (absolute addressing) и символическая адресация (symbolic addressing). Абсолютная адресация использует численные адреса, начиная с нулевого (0), для каждой адресной области. Символическая адресация применяет буквенно-цифровые имена, которые вы определяете в таблице символов для глобальных адресов или в разделе описаний (объявлений) для локальных (внутриблочных) адресов. Расширением абсолютной адресации является косвенная адресация (indirect addressing), при которой адреса ячеек памяти неизвестны до начала выполнения программы и вычисляются во время ее исполнения.

Абсолютная адресация переменных

Доступ к переменным простых типов данных может быть осуществлен по абсолютным адресам.

Абсолютный адрес входа или выхода вычисляется на основе стартового адреса модуля, который вы устанавливаете или уже установили в конфигурационной таблице, и типа сигнального соединения в модуле. Различают бинарные (дискретные) и аналоговые сигналы.

Бинарные (дискретные) сигналы

Бинарный сигнал (binary signal) содержит один бит информации. Примерами дискретных сигналов являются входные сигналы от конечных выключателей, переключателей мгновенного контакта и т. п., которые поступают на цифровые входные модули, и выходные сигналы, которые управляют лампами, контакторами и т. п. через цифровые выходные модули.

Аналоговые сигналы

Аналоговый сигнал (analog signal) содержит 16 бит информации. Аналоговый сигнал соответствует «каналу», который отображается в контроллере в виде машинного слова (word), то есть двух байт (смотрите ниже). Аналоговые входные сигналы (на-

пример, напряжения от терморезисторов) поступают в 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 аналогового выходного модуля, где информация преобразовывается в аналоговую величину (например, ток).

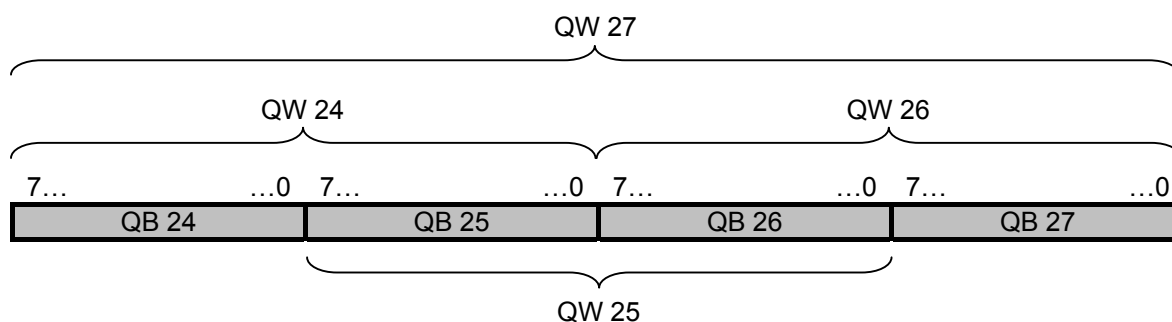


Рисунок 3.13 Содержимое байтов в машинных словах и двойных словах

Разрядность («information width» - «информационный диапазон») сигнала также соответствует разрядности переменной, в которой сигнал хранится и обрабатывается. Информационная разрядность и интерпретация информации (например, позиционный весовой коэффициент), взятые вместе, образуют *тип данных* (*data type*) переменной. Бинарные сигналы хранятся в переменных типа BOOL (логический), аналоговые сигналы – в переменных типа INT (целочисленный).

Единственным определяющим фактором для адресации переменной является разрядность. В STEP 7 с помощью абсолютной адресации осуществляется доступ к объектам, имеющим следующие четыре типа разрядности:

- 1 бит Тип данных BOOL (логический);
- 8 бит Тип данных BYTE (байтовый) или другой 8-битовый тип данных;
- 16 бит Тип данных WORD (слово) или другой 16-битовый тип данных;
- 32 бита Тип данных DWORD (двойное слово) или другой 32-битный тип.

На переменные типа BOOL (логический) 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 номер 0 в байте номер 1;
- Q 16.4 выходной бит номер 4 в байте номер 16.

Для переменных типа BYTE (байт) в качестве абсолютного адреса используются идентификатор адреса и номер байта, содержащего переменную. 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 переменная. 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 содержат четыре байта (двойное слово). Абсолютный адрес составляют идентификатор адреса и номер младшего байта двойного машинного слова, содержащего переменную. 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.

Адреса области данных включают блок 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 параграфе 18.2.2 «Доступ к операндам данных».

Символическая адресация переменных

При символической адресации вместо абсолютных адресов используются имена (называемые символами, symbol). Выбрать имя можете вы сами. Такое имя должно начинаться с литеры и может содержать до 24 знаков. Учитывается регистр (верхний и нижний) написания букв. В STL не разрешено использовать ключевые слова в качестве символов. Для использования ключевых слов в качестве символов в SCL вставьте символ решетки «#» перед таким именем.

Имя (или символ) должно быть назначено абсолютному адресу. Различают глобальные (global) символы и локальные (local) символы, действительные только в блоке.

Глобальные символы

Назначить имена в таблице символов вы можете следующим объектам:

- Блокам данных и кодовым блокам;
- Входам, выходам, периферийным входам, периферийным выходам;
- Меркерам, таймерам и счетчикам;
- Пользовательским типам данных;
- Таблицам переменных.

Глобальный символ может также содержать пробелы, специальные и национальные литеры, такие как умляут. Исключение составляют символьные обозначения 00_{hex} и FF_{hex} . При использовании символов со специальными литерами вы должны заключать в программе такие символы в кавычки. В компилированных блоках редактор STL всегда отображает глобальные символы в кавычках.

Вы можете использовать глобальные символы на протяжении всей программы; каждый такой символ в программе должен быть уникален.

Редактирование, импорт и экспорт глобальных символов описано в параграфе 2.5.2 «Таблица символов».

Локальные символы

Имена локальных данных определяются в разделе описаний соответствующего блока. Эти имена могут содержать только буквы, цифры и знак подчеркивания.

Локальные символы действуют только внутри блока. 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 с впереди стоящим знаком «#». Если редактор не может отличить локальный символ от адреса вы должны предварить (добавить к началу) этот символ знаком «#».

Локальные символы доступны только в базе данных программирующего устройства (в автономном контейнере *Blocks (Блоки)*). Если при декомпиляции эта информация отсутствует, то редактор вставляет заменяющий символ.

В зависимости от структуры и приложения типы данных классифицируются следующим образом.

Использование символических имен

Если вы используете символические имена при программировании в пошаговом редакторе, то они уже должны быть назначены абсолютным адресам. Кроме того, вы можете вводить новые символические имена в таблице символов, программируя в пошаговом редакторе. После ввода нового символического имени вы можете немедленно задействовать его при написании оставшейся части программы. Если вы для ввода программы используете (исходный) текстовый файл, то вы должны сделать абсолютные адреса доступными (присвоить им имена) на момент компиляции.

В случае массивов доступ к их отдельным компонентам осуществляется по имени массива и индексу, например, MSERIES[1] – первый компонент. В LAD и FBD индекс – это целочисленное (INT) постоянное (constant) значение.

В структурах каждый подидентификатор отделяется от предшествующего подидентификатора десятичной точкой, к примеру, FRAME.HEADER.CNUM. Компоненты пользовательских типов данных адресуются точно так же как структуры.

Адреса данных

Символическая адресация данных использует полный путь доступа (адрес), включая блок данных. Пример: блок данных с символическим именем MVALUES содержит переменные MVALUE1, MVALUE2 и MTIME. Эти переменные могут быть адресованы в следующем виде:

```
"MVALUES".MVALUE1
"MVALUES".MVALUE2
"MVALUES".MTIME
```

За дальнейшей информацией вы можете обратиться к параграфу 18.2.2 «Доступ к операндам данных».

3.5.3 Обзор типов данных

Типы данных (data types) обуславливают характеристики данных, по сути, представление содержимого переменной, и допустимые области значений. STEP 7 предлагает predefined типы данных, из которых вы можете составлять пользовательские типы данных (user-defined data types, UDT).

Типы данных доступны на основе принципа глобальности и могут быть использованы в каждом блоке. LAD и FBD используют одни и те же типы данных.

- Простые типы данных (Elementary data types),
- Сложные типы данных (Complex data types),
- Пользовательские типы данных (User data types),
- Параметрические типы данных (для параметров) (Parameter data types).

В таблице 3.3 приведены свойства этих классов типов.

На дискете, прилагаемой к этой книге, в библиотеках «LAD_Book» и «FBD_Book» в разделе «Data Types» вы найдете примеры описания (объявления) и использования переменных всех типов данных.

Таблица 3.3 Классификация типов данных

Простые типы данных	Сложные типы данных	Пользовательские типы данных	Типы данных параметров
BOOL, BYTE, CHAR, WORD, INT, DATE, DWORD, DINT, REAL, S5TIME, TOD	DT, STRING, ARRAY, STRUCT	UDT Глобальных блоки данных Экземпляры	TIMER, COUNTER, BLOCK_DB, BLOCK_SDB, BLOCK_FC, BLOCK_FB, POINTER, ANY
Типы данных, размер которых не превышает одного двойного слова (32 бита)	Типы данных, которые могут содержать более одного двойного слова (DT, STRING) или состоят из нескольких компонентов	Структуры или области данных, которым может быть присвоено имя	Параметры блоков
Могут быть соотнесены с операндами, адресуемыми по абсолютным или символическим адресам	Могут быть назначены только переменным, обращение к которым происходит с помощью символической адресации		Могут быть назначены только параметрам блоков (только символическая адресация)
Допустимы во всех адресных областях	Допустимы в блоках данных (как глобальные данные и экземплярные данные), как временные локальные данные и в качестве параметров блоков		Допустимы в сочетании с параметрами блоков

3.5.4 Простые типы данных

Простые типы данных могут резервировать бит, байт, слово или двойное слово.

В таблице 3.5 показаны простые типы данных. Для многих типов данных существует два представления констант, которые вы можете применять в равной степени (например, TIME# или T#). Таблица содержит минимальное значение для типа данных в верхней строке и максимальное значение в нижней строке.

Описание (объявление) простых типов данных

В таблице 3.4 приведены некоторые примеры описания переменных простых типов. *Имя (Name)* – это идентификатор локальной (внутриблочной) переменной (может содержать до 24 знаков, включая только буквы, цифры и знак подчеркивания). В столбце *Тип (Type)* вводится соответствующий тип данных.

Таблица 3.4 Примеры описаний и начальных значений простых типов данных

Имя (Name)	Тип (Type)	Начальное значение (Initial Value)	Комментарии (Comments)
Automatic	BOOL	FALSE	Начальное значение – сигнальное состояние «0»
Manual_off	BOOL	TRUE	Начальное значение – сигнальное состояние «1»
Measured_value	DINT	L#0	Начальное значение переменной типа DINT
Memory	WORD	W#16#FFFF	Начальное значение переменной типа WORD
Waiting_time	S5TIME	S5T#20s	Начальное значение переменной типа S5TIME

За исключением временных локальных данных и блочных параметров функций переменным можно назначить *начальное значение (initial value)*. Для этих целей используйте синтаксис, соответствующий типу данных. *Комментарии (Comments)* не обязательны.

BOOL (логический), BYTE (байт), WORD (слово), DWORD (двойное слово), CHAR (литерный)

Переменная типа BOOL представляет значение бита, к примеру, I 1.0. Переменные типов BYTE, WORD и DWORD – это битовые строки (совокупности битов), содержащие 8, 16 и 32 бита соответственно. Отдельные биты не определяются.

Особыми формами этих типов данных являются двоично-десятичные числа (BCD-числа) и значения счетчика (count), используемые совместно с функциями счетчика, а также тип данных CHAR, который представляет ASCII-символ (литера).

BCD-числа

Двоично-десятичные числа (Binary coded decimal – BCD) не имеют специального идентификатора. Просто вводите BCD-число с типом данных 16# (шестнадцатеричный) и используйте только цифры от 0 до 9.

BCD-числа встречаются в кодированной обработке значений таймеров и счетчиков и в сочетании с функциями преобразования. Тип данных S5TIME# применяется для определения значения времени для запуска таймера (смотрите ниже), тип данных 16# или C# - для определения значения счетчика. Значением счетчика C# является BCD-число из интервала 000 ... 999, вследствие чего бит знака всегда равен 0.

Как правило, BCD-числа не имеют знака. В сочетании с функциями преобразования знак BCD-числа хранится в самом левом (высшем) разряде, поэтому под само число отведено на один разряд меньше.

Когда BCD-число занимает 16-битное слово, знак находится в высшем разряде, то есть в 15-м бите. Сигнальное состояние «0» означает, что число положительное. Сигнальное состояние «1» обозначает тот факт, что число отрицательное. Знак не

влияет на содержимое отдельных разрядов. Подобное распределение действительно и для 32-битного слова.

Область значений 16-битного BCD-числа: от 0 до ± 999 . Область значений 32-битного двоично-десятичного числа: от 0 до $\pm 9\,999\,999$.

Таблица 3.5 Обзор простых типов данных

Тип данных	(Разрядность)	Описание	Пример записи констант
BOOL	(1 бит)	Бит (одноразрядное значение)	FALSE TRUE
BYTE	(8 битов)	8-разрядное шестнадцатеричное число	B#16#00, 16#00 B#16#FF, 16#FF
CHAR	(8 битов)	Одна литера (ASCII)	Печатные литеры, например, 'A'
WORD	(16 битов)	16-разрядное шестнадцатеричное число	W#16#0000, 16#0000 W#16#FFFF, 16#FFFF
		16-разрядное двоичное число	2#0000_0000_0000_0000 2#1111_1111_1111_1111
		Значение счетчика, 3 декады (разряда) BCD	C#000 C#999
		Два 8-разрядных числа без знака	B(0,0) B(255,255)
DWORD	(32 бита)	32-разрядное шестнадцатеричное число	DW#16#0000_0000, 16#0000_0000 DW#16#FFFF_FFFF, 16#FFFF_FFFF
		32-разрядное двоичное число	2#0000_0000...0000_0000 2#1111_1111...1111_1111
		Четыре 8-разрядных числа без знака	B(0,0,0,0) B(255,255,255,255)
INT	(16 битов)	Число с фиксированной запятой	-32 768 +32 767
DINT	(32 бита)	Число с фиксированной запятой	L#-2 147 483 648 L#+2 147 483 647
REAL	(32 бита)	Число с плавающей запятой (область значений указана в тексте)	+1.234567E+02 в экспоненциальном виде
			123.4567 в десятичном представлении
S5TIME	(16 битов)	Значение времени в формате SIMATIC	S5T#0ms S5TIME#2h46m30s
TIME	(32 бита)	Значение времени в формате IEC	T#-24d20h31m23s647ms TIME#24d20h31m23s647ms
			T#-24.855134d TIME#24.855134d
DATE	(16 битов)	Дата	D#1990-01-01 DATE#2168-12-31
TIME_OF_DAY	(32 бита)	Время суток	TOD#00:00:00 TIME_OF_DAY#23:59:59.999

Char (литера)

Переменная типа CHAR (character, литера) занимает один байт. Тип данных CHAR представляет одну литеру в ASCII-формате, например, 'A'.

Работая с этим типом данных, вы можете использовать любую печатную литеру в апострофах. Некоторые особые литеры требуют записи, показанной в таблице 3.6. Пример: '\$\$' представляет знак доллара в ASCII-коде.

Функция MOVE (Переместить) позволяет вам задействовать два или четыре ASCII-символа, заключенных в апострофы, в качестве специальной формы типа данных CHAR для начертания ASCII-символов в переменной.

Таблица 3.6 Специальные литеры для типа данных CHAR

CHAR	Hex (шестнадцатеричное число)	Описание
\$\$	24 _{hex}	Знак доллара (Dollar sign)
'\$'	27 _{hex}	Апостроф (Apostrophe)
\$L или \$l	0A _{hex}	Подача (бумаги) на одну строку (Line feed, LF)
\$P или \$p	0C _{hex}	Новая страница (New page, FF)
\$R или \$r	0D _{hex}	Возврат каретки (Carriage return, CR)
\$T или \$t	09 _{hex}	Табулятор (Tabulator)

INT (целое число)

Переменная типа INT (integer) хранится как целое число (16-битное число с фиксированной запятой или десятичной точкой). Тип данных INT не имеет специального идентификатора.

Целочисленная переменная занимает одно машинное слово. Сигнальные состояния битов с 0-го по 14-ый представляют цифровые разряды (позиции) числа. Сигнальное состояние 15-го бита представляет знак (sign, S). Сигнальное состояние «0» означает, что число положительное, сигнальное состояние «1» обозначает отрицательное число. Отрицательное число представляется в дополнительном коде (в форме дополнения до двух). Допустимая область значений чисел: от +32 767 (7FFF_{hex}) до -32 768 (8000_{hex}).

DINT (двойное целое число)

Переменная типа DINT хранится как целое число (32-битное число с фиксированной запятой). Целое сохраняется в DINT-переменной, когда оно превышает 32 767 или меньше -32 768, или когда число предваряется идентификатором типа L#.

Под переменную типа DINT отводится двойное слово. Сигнальные состояния битов с 0-го по 30-ый представляют цифровые позиции числа. Знак хранится в 31-м бите.

Бит 31, установленный в «0», обозначает положительное число; если его значение «1», то данное число отрицательное. Отрицательные числа хранятся в дополнительном коде (дополнение до двух). Область значений чисел:

от +2 147 483 647 (7FFF FFFF_{hex})
до -2 147 483 648 (8000 0000_{hex}).

REAL (вещественный)

Переменная типа REAL представляет дробь и хранится как 32-битное число с плавающей запятой (десятичной точкой). Целое сохраняется как переменная типа REAL при добавлении десятичной точки и нуля.

В экспоненциальном представлении вы можете предварить «e» или «E» целым числом или дробью из семи соответствующих чисел и знака. Цифры, которые расположены за «e» или «E» представляют экспоненту по базе 10. STEP 7 производит преобразование REAL-переменной во внутренне представление числа с плавающей точкой.

Переменные типа REAL разделяются на числа, которые могут быть представлены с полной точностью («нормализованные» числа с плавающей точкой) и ограниченной точностью («ненормализованные» числа с плавающей точкой). Область значений нормализованных чисел с плавающей точкой:

от $-3.402\,823 \times 10^{+38}$ до $-1.175\,494 \times 10^{-38}$
 ± 0
от $+1.175\,494 \times 10^{-38}$ до $+3.402\,823 \times 10^{+38}$.

Ненормализованное число с плавающей точкой может находиться в следующем диапазоне:

от $-1.175\,494 \times 10^{-38}$ до $-1.401\,298 \times 10^{-45}$
и
от $+1.401\,298 \times 10^{-45}$ до $+1.175\,494 \times 10^{-38}$.

CPU серии S7-300 (за исключением CPU 318) не производят расчетов с ненормализованными числами с плавающей точкой. Битовый образ ненормализованного числа интерпретируется как нуль. Если результат попадает в этот диапазон, то он представляется как нуль, и устанавливаются биты статуса OV и OS (переполнение - overflow).

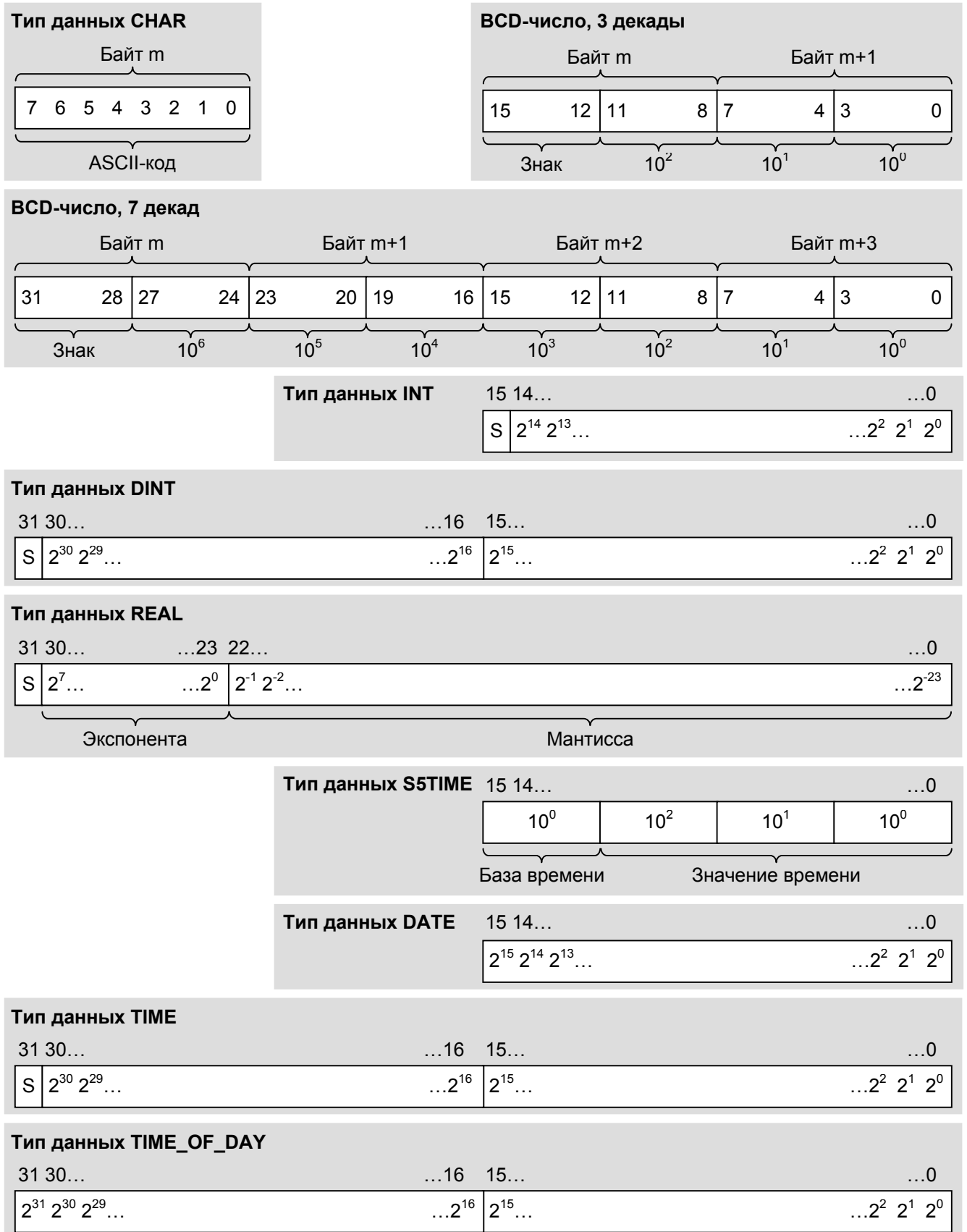


Рисунок 3.14 Структура переменных простых типов данных (S – знак)

Переменная типа REAL внутри состоит из следующих трех компонентов: знака (31-й бит), 8-битной экспоненты по базе 2 (биты с 23-го по 30-ый) и 23-битной мантиссы (биты с 0-го по 22-ой). Знак может принимать значения «0» (положительное) и «1» (отрицательное). Перед сохранением экспоненты к ней добавляется постоянное значение (смещение, +127), поэтому она отображает диапазон значений от 0 до 255. Мантисса представляет дробную часть числа. Целая часть мантиссы не сохраняется, так как она всегда равна 1 (в случае нормализованного числа с плавающей точкой) или 0 (в случае ненормализованного числа с плавающей точкой). В таблице показан внутренний диапазон чисел с плавающей точкой.

Таблица 3.7 Границы диапазона числа с плавающей точкой

Знак	Экспонента	Мантисса	Описание
0	255	Не равна 0	Некорректное число с плавающей точкой
0	255	0	+ бесконечность
0	1 ... 254	Произвольна	Положительное нормализованное число с плавающей точкой
0	0	Не равна 0	Положительное ненормализованное число с плавающей точкой
0	0	0	+ ноль
1	0	0	- ноль
1	0	Не равна 0	Отрицательное нормализованное число с плавающей точкой
1	1 ... 254	Произвольна	Отрицательное ненормализованное число с плавающей точкой
1	255	0	- бесконечность
1	255	Не равна 0	Некорректное число с плавающей точкой

S5TIME

Переменная типа S5TIME используется в базовых языках STL, LAD и FBD для установки таймеров системы SIMATIC. Она занимает одно 16-битное слово с 1 + 3 декадами.

Время устанавливается в часах (hours), минутах (minutes), секундах (seconds) и миллисекундах (milliseconds). STEP 7 производит преобразование во внутренне представление, которое является BCD-числом в диапазоне 000 ... 999. Интервалы времени могут принимать следующие значения: 10 мс (0000), 100 мс (0001), 1 с (0010) и 10 с (0011). Длительность складывается из временного интервала и значения времени.

Примеры:

S5TIME#500ms (= 0050_{hex})

S5T#2h46m30s (= 3999_{hex})

DATE (Дата)

Переменная типа DATE хранится в машинном слове как число с фиксированной точкой без знака. Содержимое переменной соответствует количеству дней, начиная с 01.01.1990. Ее представление показывает год, месяц и день, разделенные дефисом.

Примеры:

DATE#1990-01-01 (=0050_{hex})

D#2168-12-31 (=FF62_{hex})

TIME (Время)

Переменная типа TIME резервирует одно двойное слово. Ее представление содержит информацию о днях (d), часах (h), минутах (m), секундах (s) и миллисекундах (ms), отдельные элементы этих данных могут быть опущены. Содержимое переменной интерпретируется в миллисекундах (ms) и хранится как 32-битное число с фиксированной точкой со знаком.

Примеры:

TIME#24d20h31m23s647ms (= 7FFF_FFFF_{hex})

TIME#0ms (= 0000_0000_{hex})

T#-24d20h31m23s648ms (= 8000_0000_{hex})

Для типа TIME также возможно «десятичное представление», например, TIME#2.25h или T#2.25h.

Примеры:

TIME#0.0h (= 0000_0000_{hex})

TIME#24.855134d (= 7FFF_FFFF_{hex})

TIME_OF_DAY (Время суток)

Переменная типа данных TIME_OF_DAY резервирует для себя одно двойное слово. Она содержит количество миллисекунд с начала суток (со времени 00:00) в виде числа с фиксированной точкой без знака. Ее представление содержит информацию о часах, минутах и секундах, разделенных двоеточием. Миллисекунды, которые следуют за секундами, отделены от них десятичной точкой. Миллисекунды могут отсутствовать.

Примеры:

TIME_OF_DAY#00:00:00 (= 0000_0000_{hex})

TOD#23:59:59.999 (= 0526_5BFF_{hex})

3.5.5 Сложные типы данных

STEP 7 определяет следующие четыре сложных типа данных:

- DATE_AND_TIME (DT, Дата и время)
Дата и время (в формате BCD-числа);
- STRING (Строка)
Строка литер длиной до 254 знаков;
- ARRAY (Массив)
Переменная-массив (совокупность переменных одного типа);
- STRUCT (Структура)
Переменная-структура (совокупность переменных разных типов).

Типы данных предопределяются пользователем при их использовании: задается длина в типе STRING (строка литер), сочетание и размер в типах ARRAY и STRUCT (структура).

Таблица 3.8 Примеры описания переменных типов DT и STRING

Name (Имя)	Type (Тип)	Initial Value (Начальное значение)	Comments (Комментарии)
Date1	DT	DT#1990-01-01- 00:00:00	Минимальное значение переменной типа DT
Date2	DATE_ AND_ TIME	DATE_AND_TIME# 2089-12-31- 23:59:59.999	Максимальное значение переменной типа DT
First_Name	STRING[10]	'Jack'	Переменная типа STRING, определены 4 из 10 литер
Last_Name	STRING[7]	'Daniels'	Переменная типа STRING, определены все 7 литер
NewLine	STRING[2]	'\$R\$L'	Переменная типа STRING, определены специальные литеры
BlankString	STRING[16]	''	Переменная типа STRING, не определена

Переменные сложных типов могут быть объявлены (описаны) только в глобальных блоках данных, в экземплярных блоках данных, как временные локальные данные или как параметры блока.

При работе с параметрами блоков переменные сложных типов могут быть применены только целиком (использование ее компонентов недопустимо).

Для обработки переменных типов DT и STRING, например, извлечения даты и преобразования в представление типа DATE или объединения двух символьных строк в одну переменную, используются IEC-функции. Эти функции являются загружаемыми стандартными блоками FC, которые можно найти в *Стандартной библиотеке (Standard Library)* в программе *Блоки IEC-функций (IEC Function Blocks)*.

DATE_AND_TIME (дата и время)

Тип данных DATE_AND_TIME представляет формат времени, состоящий из даты и времени суток. Допускается использование аббревиатуры DT вместо полного названия типа DATE_AND_TIME.

Отдельные компоненты DT-переменной закодированы в формате ASCII (рисунок 3.15).

STRING (строка символов)

Тип данных STRING представляет строку символов (литер), содержащую до 254 литер. Вы должны определить максимально допустимое количество знаков и указать его в квадратных скобках после ключевого слова STRING.

Длину строки можно не определять, опустив квадратные скобки со значением длины строки. При этом редактор будет использовать максимальную длину (254 байта). В случае функций FC редактор не допускает определения длины или требует стандартного размера в 254 байта.

Переменная типа STRING занимает памяти на два байта больше, чем объявленная максимальная длина строки.

Предустановка выполняется с использованием символов в формате ASCII, заключенных в одинарные кавычки (апострофы), или префикса в виде знака доллара в случае специальных символов (обратитесь к разделу о типе данных CHAR).

Если начальное (initial) или предустановленное (pre-assigned) значение короче объявленной максимальной длины, оставшиеся литерные ячейки не резервируются. Когда обработка переменной типа STRING завершена (переменная в режиме постпроцесса), в расчет принимаются только ячейки, в текущий момент занятые литерами. В качестве начального значения также возможно определить «пустую строку». На рисунке 3.15 показана структура переменной типа STRING.

Формат данных DT

Байт n	Year (Год)	0 ... 99
Байт n+1	Month (Месяц)	1 ... 12
Байт n+2	Day (День)	1 ... 31
Байт n+3	Hour (Час)	0 ... 23
Байт n+4	Minute (Минута)	0 ... 59
Байт n+5	Second (Секунда)	0 ... 59
Байт n+6	ms (мс)	0 ... 999
Байт n+7		Week-day

Week-day (День недели)

от 1 = Sunday (Воскресение)

до 7 = Saturday (Суббота)

Формат данных STRING

Байт n	Максимальная длина	(k)
Байт n+1	Текущая длина	(m)
Байт n+2	литера 1	} Текущая длина
Байт n+3	литера 2	
Байт	
Байт n+m+1	литера m	
Байт	
Байт n+k+1	...	} Максимальная длина

Рисунок 3.15 Структура переменных типов DT и STRING

ARRAY (массив)

Тип данных ARRAY представляет массив, состоящий из фиксированного числа однотипных элементов.

В квадратных скобках после наименования типа ARRAY вы должны задать область индексов поля. Начальное значение слева должно быть меньше или равно завершающему значению справа. Оба индекса являются числами типа INT из области значений $-32\,768 \dots +32\,767$. Наибольшая размерность поля – 6, границы «измерений» разделяются запятой.

Тип данных отдельного компонента поля располагается в строке под определением типа данных ARRAY. Допускаются все типы данных за исключением ARRAY; также это может быть пользовательский тип данных.

Предустановка

На этапе описания вы можете предустановить значения отдельных компонентов поля (не как в случае параметров блока в функции, входных/выходных параметров в функциональном блоке или временной переменной). Тип данных предустанавливаемого значения должен быть тем же, что и тип поля.

Вы можете не устанавливать предварительно все компоненты поля. Если количество предустановленных значений меньше числа компонентов поля, то предварительно устанавливаются только первые компоненты. Количество предустановленных компонентов не должно превышать числа компонентов поля. Предустановленные значения разделяются запятыми. Предварительно присваиваемое элементам массива

одинаковое значение указывается в круглых скобках, перед открывающей скобкой ставится коэффициент повторения.

Таблица 3.9 Примеры описания массивов

Name (Имя)	Type (Тип)	Initial Value (Начальное значение)	Comments (Комментарии)
Meas. val.	ARRAY[1..24]	0.4, 1.5, 11 (2.6, 3.0)	Массив с 24 элементами типа REAL
	REAL		
TOD	ARRAY[-10..10]	21 (TOD#08:30:00)	Массив TOD с 21 элементом
	TIME_OF_DAY		
Result	ARRAY[1..24,1..4]	96 (L#0)	Двумерный массив с 96 элементами
	DINT		
Char.	ARRAY[1..2,3..4]	2 ('a'), 2 ('b')	Двумерный массив с 4 элементами
	CHAR		

Приложение

Вы можете применить массив как составную переменную (целиком) в параметрах блоков типа ARRAY с однородной структурой или в параметрах блоков типа ANY (Любой). Например, вы можете скопировать содержимое переменной-массива с помощью системной функции SFC 20 BLKMOV. Также вы можете определить отдельные компоненты массива в параметре блока, если он того же типа, что и компоненты.

Если отдельные компоненты поля элементарных типов данных, то вы можете обрабатывать их с помощью «нормальных» функций LAD и FBD.

Доступ к компонентам массива осуществляется по имени массива и индексу, указываемому в квадратных скобках. В LAD и FBD значение индекса является фиксированным и не может меняться во время исполнения (переменные индексы не допускаются).

Многомерные массивы, размерности

Массивы могут быть максимум 6-мерными. Многомерные массивы аналогичны одномерным массивам. На этапе декларирования границы измерений записываются в квадратных скобках и разделяются запятыми.

Структура переменных

Переменная типа ARRAY всегда начинается на границе машинного слова, то есть с байта с четным адресом. Массив занимает область памяти до границы следующего слова.

Компоненты типа BOOL (Логический) начинаются с младшего значащего бита; компоненты типов BYTE (Байт) и CHAR (Литерный) начинаются с правого байта. Отдельные компоненты расположены упорядоченно.

В многомерных массивах компоненты хранятся построчно (по размерностям), начиная с первой размерности. В случае однобитных и однобайтных компонентов новая размерность всегда начинается со следующего байта. Если компоненты относятся к другим типам данных, то новая размерность всегда начинается со следующего слова (в следующем четном байте).

STRUCT (структура)

Тип данных STRUCT представляет структуру данных, состоящую из фиксированного числа компонентов, каждый из которых может быть отличным от других типа данных.

Вы должны определить отдельные компоненты структуры и их типы данных под строкой с именем переменной и ключевым словом STRUCT. Использовать в построении рассматриваемого типа данных можно любые типы, включая другие структуры.

Таблица 3.10 Пример описания структуры

Name (Имя)	Type (Тип)	Initial Value (Начальное значение)	Comments (Комментарии)
MotCont	STRUCT		Простая переменная структурного типа с 4 компонентами
On	BOOL	FALSE	Переменная MotCont.On типа BOOL
Off	BOOL	TRUE	Переменная MotCont.Off типа BOOL
Delay	S5TIME	S5TIME#5s	Переменная MotCont.Delay типа S5TIME
maxSpeed	INT	5000	Переменная MotCont.maxSpeed типа INT
	END_STRUCT		

Предустановка

На стадии описания вы можете предварительно присвоить значения отдельным компонентам структуры (не как в случае параметров блока в функции, входных / выходных параметров в функциональном блоке или временной переменной). Тип данных предустанавливаемых значений должен совпадать с типом компонентов.

Приложение

Вы можете применить составную переменную (целиком) в параметрах блока типа STRUCT с такой же структурой или в параметрах блока типа ANY (Любой). Например, вы можете скопировать содержимое переменной типа STRUCT при помощи

системной функции SFC 20 BLKMOV. Вы также можете определить отдельный компонент структуры в параметре блока, если параметр относится к тому же типу данных, что и компонент.

Если отдельные компоненты структуры имеют простой тип данных, то вы можете обрабатывать их, применяя «нормальные» функции LAD и FBD.

Доступ к компоненту структуры осуществляется по имени структуры и имени компонента, разделенных точкой.

Структура переменных

Переменные типа STRUCT всегда начинаются с границы машинного слова, то есть с байта по четному адресу. Поэтому отдельные компоненты расположены в памяти в порядке их объявления. STRUCT-переменные занимают область памяти до границы следующего слова.

Компоненты типа BOOL (Логический) начинаются с младшего значащего бита; компоненты типов BYTE (Байтовый) и CHAR (Литерный) начинаются с правого байта. Компоненты других типов начинаются с границы слова.

Вложенная структура – это структура, являющаяся компонентом другой структуры. Возможная глубина вложения структур – 6 вложений. Все компоненты простых типов данных могут быть доступны с помощью «нормальных» функций LAD или FBD. Каждое индивидуальное имя отделяется точкой.

3.5.6 Параметрические типы

Параметрические типы (parameter types) – это типы данных для параметров блоков (таблица 3.11). Спецификации размеров (длины) в таблице указывают на требуемые объемы памяти для параметров функциональных блоков. Вы также можете использовать TIMER и COUNTER в таблице символов в качестве типов данных для таймеров и счетчиков.

3.5.7 Пользовательские типы данных

Пользовательский тип данных (user data type – UDT) соответствует структуре (комбинация компонентов любых типов) с действием на глобальном уровне. Вы можете воспользоваться пользовательским типом данных, если в вашей программе часто фигурирует структурный тип и переменные, или вы хотите структуре данных присвоить имя.

Типы UDT обладают глобальным действием; то есть, они описываются один раз и доступны для использования во всех блоках. UDT могут адресоваться символически; вы должны соответственно назначить абсолютный адрес в таблице символов. Тип данных UDT (в таблице символов) идентичен абсолютному адресу.

Если вы хотите объявить переменную, определенную в виде UDT, при описании назначьте ей тип UDT как в случае «нормального» типа данных. UDT могут адресоваться абсолютно (UDT 0 ... UDT 65 535).

Вы также можете определить UDT для единого типа данных. При программировании блока данных назначьте блоку этот UDT в качестве структуры данных.

Как работать с пользовательскими типами данных показано в примере «Message Frame Data» («Данные фрейма сообщения») в параграфе 24.3 «Краткое описание «Message Frame Data» («Данные фрейма сообщения»)».

Таблица 3.11 Обзор параметрических типов

Параметрический тип	Описание		Примеры фактических адресов
TIMER	Таймер	16 бит	T 15 или символ
COUNTER	Счетчик	16 бит	C 16 или символ
BLOCK_FC	Функция	16 бит	FC 17 или символ
BLOCK_FB	Функциональный блок	16 бит	FB 18 или символ
BLOCK_DB	Блок данных	16 бит	DB 19 или символ
BLOCK_SDB	Блок системных данных	16 бит	SDB 100 или символ
POINTER	DB-указатель	48 бит	P#M10.0 (указатель) P#DB20.DBX22.2 (указатель) MW 20 (указатель) I 1.0 (указатель)
ANY	ANY-указатель	80 бит	P#DB10.DBX0.0 WORD 20 или любая переменная

Программирование структур UDT

Создать определяемый пользователем тип данных вы можете либо в SIMATIC-менеджере, отметив объект *Blocks* (Блоки) и выбрав пункт меню Insert → S7 Block → Data Type (Вставка → Блок S7 → Тип данных), либо в редакторе путем выбора команды меню File → New (Файл → Новый) и ввода «UDTn» в строке «Object Name» («Имя объекта»).

Двойной щелчок на объекте *UDT* в окне программы откроет таблицу описаний, которая выглядит точно так же, как таблица описаний блока данных. UDT программируется как блок данных, с заполнением отдельных строк для имени (Name), типа данных (Type), начального значения (Initial value) и комментариев (Comments). Единственное отличие заключается в том, что нельзя переключиться в просмотр данных (data view). (С UDT вы не создаете никаких переменных, а формируете только коллекцию типов данных; по этой причине здесь нет фактических значений).

Начальные значения, заданные вами в UDT, передаются в переменные при их описании.

Основные функции

Этот раздел книги посвящен описанию функций языков программирования LAD и FBD, которые представляют в определенном смысле «основную функциональность». Эти функции позволят вам программировать PLC на базе контакторов или реле управления.

В контактном плане (ladder diagram, LAD) компоновка контактов в последовательные и параллельные схемы определяет комбинирование бинарных сигнальных состояний. В функциональном плане (function block diagram, FBD) блочные элементы, аналогичные электронным переключающим системам, представляют булевские (логические) функции AND (И) и OR (ИЛИ).

Функции для работы с памятью находят свое отражение в RLO (Result of logic operation, результат логической операции), поэтому, например, возможно сканирование и передача данных дальше, в другие части программы.

Функции перемещения (move) задействованы в обмене значениями отдельных операндов и переменных или в копировании целых областей данных.

Синхронизирующие реле в контакторных управляющих системах являются таймерами в программируемых контроллерах. Таймеры, встроенные в CPU, позволяют вам, к примеру, запрограммировать время ожидания и наблюдения.

Наконец, счетчики могут вести отсчет по возрастанию и по убыванию в интервале значений 0...999.

Настоящий раздел книги рассказывает о функциях, работающих с областями операндов для входов, выходов и памяти меркеров. Входы и выходы представляют собой связующие элементы в процессе или модели предприятия. Память меркеров соответствует дополнительным контакторам, которые хранят двоичные состояния. Следующие разделы книги приводят описание остальных областей операндов, которые вы можете обрабатывать с использованием бинарной логики. По существу, они являются битами данных в блоках глобальных данных, а также битами временных и статических локальных данных.

В главе 5 «Функции по работе с памятью» вы найдете примеры программирования с операциями бинарной логики и функциями по работе с памятью. Глава 8 «Счетчики» содержит пример, демонстрирующий таймеры и счетчики. В обоих случаях примеры представлены в виде функций FC без параметров блоков. Те же примеры в функциональных блоках (FB) с параметрами блоков вы можете найти в главе 19 «Параметры блоков».

4 Бинарные логические операции (binary logic)

Последовательные и параллельные схемы (LAD), функции AND, OR и исключающее OR (FBD); отрицание (инвертирование); (принимая во внимание тип датчика)

5 Функции для работы с памятью (memory functions)

Катушки LAD; блочные элементы FBD; выводы средней линии (коннекторы); оценка фронта (проверка наличия фронта сигнала); пример с ленточным транспортером конвейера

6 Функции перемещения (move functions)

Блочный элемент MOVE, системные функции для передачи данных

7 Таймеры (timers)

Пять различных видов запуска таймера, сброс и сканирование таймера; IEC-таймеры

8 Счетчики (counters)

Установка счетчика; счет по возрастанию и по убыванию (прямой и обратный счет); сброс и сканирование счетчика; IEC-счетчики; пример подачи питания

Глава 4

Операции бинарной логики

Содержание главы 4

4	<u>Операции бинарной логики</u>	4
4.1	<u>Последовательные и параллельные схемы (LAD)</u>	4
4.1.1	<u>NO-контакт и NC-контакт</u>	4
4.1.2	<u>Последовательные схемы</u>	7
4.1.3	<u>Параллельные схемы</u>	7
4.1.4	<u>Комбинации операций бинарной логики</u>	8
4.1.5	<u>Инвертирование результата логической операции</u>	10
4.2	<u>Операции бинарной логики (FBD)</u>	11
4.2.1	<u>Элементарные операции бинарной логики</u>	11
4.2.2	<u>Комбинации операций бинарной логики</u>	17
4.2.3	<u>Инвертирование результата логической операции</u>	18
4.3	<u>Действия в зависимости от типа датчика</u>	19

4 Операции бинарной логики

4.1 Последовательные и параллельные схемы (LAD)

Бинарные сигнальные состояния группируются в LAD (контактные планы) посредством последовательных (series) и параллельных (parallel) соединений контактов. Последовательное соединение соответствует функции AND (И), а параллельное соединение – функции OR (ИЛИ). Вы будете использовать контакты для проверки сигнальных состояний следующих двоичных операндов:

- Входные и выходные биты, память меркеров;
- Таймеры и счетчики;
- Биты глобальных данных;
- Биты временных локальных данных;
- Биты слова состояния (оценка результатов вычислений).

Вы можете обратиться к операнду через контакт, используя либо абсолютный, либо символический адрес. LAD применяет только контакты NO (сканируют с ожиданием сигнального состояния «1») и контакты NC (сканируют сигнальное состояние «0»).

Цепь, или звено (rung), может состоять из единственного контакта или же большого числа соединенных контактов. Звено всегда должно быть завершено, к примеру, катушкой (coil). Катушка управляет двоичным операндом с помощью RLO («power flow», «протекание тока») цепи.

Примеры, показанные в этой главе, также имеются на дискете, поставляемой с книгой, в функциональном блоке FB 104 программы «Basic Functions» («Основные функции»), библиотека «LAD_Book». Что касается пошагового программирования, то элементы для двоичных логических операций вы можете найти в каталоге программных элементов (Program Element Catalog) – с помощью команды меню View → Catalog (Вид → Каталог) [Ctrl + K] или Insert → Program Elements (Вставка → Программные элементы) – в разделе «Bit Logic» («Битовая логика»).

4.1.1 NO-контакт и NC-контакт

Чтобы объяснить комбинации битовой логики в контактном плане, ниже мы будем использовать графические представления и термины «contact closed» («контакт замкнут»), «power flowing» («протекание тока») и «coil energized» («катушка под напряжением», «через/в катушке течет ток» или «катушка возбуждена»). Если «ток» («power») течет в точке контактного плана, то это означает, что в этой точке битовая логика выполняется. Результатом логической операции (RLO) является «1». Если «ток» протекает в одиночной катушке, то она находится под напряжением. Соответствующий двоичный операнд имеет сигнальное состояние «1».

LAD использует два вида контактов для сканирования битовых операндов: NO-контакт и NC-контакт.

**Нормально разомкнутый контакт
(Normally-open, NO)**

Бинарный операнд



**Нормально замкнутый контакт
(Normally-closed, NC)**

Бинарный операнд



Нормально разомкнутый контакт (NO-контакт)

Нормально разомкнутый контакт соответствует сканированию с ожиданием сигнального состояния «1». Если сканированный бинарный операнд имеет сигнальное состояние «1», то NO-контакт активирован, замыкается и «ток течет».

На рисунке 4.1 (верхняя схема) показан пример, где датчик (sensor) S1 подключен к входу I 1.0 и сканируется NO-контактом. Если датчик S1 разомкнут, то вход I 1.0 содержит «0», и ток через NO-контакт не проходит. Контакт (contactor) K1, управляемый выходом Q 4.0, не включен.

Если датчик S1 активирован, то вход I 1.0 имеет сигнальное состояние «1». Ток от левой несущей (питающей шины) течет через NO-контакт в катушку, и контактор K1, подключенный к выходу Q 4.0, активируется. NO-контакт сканирует вход с ожиданием сигнального состояния «1» и затем замыкается, независимо от того, каким контактом, NO или NC, при входе является датчик.

Нормально замкнутый контакт (NC-контакт)

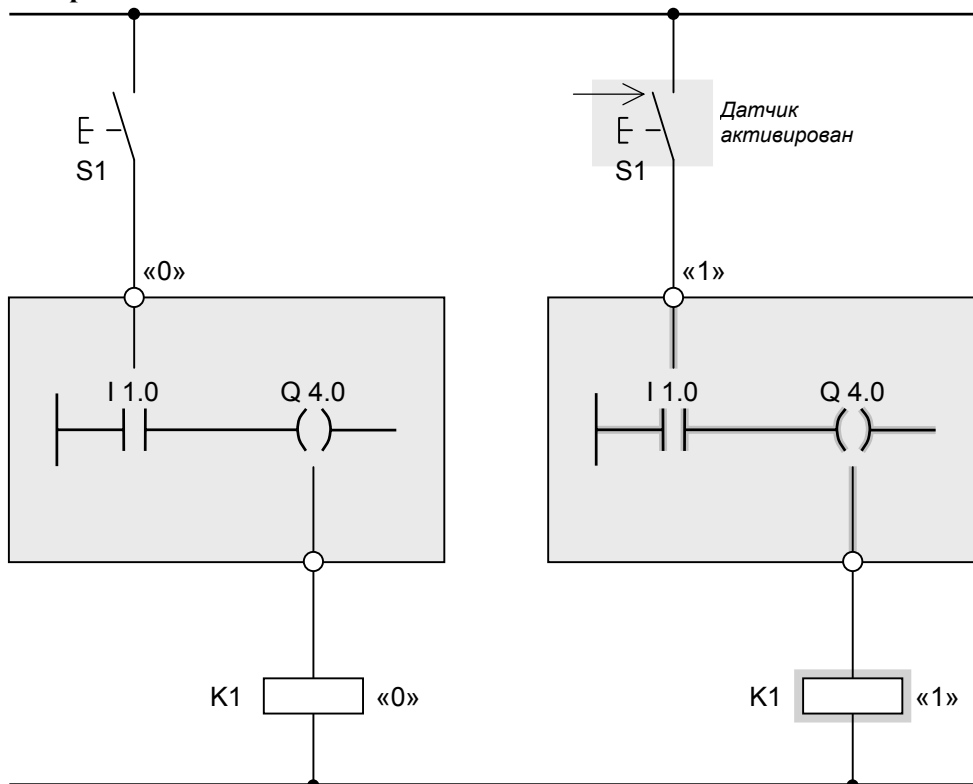
Ток протекает через NC-контакт, если бинарный операнд имеет сигнальное состояние «0». Если сигнальное состояние «1», то NC-контакт «размыкается», и протекание тока прерывается.

В примере на рисунке 4.1 (нижняя схема) ток течет через NC-контакт, если датчик S2 незамкнут (вход I 1.1 имеет сигнальное состояние «0»). Ток также течет в катушке и возбуждает контактор K2 на выходе Q 4.1.

Если датчик S2 активирован, то вход I 1.1 имеет сигнальное состояние «1», и NC-контакт размыкается. Протекание тока прерывается, и с контактора K2 снимается нагрузка.

NC-контакт проверяет вход на наличие нулевого сигнального состояния и пребывает затем в замкнутом состоянии, несмотря на то, каким контактом на входе является датчик, NO- или NC-контактом (обратитесь также к параграфу 4.3 «Действия в зависимости от типа датчика»).

Как работает NO-контакт



Как работает NC-контакт

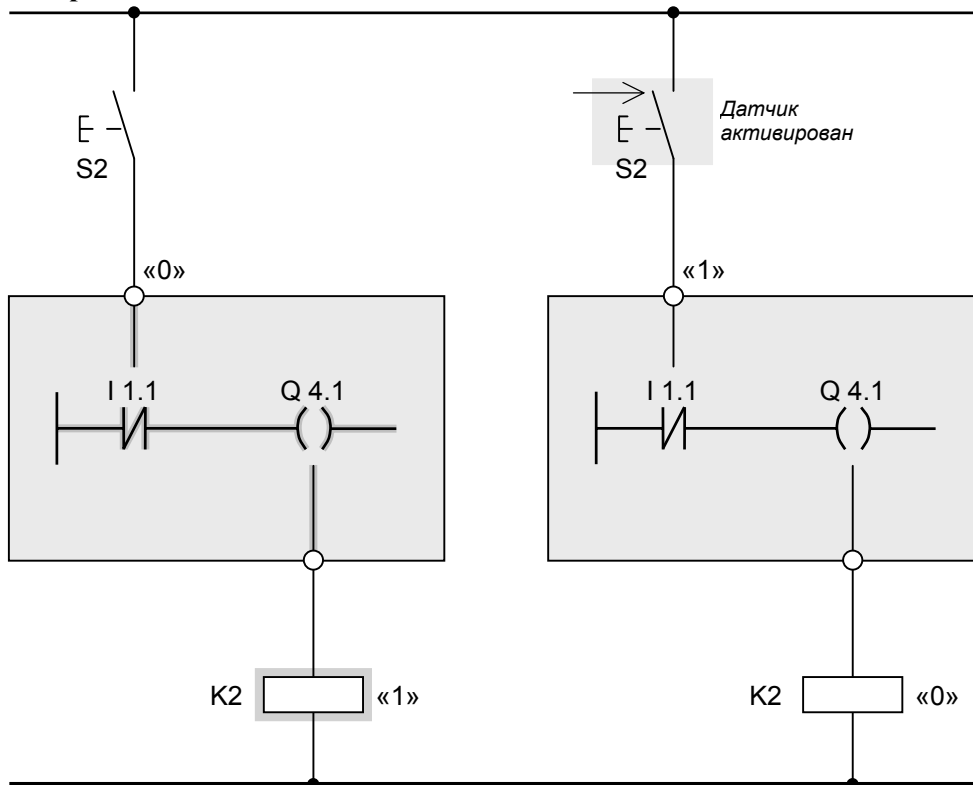


Рисунок 4.1 NO-контакты и NC-контакты

4.1.2 Последовательные схемы

В последовательных схемах два или более контактов соединены последовательно. Ток в последовательной схеме течет, когда все контакты замкнуты.

На рисунке 4.2 показаны типичные последовательные схемы. В сети 1 (network 1) последовательная схема содержит три контакта; любые бинарные операнды могут сканироваться. Все контакты являются НО-контактами. Если все соответствующие операнды имеют сигнальное состояние «1» (то есть если НО-контакты активированы), то в цепи ток течет к катушке. Операнд, управляемый катушкой, устанавливается в «1». Во всех других случаях ток не течет, и операнд *Coil1* (*Катушка1*) сбрасывается в «0».

Сеть 2 (Network 2) демонстрирует последовательный план с одним НС-контактом. Ток течет через НС-контакт, если соответствующий операнд находится в сигнальном состоянии «0» (то есть НС-контакт не активирован). Таким образом, ток протекает через последовательную схему в случае, если операнд *Contact4* (*Контакт4*) имеет сигнальное состояние «1», а операнд *Contact5* (*Контакт5*) имеет сигнальное состояние «0».

4.1.3 Параллельные схемы

Когда два или более контактов скомпонованы один под другим, мы называем это параллельной схемой. Ток протекает через параллельную схему, если один из контактов замкнут.

На рисунке 4.2 показаны типичные параллельные схемы. В сети 3 (Network 3) параллельный план состоит из трех контактов; сканироваться может любой операнд. Все контакты являются НО-контактами. Если один из операндов имеет сигнальное состояние «1», то по схеме к катушке течет ток. Операнд, управляемый катушкой, устанавливается в «1». Если все сканируемые операнды находятся в сигнальном состоянии «0», то ток к катушке не проходит, и операнд *Coil3* (*Катушка3*) сбрасывается в «0».

Сеть 4 (Network 4) демонстрирует параллельный план с одним НС-контактом. Ток течет через НС-контакт, если соответствующий операнд – «0», то есть ток протекает через параллельную схему в случае, если операнд *Contact4* (*Контакт4*) имеет сигнальное состояние «1», или операнд *Contact5* (*Контакт5*) имеет сигнальное состояние «0».

В LAD вы также можете запрограммировать ветвь в середине цепи, как это изображено в примере на рисунке 4.3, сеть 8 (Network 8). У вас может быть, таким образом, параллельная ветвь, которая не начинается на левой несущей (шине питания). Использование программных элементов LAD в такой ветви ограничено; этот случай рассмотрен в соответствующих главах. «Открытая» («разомкнутая», «орен») параллельная цепь называется «Т-ветвь».

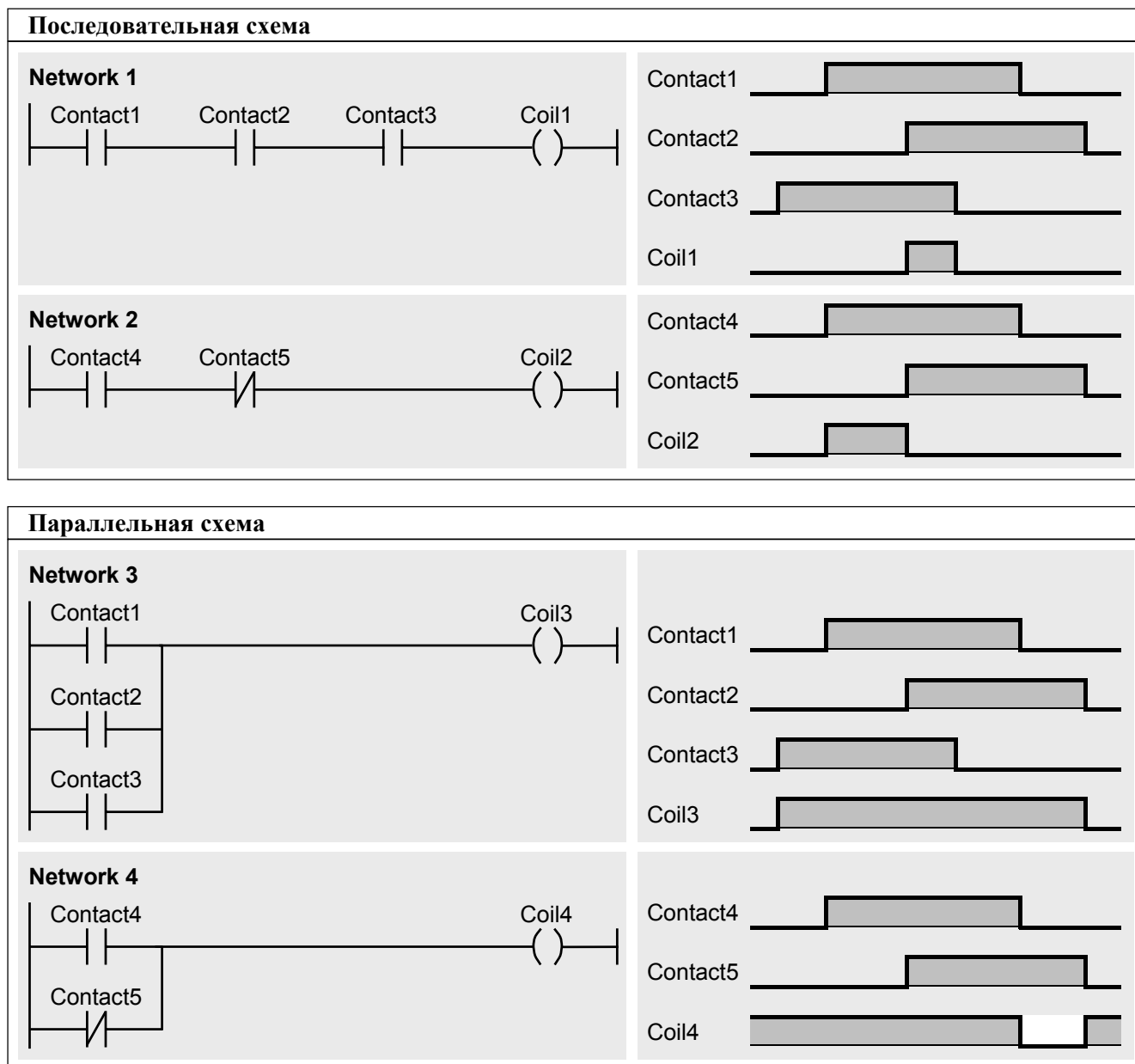


Рисунок 4.2 Последовательные и параллельные схемы

4.1.4 Комбинации операций бинарной логики

Вы можете объединять последовательные и параллельные схемы, например, путем группировки нескольких последовательных схем в параллельные планы или нескольких параллельных схем в последовательные планы. Вы можете комбинировать последовательные и параллельные схемы, даже когда оба типа являются сложными изначально (рисунок 4.3).

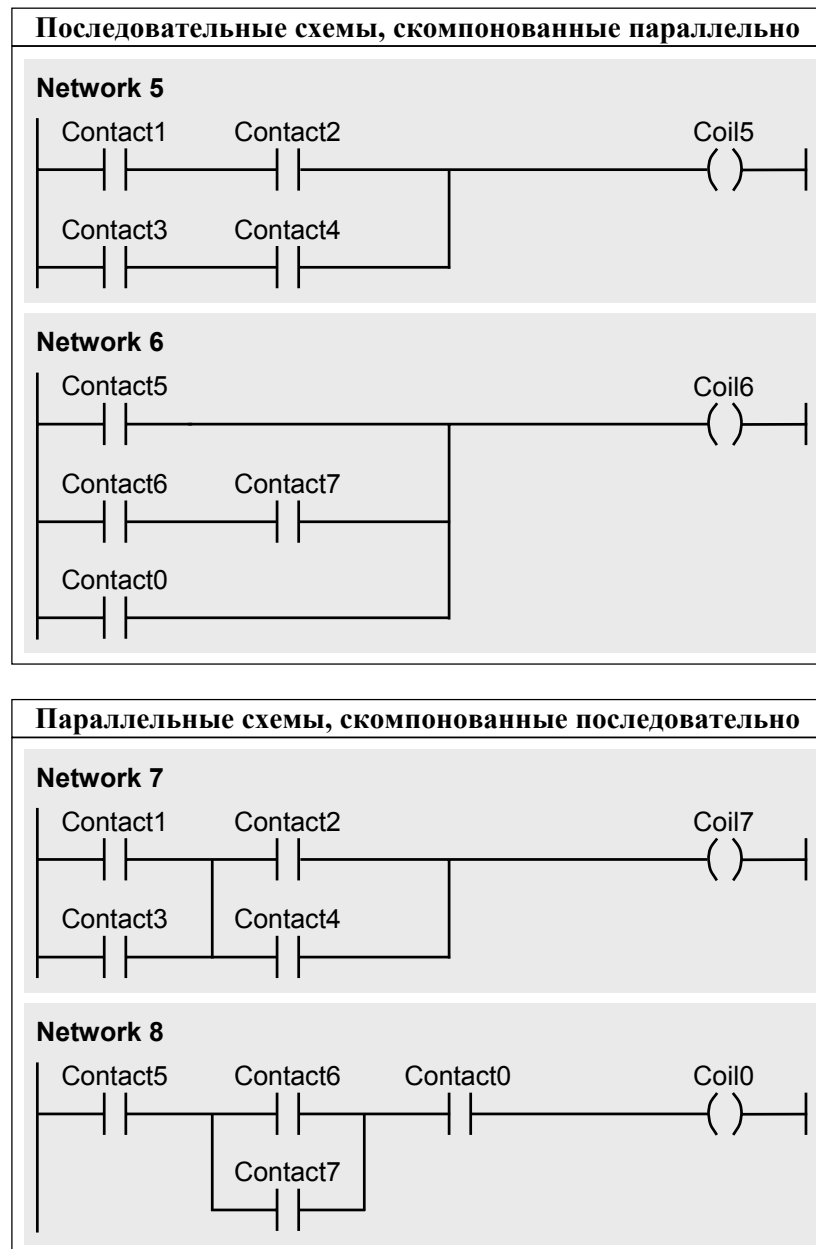


Рисунок 4.3 Последовательные и параллельные схемы в комбинации

Параллельное соединение последовательных схем

Наряду с контактами вы можете также скомпоновать последовательные схемы одна под другой. На рисунке 4.3 приведены два примера. В сети 5 (Network 5) ток течет в катушке (достигает ее), если *Contact1* (Контакт1) и *Contact2* (Контакт2) замкнуты или *Contact3* (Контакт3) и *Contact4* (Контакт4) находятся в замкнутом состоянии. В нижней цепи (сеть 6, Network 6) ток течет, если замкнуты *Contact5* (Контакт5) или *Contact6* (Контакт6) и *Contact7* (Контакт7) или *Contact0* (Контакт0).

Последовательное соединение параллельных схем

Как и в случае контактов, вы также можете группировать параллельные схемы в последовательности. Рисунок 4.3 содержит два подобных примера. В сети 7 (Network 7) ток течет в катушке (достигает ее), если либо *Contact1* (Контакт1) или *Contact3* (Контакт3) и либо *Contact2* (Контакт2) или *Contact4* (Контакт4) замкнуты. Чтобы позволить току течь в нижнем примере (сеть 8, Network 8), *Contact5* (Контакт5), *Contact0* (Контакт0) и либо *Contact6* (Контакт6), либо *Contact7* (Контакт7) должны быть замкнуты.

4.1.5 Инвертирование результата логической операции

NOT-контакт инвертирует результат логической операции (RLO). Можно использовать этот контакт, чтобы, к примеру, передать в катушку инвертированное значение последовательной цепи (рисунок 4.4, сеть 9, Network 9). Ток будет течь в катушке только тогда, когда нет тока в контакте NOT, то есть если разомкнут *Contact1* (Контакт1) или *Contact2* (Контакт2) (обратите внимание в рисунке на смежную диаграмму импульсов).

То же самое применимо по аналогии для сети 10 (Network 10), в которой NOT-контакт вставлен после параллельной цепи. Здесь *Coil10* (Катушка10) устанавливается, если ни один из контактов не замкнут.

Вы можете вставить NOT вместо другого контакта в цепь, которая начинается на левой несущей (питающей шине). Добавление NOT-контакта в параллельную ветвь, начинающуюся в середине цепи, не допускается.

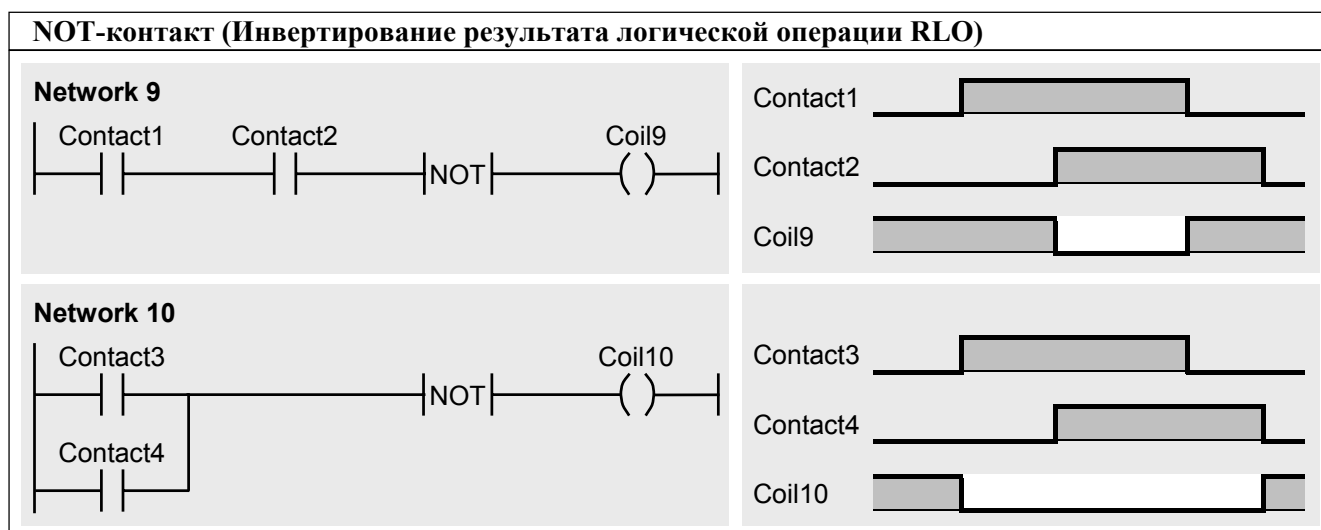


Рисунок 4.4 Примеры включений NOT-контакта

4.2 Операции бинарной логики (FBD)

В FBD логические операции, выполняемые над двоичными сигнальными состояниями, принимают форму функций AND (И), OR (ИЛИ) и Exclusive OR (Исключающее ИЛИ). Операнды, чьи сигнальные состояния вы хотите сканировать и комбинировать, подаются (записываются) на входы этих функций. Вы можете сканировать следующие операнды:

- Входные и выходные биты, меркеры (рассматриваются в данном разделе);
- Таймеры и счетчики;
- Биты глобальных данных;
- Биты временных локальных данных;
- Биты статических локальных данных;
- Биты слова состояния (результаты оценки и вычислений).

Каждый бинарный операнд может быть адресован абсолютно или символически. При сканировании бинарного операнда, или внутри схемы бинарной логики (функционального плана), вы можете инвертировать результат логической операции с помощью символа инвертирования (отрицания). Этот символ представляет собой кружок.

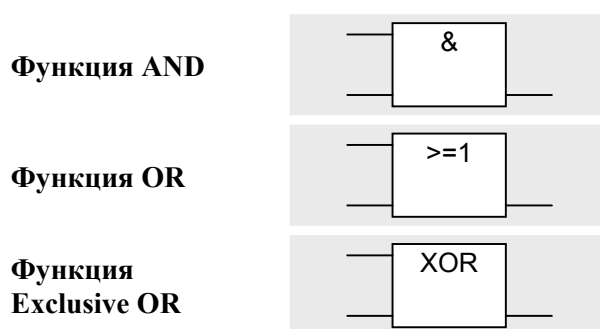
В FBD для каждого сегмента (сети, схемы) программируется одна схема бинарной логики (binary logic circuit). Логическая схема может состоять из одного или очень большого числа соединенных между собой функциональных элементов. Логическая схема, или логическая операция (logic operation), всегда должна быть завершена, например, оператором присваивания. Присваивание воздействует на бинарный операнд с помощью результата логической операции.

Примеры, представленные в этой главе, также имеются на прилагаемой дискете, в функциональном блоке FB 104 программы «Basic Functions» («Основные функции»), в библиотеке «FBD_Book». Для пошагового программирования элементы, составляющие операции бинарной логики, вы можете найти в каталоге программных элементов (Program Element Catalog) – с помощью команды меню View → Catalog (Вид → Каталог) [Ctrl + K] или Insert → Program Elements (Вставка → Программные элементы) – в разделе «Bit Logic» («Битовая логика»).

4.2.1 Элементарные операции бинарной логики

FBD использует бинарные функции AND (И), OR (ИЛИ) и Exclusive OR (Исключающее ИЛИ). Все функции могут иметь (теоретически) любое количество функциональных входов (входов функции). Если вход ведет напрямую к функциональному элементу, то сигнальное состояние сканируемого операнда непосредственно используется в логической операции; если вход снабжен знаком отрицания (кружок),

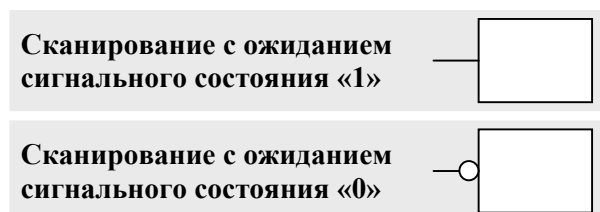
то сигнальное состояние сканируемого операнда инвертируется перед выполнением логической операции (см. ниже).



Теоретически количество бинарных функций и область действия бинарной функции неограничены. Однако, на практике границы регулируются размерами блока или объемом главной памяти CPU.

Сканирование и назначение сигнальных состояний

Перед тем как бинарные функции выполняют логические операции над сигнальными состояниями, они сканируют (считывают) бинарные операнды на своих входах. Операнд может сканироваться с ожиданием «1» или «0». Если сканируется с ожиданием «1», то вход функции направлен непосредственно в блочный элемент. Сканирование с ожиданием «0» распознается по символу отрицания на входе функции.



Сканирование с ожиданием «1» генерирует результат сканирования (scan result) «1», когда сигнальное состояние сканированного двоичного операнда «1». Сканирование выводит результат сканирования «0», когда сигнальное состояние бинарного операнда «0». Сканирование с ожиданием сигнального состояния «0» инвертирует результат сканирования, то есть результат сканирования равен «1», когда статус сканированного бинарного операнда «0». Бинарные функции комбинируют *результат сканирования*, поданный «непосредственно» на блочный элемент. С точки зрения функциональности эти два метода сканирования бинарных операндов позволяют вам использовать NO-контакты и NC-контакты идентично.

Приведем пример: «0» подается на входной модуль неактивированного NO-контакта (рисунок 4.5). Сканирование с ожиданием сигнального состояния «1» передает это состояние в функциональный блочный элемент (блочный элемент функции). Для выполнения того же для NC-контакта вы должны сканировать вход с помощью NC-

контакт с ожиданием сигнального состояния «0» (должен иметься кружок для инвертирования). Сигнальное состояние «1», подаваемое на входной модуль неактивированного NC-контакта, затем конвертируется в сигнальное состояние «0» в функциональном блочном элементе.

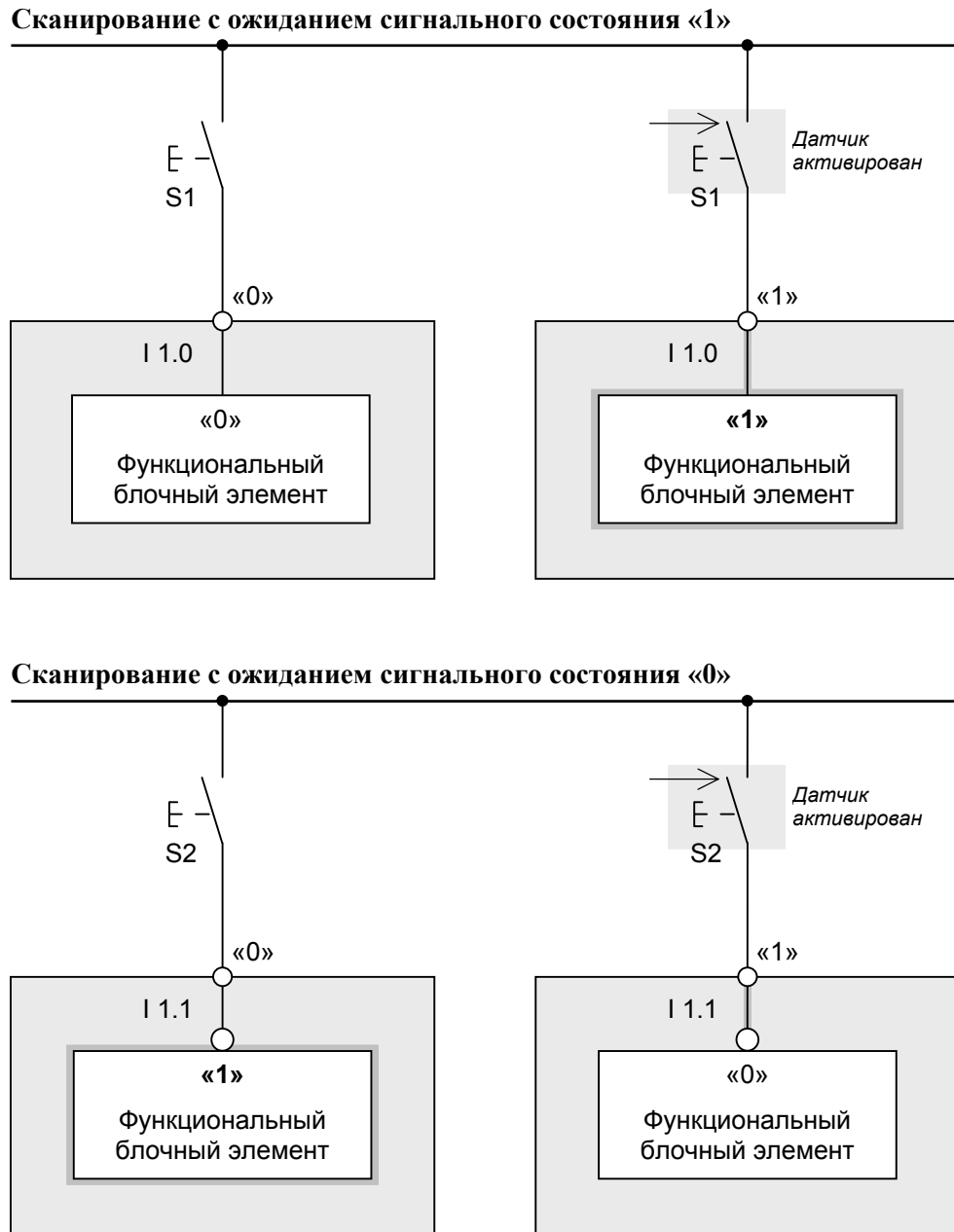


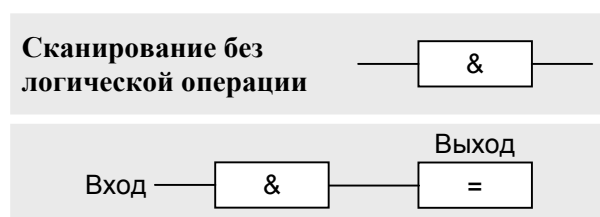
Рисунок 4.5 Сканирование с ожиданием сигнального состояния «1» и «0»

Если теперь активировать оба контакта, и NC, и NO, функциональный блочный элемент в обоих случаях покажет сигнальное состояние «1». Дополнительную информацию вы можете найти в параграфе 4.3 «Действия в зависимости от типа датчика».

Вы всегда должны соединять (с каким-либо элементом) выход бинарной функции; в простейшем случае можно просто соединить выход с блочным элементом Assign

(Присваивание) (обратитесь также к главе 5 «Функции по работе с памятью»). С помощью данного результата логической операции вы также можете запустить таймер, выполнить цифровую операцию, вызвать блок и так далее. В следующей главе содержится вся необходимая информация.

Чтобы присвоить сигнальное состояние бинарного операнда непосредственно другому бинарному операнду, не выполняя никаких дополнительных логических операций, например, подсоединить вход напрямую к выходу, обычно используется функция AND (И), хотя допустимо было бы применить OR (ИЛИ) или Exclusive OR (Исключающее ИЛИ).



Просто выберите функцию AND (И), подключив только один вход функции и удалив другой.

Функция AND (И)

Функция AND комбинирует два бинарных состояния друг с другом и генерирует результат логической операции (RLO) «1», если оба состояния (оба сканированных результата) равны «1». Если функция AND имеет несколько входов, результаты сканирования всех входов должны быть равны «1», чтобы совокупный RLO был «1». В остальных случаях функция AND на своем функциональном выходе выдает RLO «0».

Рисунок 4.6 содержит пример функции AND. В сети 1 (Network 1) функция AND имеет три входа, каждый из которых может быть подключен к любому бинарному операнду. Все операнды сканируются с ожиданием сигнального состояния «1», поэтому сигнальные состояния операндов непосредственно (напрямую) объединяются по AND. Если все сканированные операнды имеют сигнальное состояние «1», то функция AND установит операнд *Output1 (Выход1)* в «1» посредством блочного элемента Assign (Присваивание) (обратитесь к следующей главе). В остальных случаях условие AND не выполняется, и операнд *Output1 (Выход1)* сбрасывается в «0».

Сеть 2 (Network 2) показывает функцию AND с инвертированным входом. Инвертирование (отрицание) входа обозначается кружком. Результат сканирования для инвертированного операнда равен «1», если этот операнд равен «0», то есть в примере условие AND выполняется, когда операнд *Input4 (Вход4)* равен «1» и операнд *Input5 (Вход5)* равен «0».

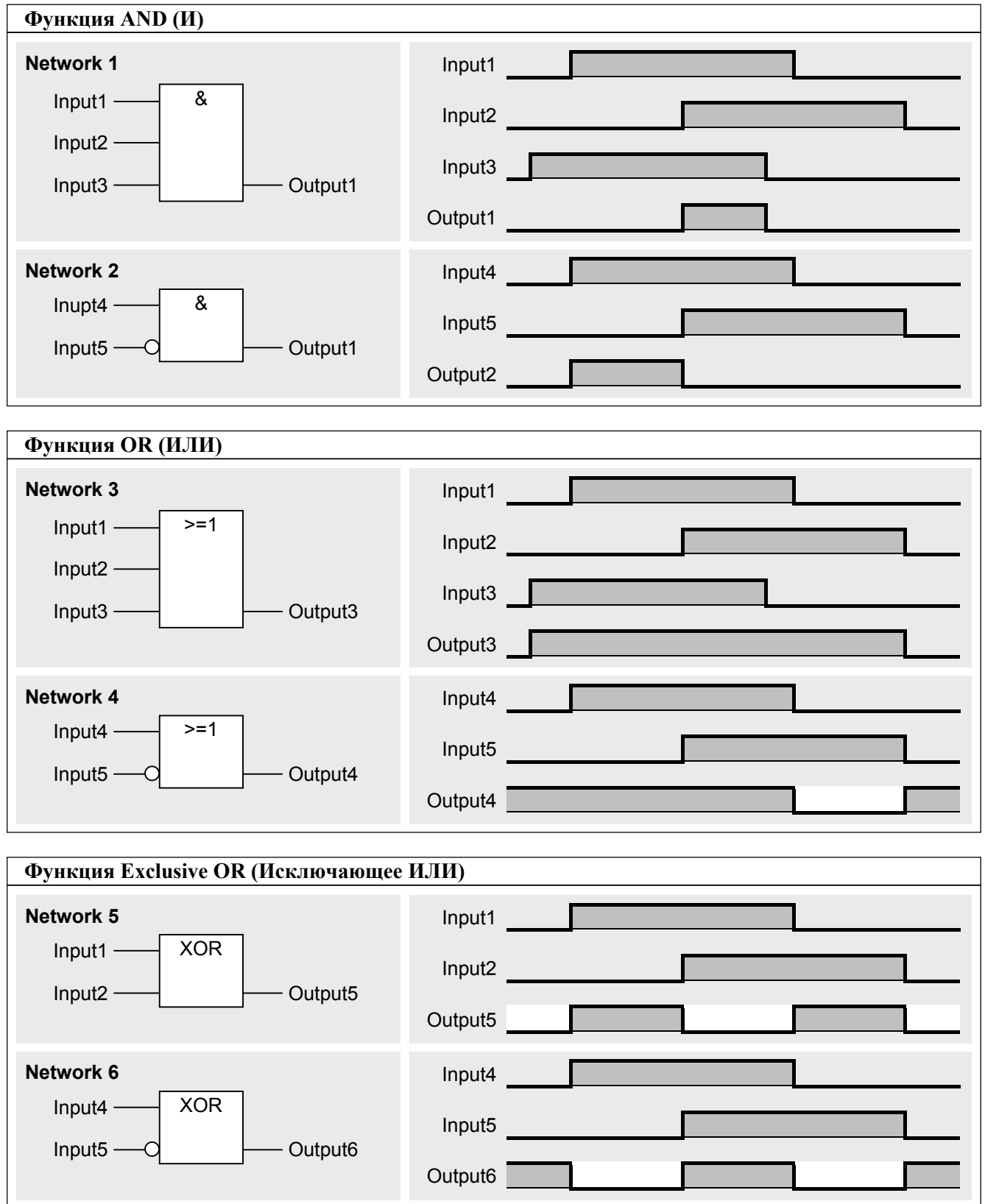


Рисунок 4.6 Примеры бинарных функций

Функция OR (ИЛИ)

Функция OR комбинирует два бинарных сигнальных состояния и возвращает RLO «1», когда одно из этих состояний (один из результатов сканирования) равно «1». Если функция OR имеет несколько входов, то для того, чтобы результат логической операции (RLO) был «1», требуется только один вход, результат сканирования которого равен «1». Функция OR возвращает RLO «0», когда результаты сканирования всех входов равны «0».

Рисунок 4.6 иллюстрирует пример функции OR. На сети 3 (Network 3) функция OR оснащена тремя входами; каждый из этих входов может быть соединен с любым бинарным операндом. Все операнды сканируются с ожиданием сигнального состояния «1», поэтому сигнальное состояние операндов напрямую объединяются по OR. Если один или более сканированных операндов имеют сигнальное состояние «1», то следующий оператор установит операнд *Output3 (Выход3)* в «1». Если все сканированные операнды имеют сигнальное состояние «0», то условие OR не удовлетворяется, и операнд *Output1 (Выход1)* сбрасывается в «0».

Сеть 4 (Network 4) демонстрирует функцию OR с инвертированным входом. Инвертирование (отрицание) представлено кружком. Результат сканирования инвертированного операнда равен «1», если данный операнд равен «0», то есть в примере условие OR выполняется, если операнд *Input4 (Вход4)* имеет сигнальное состояние «1» или операнд *Input5 (Вход5)* имеет сигнальное состояние «0».

Функция Exclusive OR (Исключающее ИЛИ)

Функция Exclusive OR комбинирует друг с другом два бинарных состояния и возвращает RLO «1», когда два состояния (результаты сканирования) не являются одинаковыми, и RLO «0», если два состояния (результаты сканирования) идентичны.

На рисунке 4.6 приведен пример функции исключающего OR. На сети 5 (Network 5) два входа, оба сканируемые с ожиданием сигнального состояния «1», ведут в функцию Exclusive OR. Если только один из сканированных операндов равен «1», то условие исключающего OR выполняется, и операнд *Output5 (Выход5)* устанавливается в «1». Если оба операнда «1» или «0», то операнд *Output5 (Выход5)* равен «0».

Схема 6 изображает функцию исключающего OR с инвертированным входом. Инвертирование (отрицание) представляется кружком. Результат сканирования инвертированного операнда равен «1», когда операнд сброшен в «0», то есть условие исключающего OR в примере выполняется, если оба входных операнда имеют одинаковые сигнальные состояния.

Вы также можете запрограммировать функцию Exclusive OR с количеством входов более двух. В таком случае условие исключающего OR удовлетворяется (в случае непосредственного сканирования), если нечетное число сканированных операндов имеют результат сканирования «1».

4.2.2 Комбинации операций бинарной логики

Вы можете свободно комбинировать бинарные функции одна с другой. Например, можно объединить несколько функций AND по функции OR или две функции OR по функции исключающего OR. Количество функций в логической операции (схеме или сети) теоретически неограничено.

Использование «Т-ветви» в логической операции (контактном плане) предоставляет дополнительные возможности, позволяющие программировать для одной схемы более одного выхода (обратитесь к параграфу 5.2 «Блочные элементы FBD»).

Вы можете соединить выход одной бинарной функции с входом другой с целью выполнения сложной бинарной логической операции. На рисунке 4.7 представлено несколько примеров.

Сеть 9 (Network 9): вы осуществляете наблюдение концевых выключателей (limit switches) на окончаниях осей X и Y. Эти концевые выключатели не могут быть активированы попарно; иначе возникнет сообщение об ошибке концевых выключателей.

Схема 10 (Network 10): вы можете соединить произвольные входы функции с бинарными функциями, например, можно поместить функцию исключающего OR перед вторым входом функции AND.

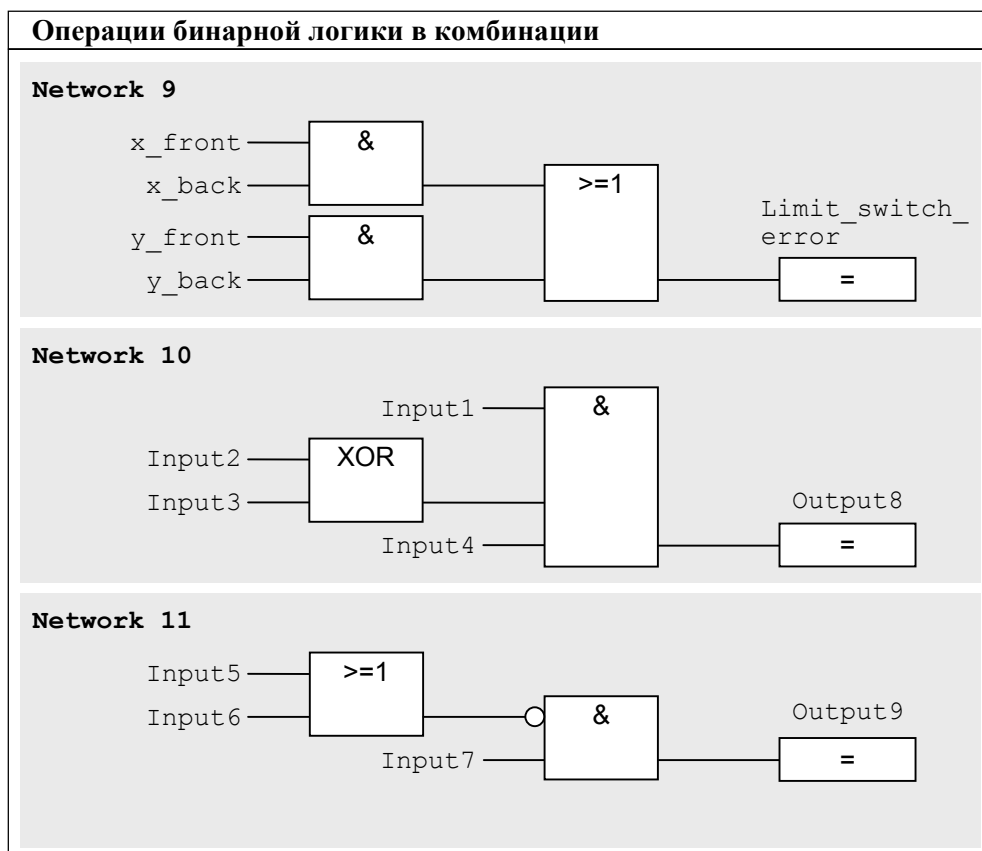


Рисунок 4.7 Примеры операций бинарной логики в комбинации

Сеть 11 (Network 11): используя символ отрицания (инвертирования), вы инвертируете RLO, даже между бинарными функциями, например, вы можете инвертировать RLO функции OR и использовать его в качестве входа в функцию AND.

4.2.3 Инвертирование результата логической операции

Кружок на входе или выходе символа функции инвертирует результат логической операции. Вы можете использовать инвертирование (отрицание)

- для сканирования бинарного операнда, что эквивалентно сканированию с ожиданием сигнального состояния «0» (см. выше),
- между двумя бинарными функциями (что эквивалентно инвертированию результата логической операции) или
- на выходе бинарной функции (к примеру, если вы хотите установить или сбросить бинарный операнд, когда условие не выполнено, то есть когда RLO = «0»).

Инвертирование не может следовать сразу за Т-ветвью.

Рисунок 4.8 показывает функцию NAND (функцию AND с инвертированным выходом) и функцию NOR (функцию OR с инвертированным выходом). RLO функции NAND равен «0» только тогда, когда все входы имеют сигнальное состояние «1». Функция NOR возвращает RLO «1» только тогда, когда ни один из входов не обладает сигнальным состоянием «1».

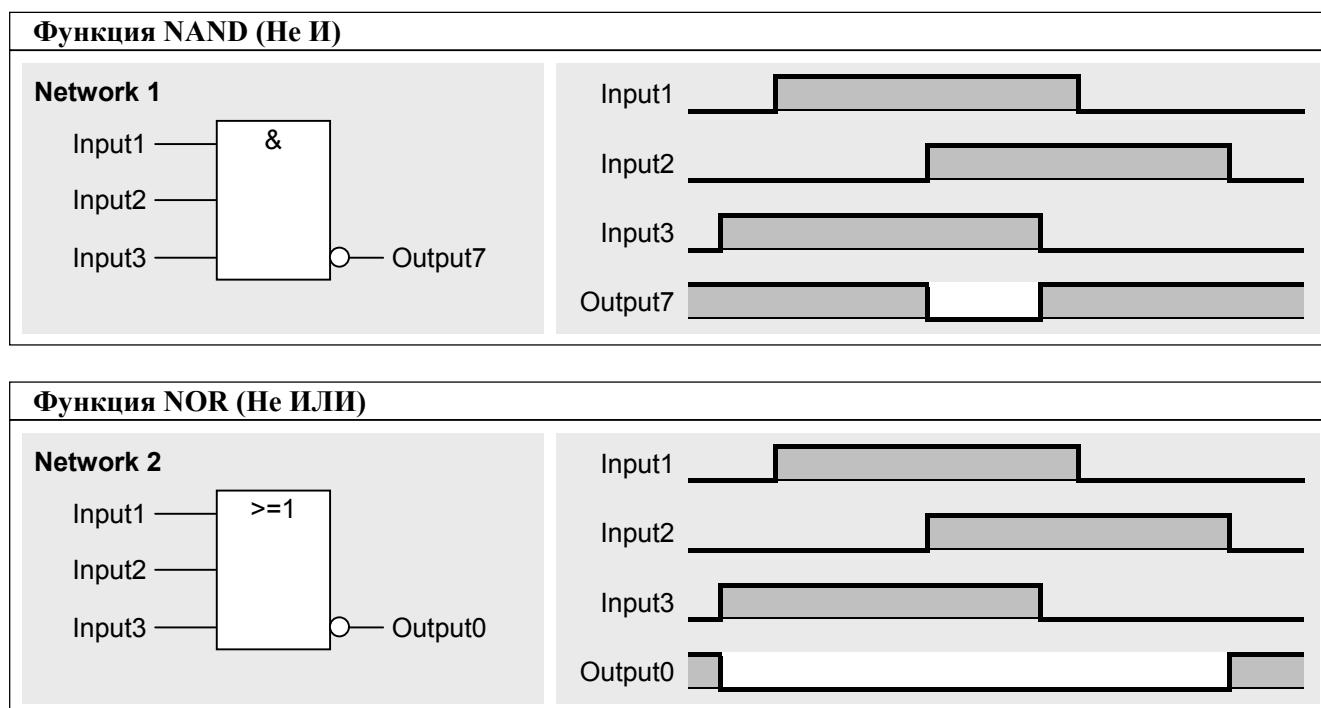


Рисунок 4.8 Функции NAND и NOR

4.3 Действия в зависимости от типа датчика

При сканировании датчика (sensor) в пользовательской программе вы должны уделять внимание тому, каким контактом он является – NC или NO. Когда датчик активирован, в зависимости от его типа на соответствующем входе имеется различное сигнальное состояние: «1» в случае NO-контакта и «0», если это NC-контакт. CPU не располагает средствами определения, занят вход NO-контактом или NC-контактом. Центральный процессор может лишь распознать сигнальное состояние «1» или сигнальное состояние «0».

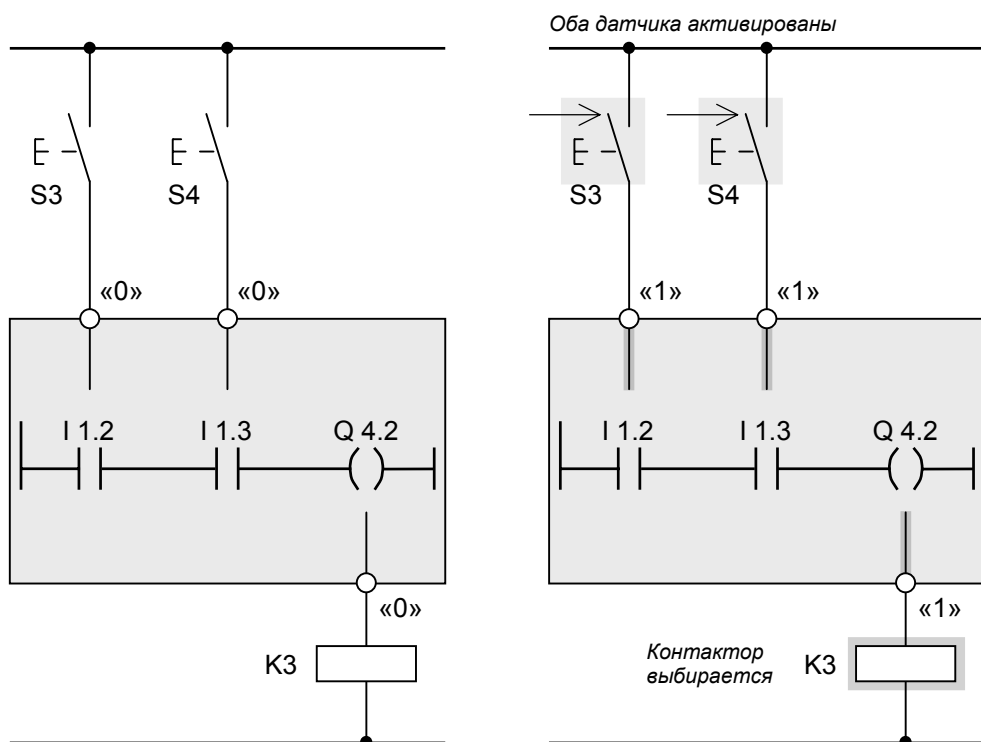
LAD-программирование

Если вы создаете структуру программы таким способом, что вы хотите получить результат сканирования «1» при активировании датчика для того, чтобы в последствии этот результат комбинировать, то для различных видов датчиков вы должны сканировать вход по-разному. Для этой цели вам доступны NO-контакты и NC-контакты. NO-контакт возвращает «1», если сканированный вход также равен «1». NC-контакт возвращает «1», если сканированный вход равен «0». Таким образом, вы можете также непосредственно (напрямую) сканировать входы, которые должны вызывать действия, находясь в состоянии «0» (входы с нулевой активностью), и в дальнейшем перепропустить (re-gate) результат сканирования.

Пример на рисунке 4.9 демонстрирует приемы программирования в зависимости от типа датчика. В первом случае два NO-контакта подключены к программируемому контроллеру, во втором случае – один NO-контакт и один NC-контакт. В обоих вариантах контактор, соединенный с выходом, должен быть выбран, если оба датчика активированы. Если NO-контакт активирован, сигнальное состояние на входе «1», и оно сканируется NO-контактом, поэтому ток может протекать, когда датчик активирован. Если оба NO-контакта активированы, ток течет по цепи к катушке, и контактор выбирается.

Если активирован NC-контакт, сигнальное состояние на входе «0». Чтобы в этом случае ток протекал при активированном датчике, результат должен сканироваться NC-контактом. Следовательно, во втором случае NO-контакт и NC-контакт должны быть подключены последовательно с целью выбора контактора при активировании обоих датчиков.

Случай 1: оба датчика являются NO-контактами



Случай 2: один NO- и один NC-контакт

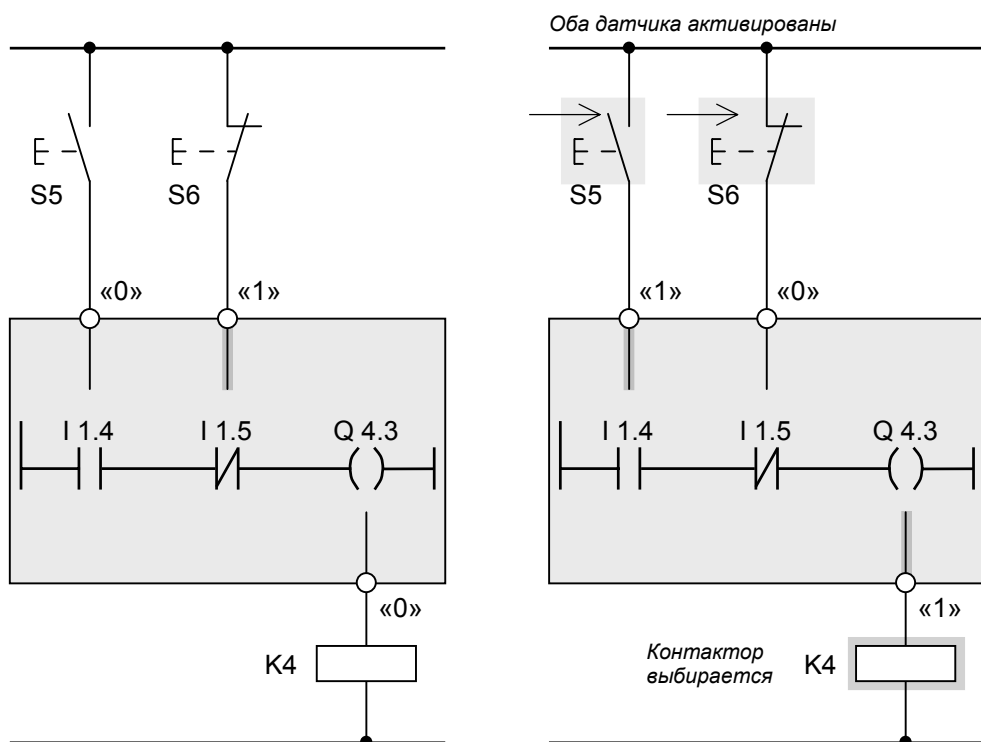


Рисунок 4.9 Действия в зависимости от типа датчика (LAD)

FBD-программирование

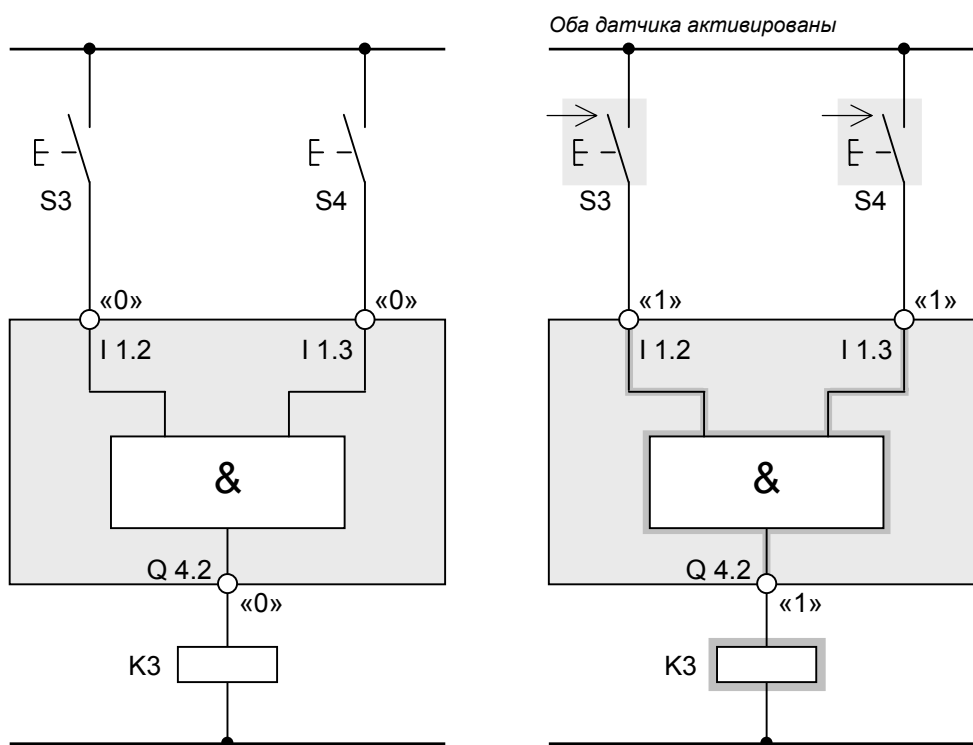
Если вы создаете структуру программы таким способом, что вы хотите получить результат сканирования «1» при активировании датчика для того, чтобы в последствии этот результат комбинировать, то для различных видов датчиков вы должны сканировать вход по-разному. NO-контакт при активировании генерирует сигнальное состояние «1» и сканируется непосредственно, когда активация датчика должна вызвать генерирование результата сканирования «1». NC-контакт возвращает при активации сигнальное состояние «0»; если вы хотите получить результат сканирования «1» при активации NC-контакта, то он должен быть инвертирован и затем сканирован.

Поэтому вы также можете сканировать входы, которые должны вызывать действия, даже если они имеют сигнальное состояние «0» (входы с нулевой активностью), и в дальнейшем комбинировать результат сканирования.

Пример на рисунке 4.10 показывает программирование, зависимое от типа датчика. В первом случае два NO-контакта подключены к программируемому контроллеру, во втором случае – один NO-контакт и один NC-контакт. В обоих вариантах контактор, соединенный с выходом, выбирается, если оба датчика активированы. Если активирован NO-контакт, сигнальное состояние на входе «1», и оно сканируется непосредственно (напрямую), поэтому результат сканирования равен «1», когда датчик активирован. Если активированы оба NO-контакта, условие AND выполняется, и контактор выбирается.

Если NC-контакт активирован, сигнальное состояние на входе «0». Для получения результата сканирования «1» вход при сканировании должен быть инвертирован. Во втором случае вам потребуется функция AND с прямым и инвертированным входами, чтобы выбрать контактор при активировании обоих датчиков.

Случай 1: оба датчика являются NO-контактами



Случай 2: один NO- и один NC-контакт

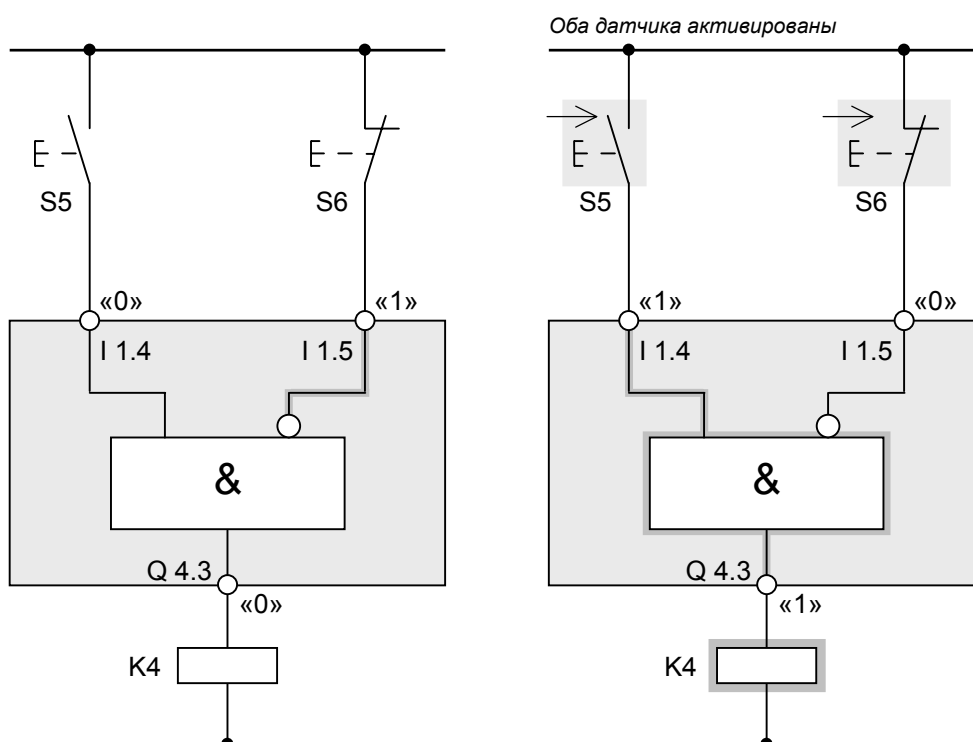


Рисунок 4.10 Действия в зависимости от типа датчика (FBD)

Глава 5

Функции для работы с памятью

Содержание главы 5

5	<u>Функции для работы с памятью</u>	4
5.1	<u>Катушки LAD</u>	4
5.1.1	<u>Одиночная катушка</u>	4
5.1.2	<u>Катушки установки и сброса</u>	5
5.1.3	<u>Блочный элемент памяти (триггер)</u>	7
5.2	<u>Блочные элементы FBD</u>	11
5.2.1	<u>Присваивание</u>	11
5.2.2	<u>Блочные элементы установки и сброса</u>	12
5.2.3	<u>Блочные элементы памяти</u>	14
5.3	<u>Коннекторы</u>	18
5.3.1	<u>Коннекторы в LAD</u>	18
5.3.2	<u>Коннекторы в FBD</u>	19
5.4	<u>Оценка фронта импульса</u>	21
5.4.1	<u>Как работает функция оценки фронта</u>	21
5.4.2	<u>Функция оценки фронта в LAD</u>	23
5.4.3	<u>Функция оценки фронта в FBD</u>	25
5.5	<u>Бинарный делитель частоты</u>	27
5.5.1	<u>Решение в LAD</u>	27
5.5.2	<u>Решение в FBD</u>	27
5.6	<u>Пример системы управления конвейером</u>	30

5 Функции для работы с памятью

5.1 Катушки LAD

В контактном плане (ladder diagram, LAD) функции для работы с памятью (или операции с памятью, memory functions) используются вместе с последовательными и параллельными схемами с целью воздействия на сигнальные состояния бинарных операндов с помощью результата логической операции (result of logic operation, RLO), генерируемого в CPU.

Доступны следующие функции для работы с памятью (функции памяти):

- Одиночная катушка (coil) как присваивание (назначение) RLO;
- Катушки R и S как индивидуально программируемые операции с памятью;
- Блочные элементы (boxes) RS и SR как функции, работающие с памятью;
- Коннекторы (midline outputs) как промежуточные буферы;
- Катушки P и N как элементы оценки (обнаружения) фронта импульса потока электроэнергии (электрического тока);
- Блочные элементы POS и NEG как элементы оценки фронта операндов.

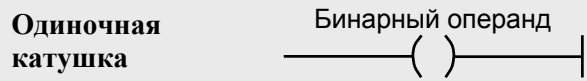
Коннекторы и элементы оценки фронта импульса подробно рассматриваются в последующих главах. Описываемые в данной главе функции для работы с памятью вы можете использовать совместно со всеми бинарными операндами. Ограничения возникают при использовании битов временных локальных данных в качестве меркеров фронта.

Примеры, приведенные в настоящей главе, также представлены на прилагаемой к книге дискете в функциональном блоке 105 программы «Basic functions» («Основные функции»), библиотека «LAD_Book».

Что касается пошагового программирования, то программные элементы для функций для работы с памятью вы найдете в каталоге программных элементов. Команда меню: View → Catalog (Вид → Каталог) [Ctrl + K] или Insert → Program Elements (Вставка → Программные элементы) в «Bit logic» («Битовая логика»).

5.1.1 Одиночная катушка

Одиночная катушка, как терминатор (завершающий элемент) цепи, назначает или направляет (assigns) поток электроэнергии (электрический ток) напрямую к операнду, расположенному при катушке. Функция одиночной катушки зависит от главного реле управления (Master Control Relay, MCR). Если MCR активировано, то бинарному операнду, расположенному над катушкой, назначается сигнальное состояние «0».



Если ток в катушке течет (достигает катушки), то операнд установлен; если тока нет, то операнд не установлен (сброшен) (рисунок 5.1, сеть 1, Network 1). С помощью NOT-контакта перед катушкой вы можете обратить функцию (сеть 2, Network 2).

Кроме того, вы можете направить ток в несколько катушек одновременно, параллельно скомпоновав их с помощью Т-ветви (сеть 3, Network 3). Все операнды, определенные над катушками, реагируют таким же образом. Параллельно можно составить до 16 катушек.

Последующие контакты вы можете скомпоновать в последовательные и параллельные схемы после Т-ветви и перед катушкой (сеть 4, Network 4).

Примеры с применением одиночных катушек приведены в параграфе 4.1 «Последовательные и параллельные схемы (LAD)».

5.1.2 Катушки установки и сброса

Катушки установки и сброса (set coil, reset coil) также могут завершать цепь. Эти катушки становятся активными, только когда через них протекает ток.



Если ток течет в катушке установки, то операнд над катушкой устанавливается в сигнальное состояние «1». Если ток течет в катушке сброса, то операнд над катушкой переустанавливается в сигнальное состояние «0» (сбрасывается). При отсутствии тока в катушке установки или сброса бинарный операнд остается без изменений (рисунок 5.1, сети 5 и 6, Network 5, 6).

Функция катушек установки и сброса зависит от главного реле управления (MCR). Если MCR активировано, то бинарный операнд над катушкой остается неизменным.

Обратите внимание, что операнд, используемый с катушкой установки или сброса, обычно сбрасывается при запуске (полный рестарт - complete restart). В особых случаях сигнальное состояние сохраняется. Это зависит от режима запуска (например, «теплый» рестарт – warm restart), используемого операнда (к примеру, статические локальные данные) и установок в CPU (например, характеристики по сохранению).

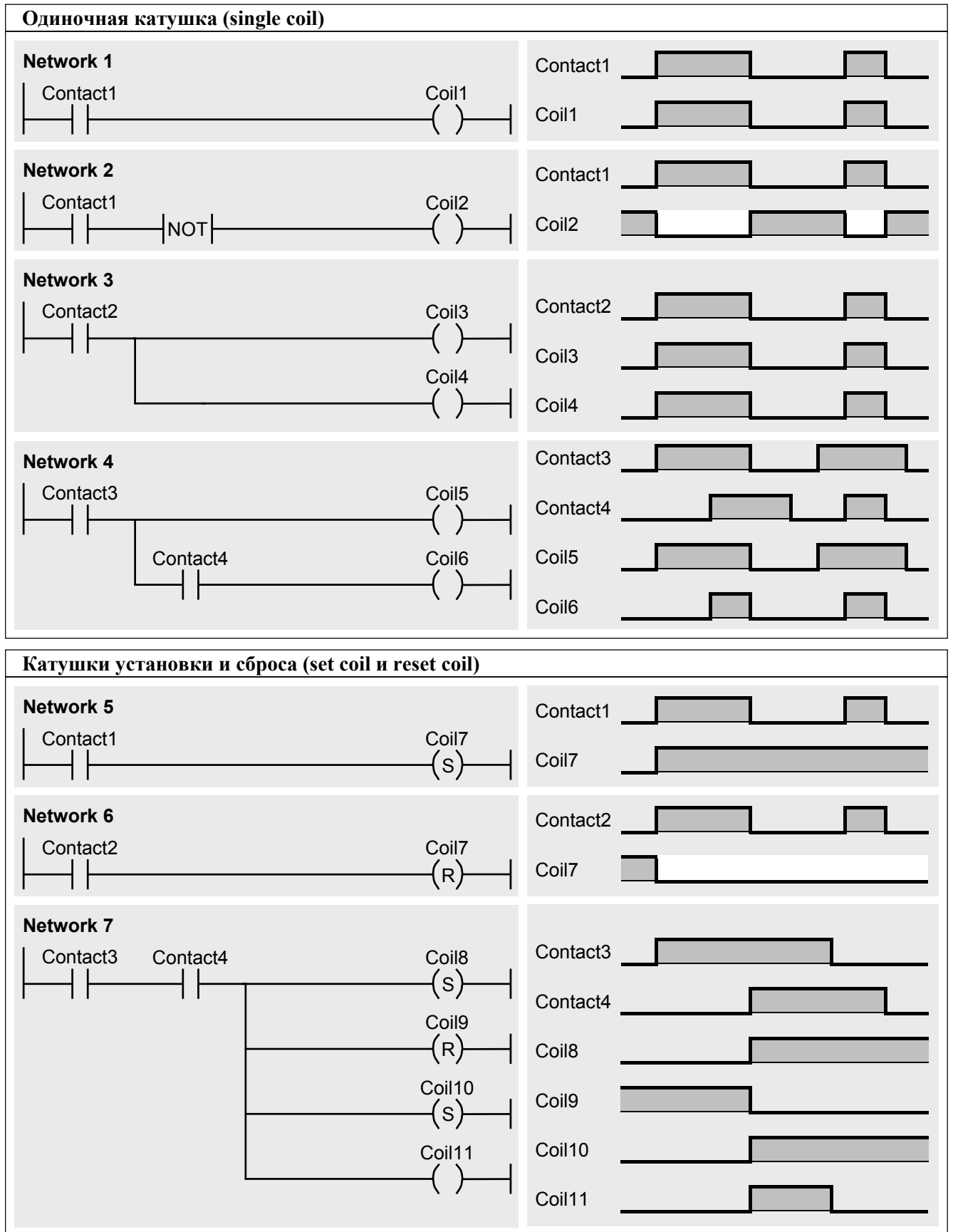


Рисунок 5.1 Одиночная катушка, катушки установки и сброса

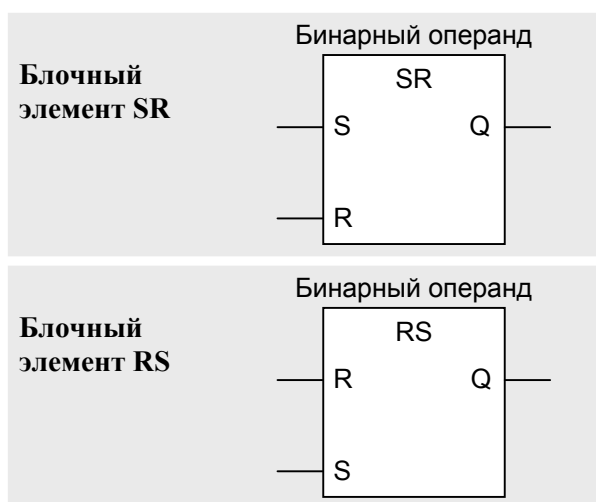
Вы можете сконфигурировать несколько катушек установки и сброса в любой комбинации и вместе с одиночными катушками в одной цепи (сеть 7, Network 7). Для достижения ясности в программировании вам рекомендуется группировать катушки установки и сброса, совместно воздействующие на операнд, попарно и использовать их в каждом случае только один раз. Кроме того, не допускайте дополнительного управления операндами одиночной катушкой.

Как в случае с одиночной катушкой, допустимо поместить контакты после других элементов ветви и перед катушками установки и сброса.

5.1.3 Блочный элемент памяти (триггер)

Функции катушек установки и сброса объединяются в блочном элементе функции для работы с памятью (memory box). Общий бинарный операнд располагается над блочным элементом. Вход S (set input) блочного элемента в данном случае соответствует катушке установки, вход R (reset input) – катушке сброса. Сигнальное состояние двоичного операнда, назначаемое функции для работы с памятью, находится на выходе Q функции памяти.

Имеются два варианта функции памяти: в виде блочных элементов SR (приоритет сброса) и RS (приоритет установки). Кроме различий в обозначении элементы отличаются также компоновкой входов S и R.



Функция памяти установлена (или, точнее, установлен бинарный операнд над блочным элементом памяти), если вход установки имеет сигнальное состояние «1», а вход сброса имеет сигнальное состояние «0». Функция памяти сброшена, если «1» находится на входе сброса, а «0» - на входе установки. Сигнальное состояние «0» на обоих входах не влияет на функцию для работы с памятью. Если оба входа одновременно равны «1», то две имеющиеся функции памяти реагируют по-разному: функция памяти SR сбрасывается, а функция памяти RS – устанавливается.

Поведение блочного элемента памяти зависит от главного реле управления. Если MCR активно, то бинарный операнд блочного элемента памяти сохраняется.

Заметьте, что операнд, используемый с функцией памяти, при запуске (полный рестарт) обычно сбрасывается. В особых случаях сигнальное состояние блочного элемента памяти сохраняется. Это зависит от режима старта (например, «теплый» рестарт), используемого операнда (к примеру, статические локальные данные) и установок в CPU (например, характеристики по сохранению).

Функция для работы с памятью SR (триггер с приоритетом сброса)

В блочном элементе памяти SR приоритет имеет вход сброса (reset input). Приоритет сброса означает, что функция памяти является или остается сброшенной, если ток течет «одновременно» во входе установки и входе сброса. Вход сброса имеет приоритет перед входом установки (рисунок 5.2а, сеть 8, Network 8).

Так как операторы выполняются последовательно, CPU изначально устанавливает операнд памяти (memory operand), потому что вход установки обрабатывается первым, но потом снова сбрасывает его, когда обрабатывает вход сброса. Пока оставшаяся программа обрабатывается, операнд памяти остается обнуленным.

Если операнд памяти является выходом, эта недолговременная установка имеет место только в выходной таблице образа процесса, а (внешний) выход в соответствующем модуле выхода остается неизменным. CPU не передает выходную таблицу образа процесса в модули выходов до конца программного цикла.

Функция памяти с приоритетом сброса является «нормальной» формой функции для работы с памятью, так как состояние сброса (сигнальное состояние «0») обычно безопаснее или менее рискованное состояние.

Функция для работы с памятью RS (триггер с приоритетом установки)

В блочном элементе памяти RS приоритет имеет вход установки (set input). Приоритет установки означает, что функция памяти является или остается установленной, если ток течет «одновременно» во входе установки и входе сброса. Поэтому вход установки имеет приоритет перед входом сброса (рисунок 5.2а, сеть 9, Network 9).

В соответствии с последовательным исполнением инструкций CPU сбрасывает операнд памяти с входом сброса, обрабатываемым первым, но затем вновь устанавливает его, когда обрабатывает вход установки. Операнд памяти остается установленным, пока обрабатывается оставшаяся программа.

Если операнд памяти является выходом, эта недолговременная установка имеет место только в выходной таблице образа процесса, а (внешний) выход в соответствующем модуле выхода остается неизменным. CPU не передает выходную таблицу образа процесса в модули выходов до конца программного цикла.

Приоритет установки при использовании функции для работы с памятью является исключением. Он используется, к примеру, в реализации буфера сообщения о сбое, если на входе установки постоянное (still) текущее сообщение об ошибке должно

продолжить установку функции памяти, несмотря на подтверждение на входе сброса.

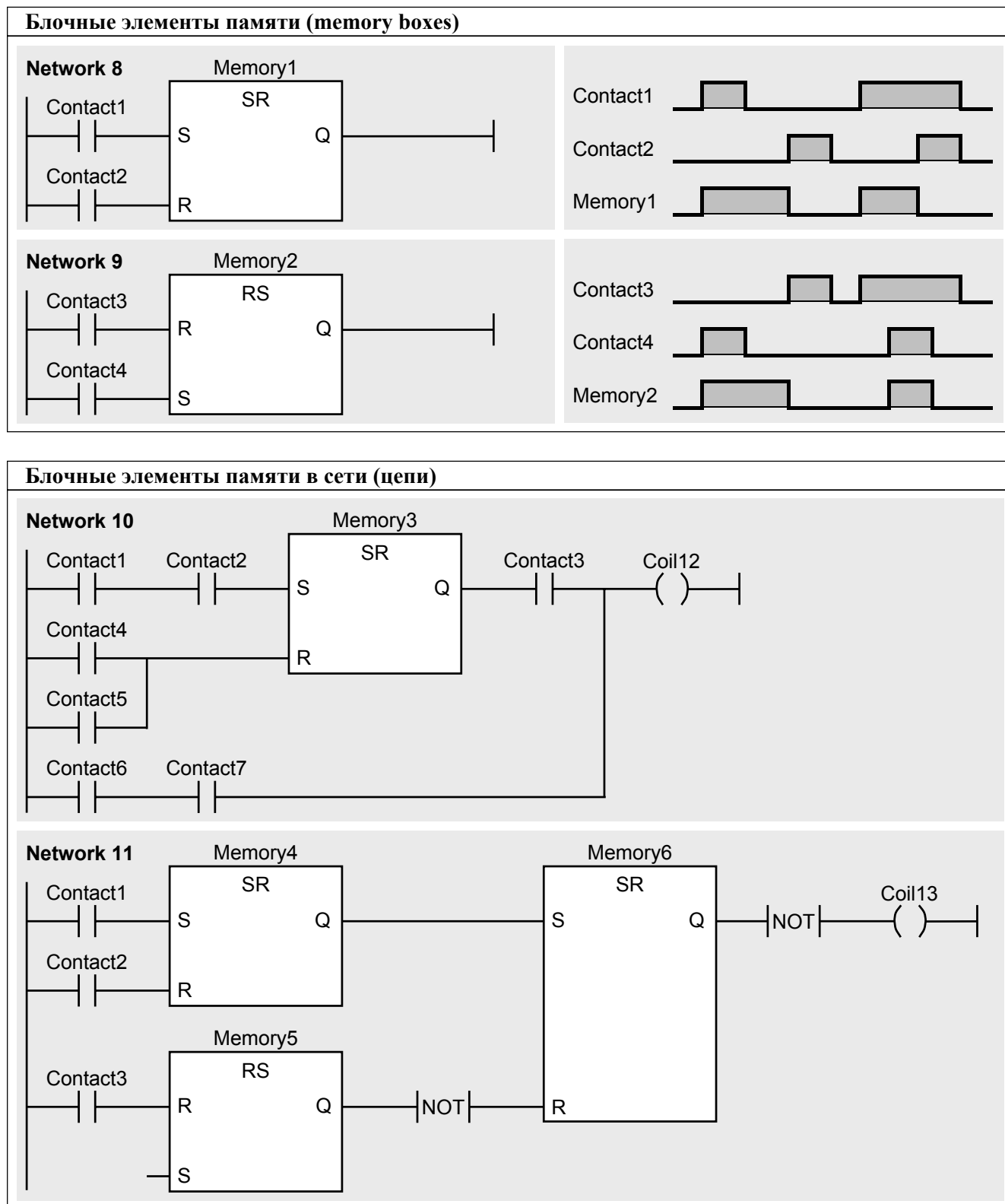


Рисунок 5.2а Функции памяти (LAD)

Функция памяти внутри цепи

Вы также можете поместить блочный элемент памяти внутри цепи. Контакты могут быть соединены последовательно и параллельно и при входах, и при выходах (рисунок 5.2а, сеть 10, Network 10). Второй вход блочного элемента памяти допускается оставлять неподключенным. В рамках одной цепи можно соединять несколько блочных элементов памяти между собой. Вы можете компоновать блочные элементы памяти последовательно и параллельно (сеть 11, Network 11). Расположить функцию памяти вы можете после Т-ветви или в звене, которое начинается от левой несущей (питающей шины).

Функция памяти с блокировкой

В релейной логической диаграмме функция памяти обычно реализуется с помощью блокировки или защелки (latching) управляемого выхода. Этот метод также может применяться при программировании контактного плана. Однако, он имеет недостаток: функция памяти не распознается непосредственно.

Сети 12 и 13 (Network 12, 13) на рисунке 5.2б приводят оба типа функции для работы с памятью, приоритет установки и приоритет сброса, используя блокировку. Принцип блокировки является простым. Бинарный операнд, управляемый катушкой, сканируется, и это сканирование («контакт катушки») соединяется параллельно с условием (состоянием) установки. Если *Contact1* (Контакт1) замыкается, *Coil14* (Катушка14) возбуждается и замыкает контакт, параллельный контакту *Contact1* (Контакт1). Если *Contact1* (Контакт1) снова размыкается, *Coil14* (Катушка14) остается под напряжением. Нагрузка с *Coil14* (Катушка14) снимается, если размыкается *Contact2* (Контакт2). Если сигнальное состояние «1» присутствует на *Contact1* (Контакт1) и *Contact2* (Контакт2), ток к катушке не течет (приоритет сброса). Эта ситуация выглядит по-другому на нижней схеме. Если сигнальное состояние «1» присутствует на *Contact3* (Контакт3) и *Contact4* (Контакт4), ток к катушке течет (приоритет установки).

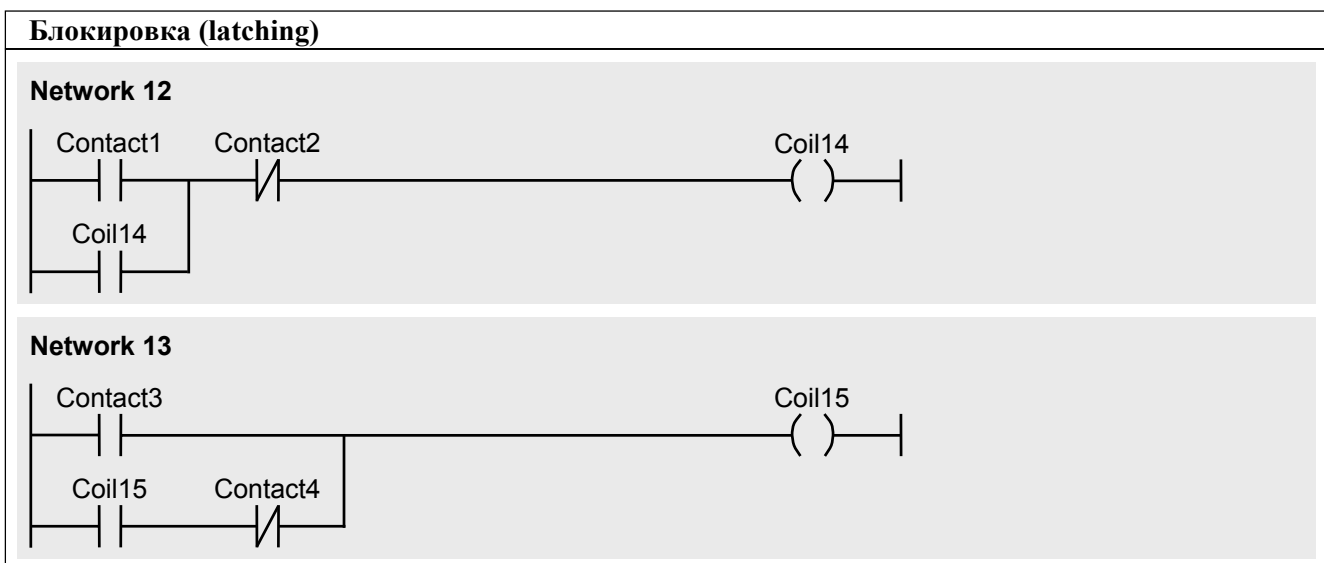


Рисунок 5.2б Функции памяти (LAD)

5.2 Блочные элементы FBD

В FBD блочные элементы памяти используются вместе с операциями бинарной логики с целью влияния на сигнальные состояния бинарных операндов с помощью результата логической операции (RLO), генерируемого CPU.

Доступны следующие функции для работы с памятью:

- Блочный элемент присваивания (assign box) для динамического управления;
- Блочные элементы установки (set) и сброса (reset) как индивидуально программируемые функции памяти;
- Блочные элементы RS и SR как вполне законченные функции памяти;
- Блочный элемент коннектора (midline output box) как промежуточный буфер;
- Блочные элементы P и N как элементы оценки (обнаружения) фронта результата логической операции;
- Блочные элементы POS и NEG как определители фронта операндов.

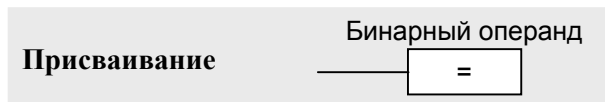
Блочные элементы коннектора и определители фронта подробно обсуждаются в следующих главах.

Вы можете использовать функции памяти, описываемые в данной главе, совместно со всеми бинарными операндами. Ограничения накладываются при использовании битов временных локальных данных в качестве меркеров фронта (edge memory bits).

Примеры, показанные в этой главе, также представлены на дискете, прилагаемой к книге, в функциональном блоке FB 105 раздела «Basic Functions» («Основные функции»), библиотека «FBD_Book». Программные элементы функций, работающих с памятью, для пошагового программирования вы найдете в каталоге программных элементов, пункт меню View → Catalog (Вид → Каталог) [Ctrl + K] или Insert → Program Elements (Вставка → Программные элементы) в разделе «Bit logic» («Битовая логика»).

5.2.1 Присваивание

Блочный элемент присваивания (assign) как терминатор (завершающий элемент) цепи присваивает (назначает) результат логической операции непосредственно операнду, смежному с блочным элементом. Если RLO равен «1» на входе блочного элемента присваивания, то бинарный операнд устанавливается; если RLO равен «0», то операнд сбрасывается. Действие блочного элемента присваивания зависит от главного реле управления (MCR). Если MCR активировано, то бинарному операнду над блочным элементом назначается сигнальное состояние «0».



Работу блочного элемента присваивания разъясняют несколько примеров на рисунке 5.3.

Сеть 1 (Network 1): операнд Output1 (Выход1) непосредственно принимает сигнальное состояние операнда Input1 (Вход1).

Сеть 2 (Network 2): вы можете применить инвертирование, чтобы обратить функцию блочного элемента присваивания.

Сеть 3 (Network 3): вы можете направить RLO нескольким блочным элементам одновременно путем вставки Т-ветви и компоновки блочных элементов с соответствующими операндами один под другим («множественный вывод или мультивывод»). Все операнды над блочными элементами реагируют таким же образом.

Сеть 4 (Network 4): вы можете вставить бинарные функции между Т-ветвью и завершающим блочным элементом, расширяя таким образом логическую операцию дополнительными функциональными блочными элементами.

Дополнительные примеры по блочному элементу присваивания вы найдете в параграфе 4.2 «Операции бинарной логики (FBD)».

5.2.2 Блочные элементы установки и сброса

Блочные элементы установки и сброса (set box и reset box) также могут завершать логическую операцию (функциональный план). Эти блочные элементы активируются только тогда, когда результат логической операции, направляющийся в блочный элемент, равен «1».



Если RLO, направляющийся в блочный элемент установки, равен «1», то операнд над элементом устанавливается в сигнальное состояние «1». Если RLO, идущий в блочный элемент сброса, равен «1», то операнд над элементом переходит в сигнальное состояние «0». Если RLO, передаваемый в блочный элемент установки или сброса, равен «0», бинарный операнд остается неизменным. Функция блочных элементов установки и сброса зависит от главного реле управления (MCR). Если MCR активировано, то бинарный операнд над блочным элементом воздействию не подвергается.

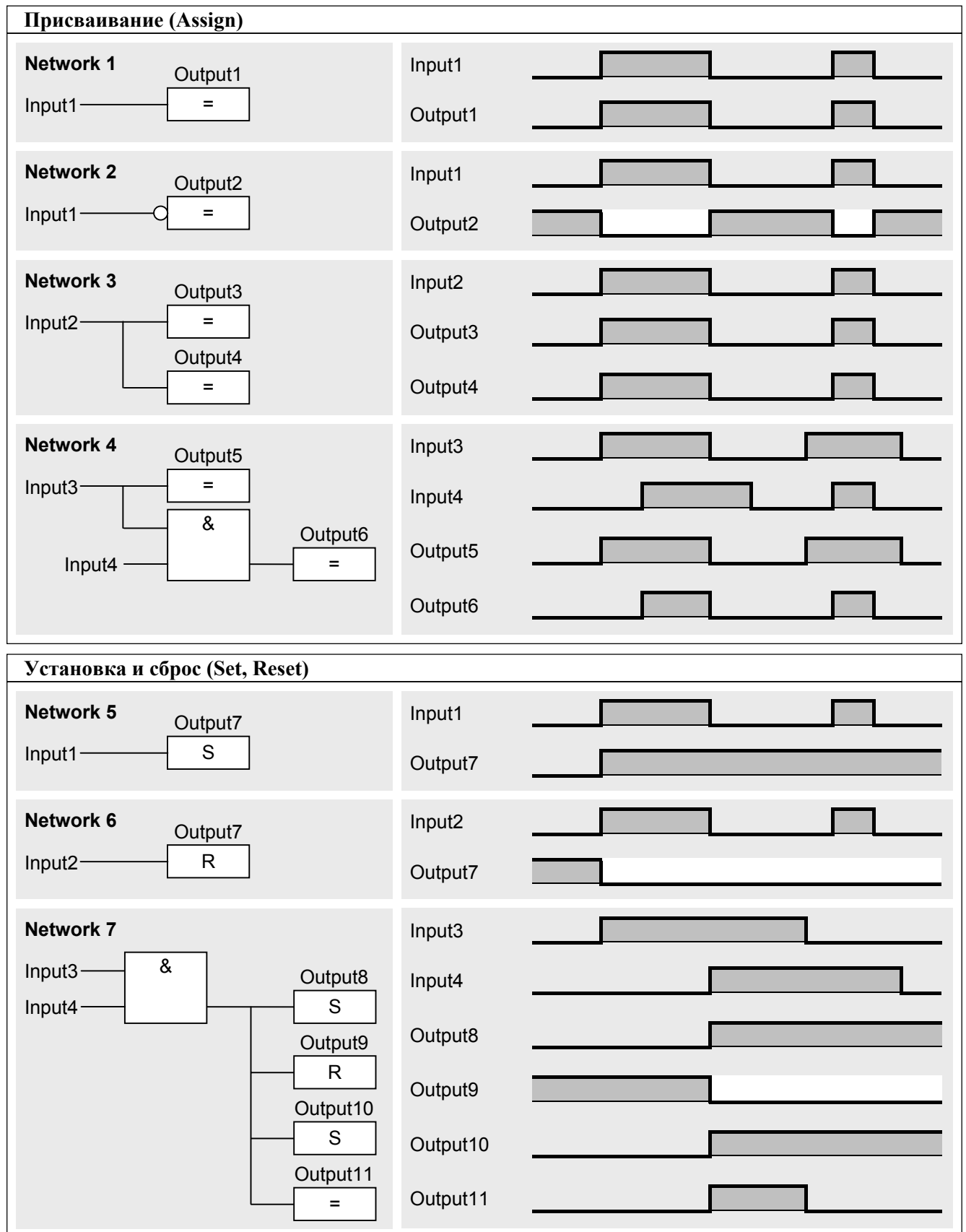


Рисунок 5.3 Присваивание, установка и сброс (FBD)

На рисунке 5.3 показаны несколько примеров работы блочных элементов установки и сброса.

Сеть 5 (Network 5): операнд *Output7 (Выход7)* устанавливается, когда операнд *Input1 (Вход1)* равен «1». Когда *Input1 (Вход1)* возвращается в сигнальное состояние «0», *Output7 (Выход7)* остается установленным.

Сеть 6 (Network 6): операнд *Output7 (Выход7)* сбрасывается, когда операнд *Input2 (Вход2)* равен «1». Когда *Input2 (Вход2)* возвращается в сигнальное состояние «0», *Output7 (Выход7)* остается сброшенным.

Сеть 7 (Network 7): вы можете сгруппировать несколько блочных элементов установки и сброса в любой комбинации и вместе с блочными элементами присваивания в одном плане после T-ветви. Как и в случае с элементом присваивания, вы также можете запрограммировать бинарные функции после T-ветви и перед элементами установки и сброса.

Для достижения прозрачности в программировании рекомендуется попарно группировать блочные элементы установки и сброса, воздействующие на операнд, и использовать их в каждом случае только один раз. Также следует избегать управления этими операндами со стороны блочного элемента присваивания.

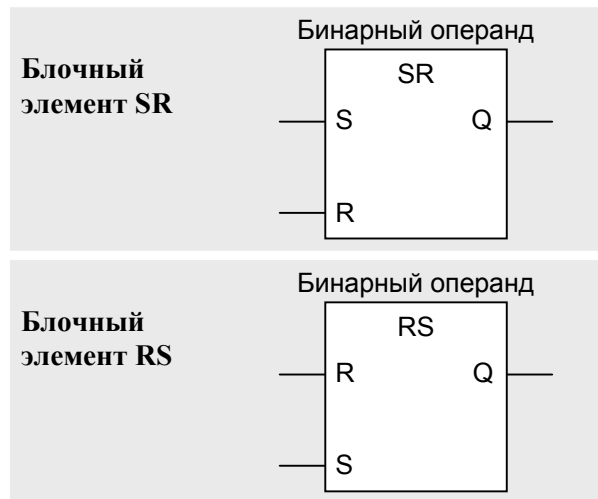
Обратите внимание на то, что операнд, используемый с блочными элементами установки и сброса, при запуске обычно сбрасывается (полный рестарт – complete restart). В особых случаях сигнальное состояние сохраняется. Это зависит от режима запуска (например, «теплый» рестарт – warm restart), задействованного операнда (например, статические локальные данные) и установок в CPU (таких как характеристики, отвечающие за сохранение).

5.2.3 Блочные элементы памяти

Функции блочных элементов установки и сброса объединены в блочном элементе функции памяти. Общий бинарный операнд расположен над блочным элементом. Вход S блочного элемента соответствует в данном случае блочному элементу установки (set box), а вход R – блочному элементу сброса (reset box). Сигнальное состояние двоичного операнда, назначаемое функции памяти, подается на выход Q функции памяти.

Предоставляются два варианта функции, работающих с памятью: в виде блочного элемента SR (приоритет сброса) и в виде блочного элемента RS (приоритет установки). Кроме обозначения элементы также отличаются друг от друга компоновкой входов S и R.

Функция памяти установлена (или точнее бинарный операнд над блочным элементом памяти установлен), когда вход установки (set input) равен «1» и вход сброса (reset input) равен «0». Функция памяти обнулена, когда вход сброса установлен в «1», а на входе установки – «0».



Сигнальное состояние «0» на обоих входах не оказывает влияния на функцию памяти. Если оба входа имеют сигнальное состояние «1» одновременно, рассматриваемые две функции памяти реагируют по-разному: функция памяти SR обнуляется, а функция памяти RS устанавливается. Функционирование блочного элемента памяти зависит от главного реле управления (MCR). Если MCR активировано, то бинарный операнд блочного элемента памяти остается без изменений.

Заметьте, что операнд, используемый с функцией памяти, при запуске обычно сброшен (полный рестарт). В особых случаях сигнальное состояние блочного элемента памяти сохраняется. Это зависит от режима запуска (например, теплый рестарт), используемого операнда (к примеру, статические локальные данные) и установок в CPU (таких как характеристик, отвечающих за сохранение).

Функция для работы с памятью SR

В блочном элементе памяти SR приоритет имеет вход сброса (reset input). Приоритет сброса означает, что функция памяти является или остается сброшенной, если RLO равен «1» на входах установки и сброса «одновременно». Таким образом, вход сброса имеет приоритет перед входом установки (рисунок 5.4, сеть 8, Network 8).

Так как операторы выполняются последовательно, CPU сначала устанавливает операнд памяти, потому что вход установки обрабатывается первым, но затем сбрасывает его снова, когда обрабатывает вход сброса. Пока остальная программа обрабатывается, операнд памяти остается обнуленным.

Если операнд памяти является выходом, эта кратковременная установка имеет место только в выходной таблице образа процесса, а (внешний) выход в соответствующем модуле выхода остается неизменным. CPU не передает выходную таблицу образа процесса в модули выходов до конца программного цикла.

Функция памяти с приоритетом сброса является «нормальной» формой функции для работы с памятью, так как состояние сброса (сигнальное состояние «0») является обычно безопаснее или менее рискованным.

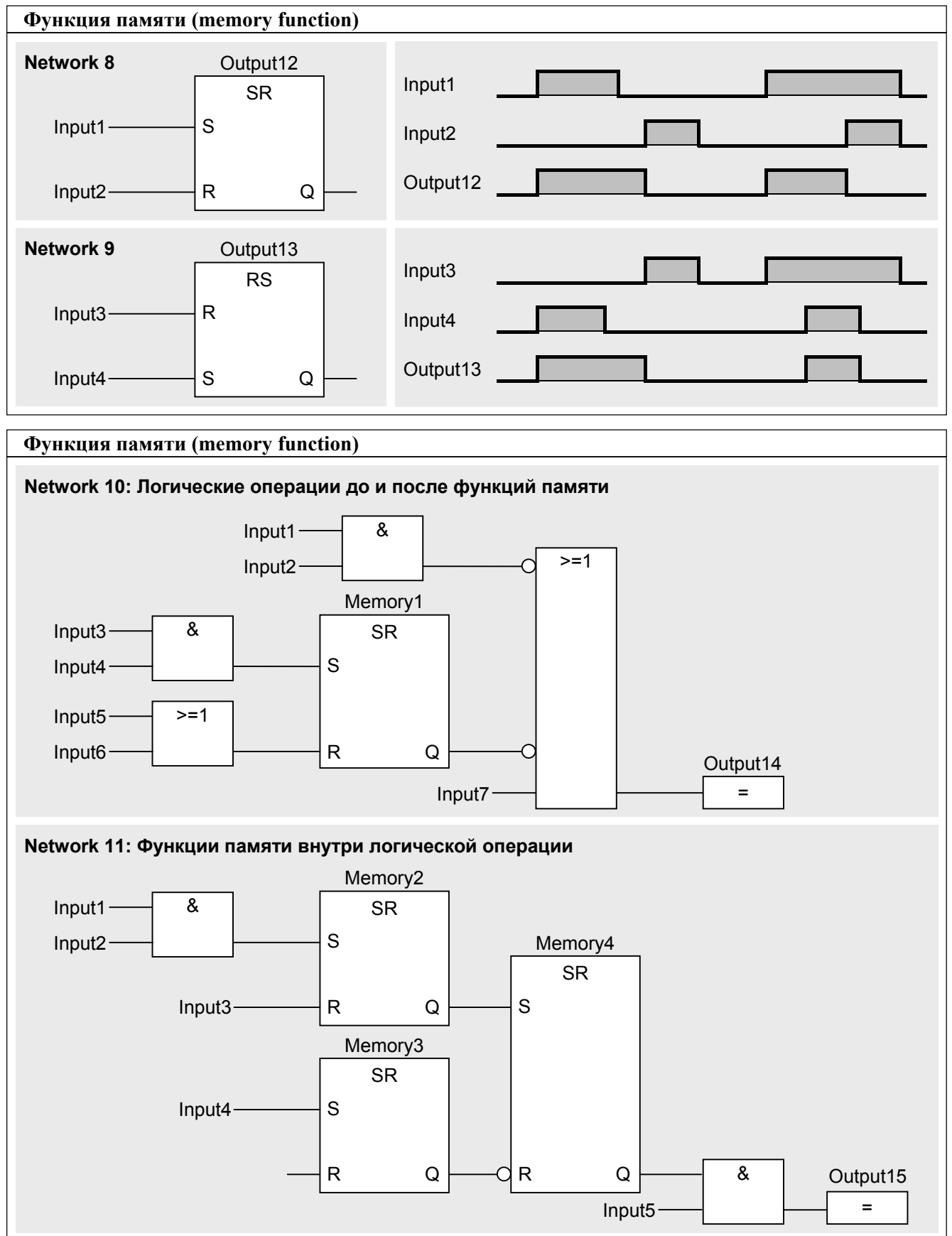


Рисунок 5.4 Функции памяти (FBD)

Функция для работы с памятью RS

В блочном элементе памяти RS приоритет имеет вход установки (set input). Приоритет установки означает, что функция памяти является или остается установленной, если RLO равен «1» на входах установки и сброса «одновременно». Поэтому вход установки имеет приоритет перед входом сброса (рисунок 5.4, сеть 9, Network 9).

Так как операторы выполняются последовательно, CPU сначала сбрасывает операнд памяти, потому что вход сброса обрабатываемым первым, затем вновь устанавливает его, когда обрабатывает вход установки. Операнд памяти остается установленным, пока обрабатывается остальная программа.

Если операнд памяти является выходом, эта недолговременная установка имеет место только в выходной таблице образа процесса, а (внешний) выход в соответствующем модуле выхода остается неизменным. CPU не передает выходную таблицу образа процесса в модули выходов до конца программного цикла.

Приоритет установки скорее является исключением, чем правилом. Он используется, к примеру, в реализации буфера сообщения о сбое, если на входе установки постоянное (still) текущее сообщение об ошибке должно продолжить установку функции памяти, несмотря на подтверждение на входе сброса.

Функция памяти внутри логической операции

Вы также можете поместить блочный элемент памяти внутри логической операции. Можно запрограммировать бинарные функции и на входах, и на выходе (рисунок 5.4, сеть 10, 11, Network 10, 11). Можно оставить второй вход блочного элемента памяти несоединенным. Внутри логической операции вы можете соединить несколько блочных элементов памяти. Блочные элементы памяти могут быть размещены один за другим или друг под другом после T-ветви.

5.3 Коннекторы

Коннекторы (midline outputs) являются промежуточными буферами в контактном или функциональном планах. RLO, действительный для коннектора, хранится в операнде над этим коннектором. Этот операнд может быть снова опрошен в другой точке программы, что позволяет вам также постобрабатывать (post-process) RLO, действительный для коннектора, в другом месте программы.

Для промежуточного хранения двоичных результатов применимы следующие бинарные операнды:

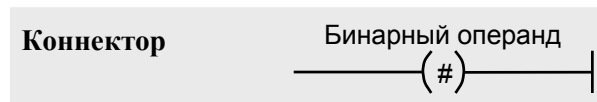
- Вы можете использовать биты временных локальных данных, если промежуточный результат вам требуется только внутри блока. Все кодовые блоки имеют временные локальные данные.
- Биты статических локальных данных доступны только в рамках функционального блока; они хранят сигнальное состояние до их повторного использования, даже за границами блока.
- Меркеры доступны глобально в определяемом моделью CPU фиксированном количестве; для достижения прозрачности программирования старайтесь избегать многократного использования меркеров (одних и тех же меркеров для различных задач).
- Биты данных в глобальных блоках данных также являются доступными на протяжении всей программы, но требуют, чтобы перед их использованием соответствующие блоки данных были открыты (даже если связаны через массовую адресацию).

Функционирование коннектора зависит от главного реле управления (MCR). Если MCR активировано, то бинарному операнду, соотнесенному с коннектором, присваивается бинарное состояние «0». Тогда и RLO вслед за коннектором будет равен «0» (то есть «поток электроэнергии» отсутствует).

Замечание: вы можете заменить сверхоперативную память (scratchpad memory) на временные локальные данные, доступные в каждом блоке.

5.3.1 Коннекторы в LAD

Коннектор является одиночной катушкой в цепи. RLO, действительный для этой точки (электрический ток, который течет в цепи, в данной точке), хранится в двоичном операнде над коннектором. Сам коннектор не оказывает влияния на электрический ток.



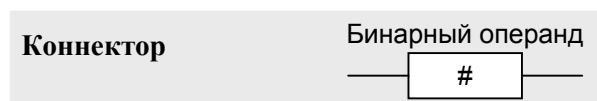
Вы можете сканировать бинарный операнд над коннектором в другой точке программы с помощью NO- и NC-контактов. В одной цепи может быть запрограммировано несколько коннекторов.

Коннектор можно поместить в ветви, которая начинается на левой несущей (шине питания). Он может также следовать за T-ветвью, но не может завершать цепь; для этой цели применяется одиночная катушка.

Рисунок 5.5 иллюстрирует пример того, как промежуточный результат сохраняется в коннекторе. RLO из цепи, формируемый контактами *Contact1* (Контакт1), *Contact2* (Контакт2), *Contact4* (Контакт4) и *Contact5* (Контакт5), сохраняется в коннекторе *Midl_out1* (Коннектор1). Если условие логической операции выполняется (ток течет в коннекторе), и если *Contact3* (Контакт3) замкнут, то *Coil16* (Катушка16) возбуждается. Хранимый RLO используется в следующей сети (network) двумя способами. С одной стороны, производится проверка выполнения условия логической операции и битовой логической комбинации, осуществленной с *Contact6* (Контакт6), а с другой стороны, производится проверка невыполнения условия логической операции и битовой логической комбинации, осуществленной с *Contact7* (Контакт7).

5.3.2 Коннекторы в FBD

Коннектор – это блочный элемент присваивания внутри логической операции. RLO, действительный для этой точки, хранится в бинарном операнде над его блочным элементом.



Вы можете опросить бинарный операнд над коннектором в другой точке программы. В одном функциональном плане может быть запрограммировано несколько коннекторов. Блочный элемент коннектора не должен завершать логическую операцию; для этой цели служит блочный элемент присваивания.

Сети 12 и 13 (Network 12, 13) на рисунке 5.5 показывают, как промежуточный результат сохраняется в коннекторе. RLO из цепи, формируемый входами *Input1* (Вход1), *Input2* (Вход2), *Input3* (Вход3) и *Input4* (Вход4), сохраняется в коннекторе *Midl_out1* (Коннектор1). Если условие логической операции выполняется, и если сигнальное состояние входа *Input5* (Вход5) равно «1», то *Coil16* (Катушка6) активируется. Хранимый RLO используется в следующей сети (network) двумя способами. С одной стороны, производится проверка выполнения условия логической операции и битовой логической комбинации, осуществленной с *Input6* (Вход6), а с другой сто-

роны, производится проверка невыполнения условия логической операции и битовой логической комбинации, осуществленной с *Input7 (Вход7)*.

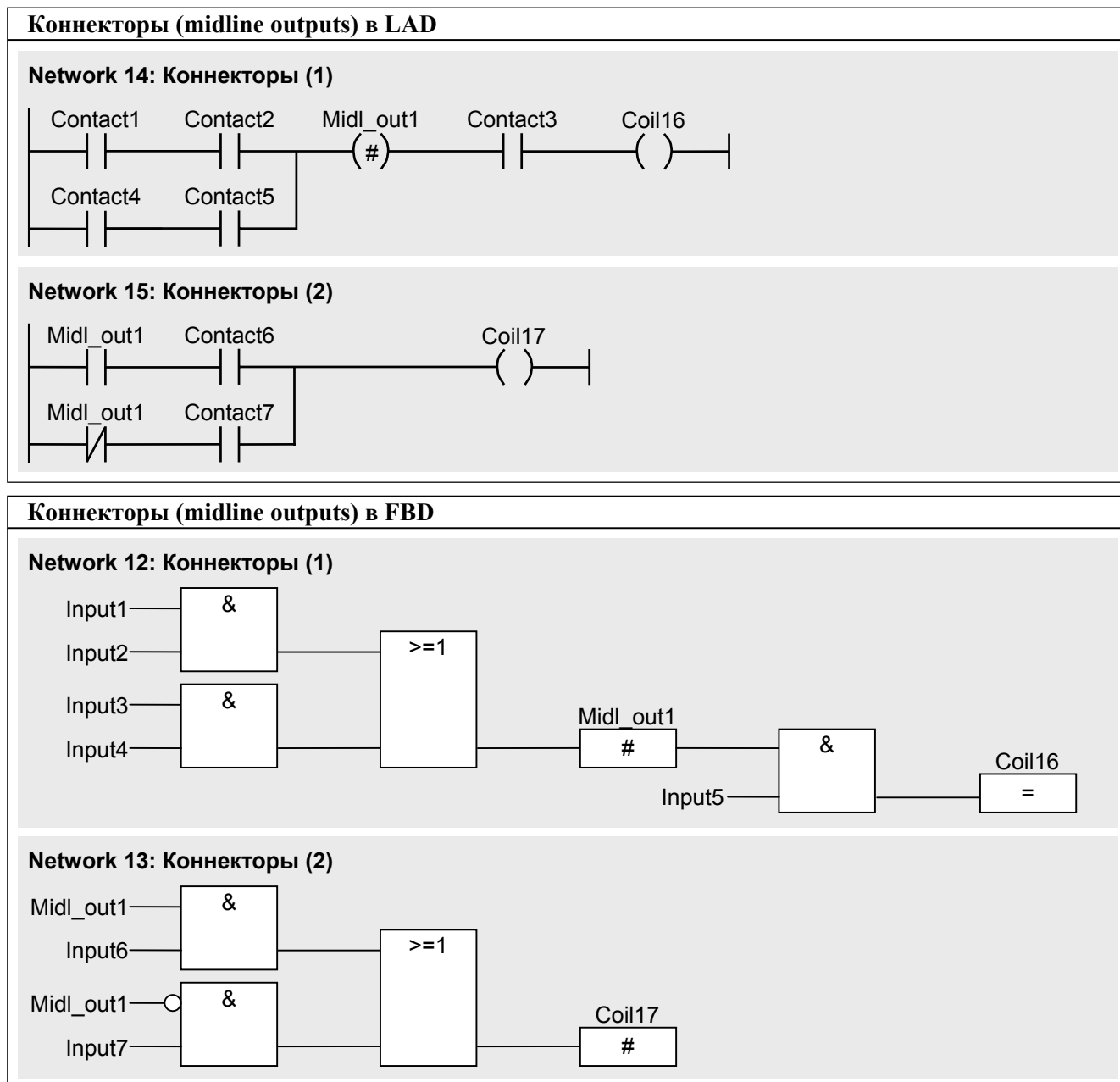


Рисунок 5.5 Коннекторы

5.4 Оценка фронта импульса

5.4.1 Как работает функция оценки фронта

С помощью функции оценки фронта (edge evaluation) вы можете обнаружить изменение в сигнальном состоянии, фронт сигнала. Фронт является положительным (возрастающим), когда сигнал меняется с «0» на «1». В противоположном случае фронт отрицателен (падает).

В цепи LAD и плане FBD эквивалентом оценки фронта является импульсный контактный элемент (pulse contact element). Если этот импульсный контактный элемент испускает импульс при включенном реле, это соответствует возрастающему фронту. Импульс из импульсного контактного элемента при выключении соответствует спадающему фронту.

Обнаружение фронта сигнала (изменение в сигнальном состоянии) реализуется в программе. CPU сравнивает текущий RLO (например, результат проверки входа) с сохраненным RLO. Если сигнальные состояния различны, то фронт сигнала присутствует.

Хранимый RLO расположен в «меркере фронта» («edge memory bit») (он не обязательно должен быть меркером). Это должен быть операнд, чье сигнальное состояние должно быть доступно, когда снова встречается оценка фронта (в следующем программном цикле), и который не используется в каком-либо другом месте программы. В качестве операндов могут применяться меркеры, биты данных в глобальных блоках данных и биты статических локальных данных в функциональных блоках.

Меркер фронта сохранит «старый» RLO, с которым CPU совершил последнюю обработку оценки фронта. Если фронт сигнала в настоящий момент присутствует, то есть если текущий RLO отличается от сигнального состояния меркера фронта, то CPU корректирует сигнальное состояние меркера фронта путем присваивания ему «значения» «нового» RLO. При следующей обработке оценки фронта (обычно в следующем программном цикле) сигнальное состояние меркера фронта то же, что и у текущего RLO (если оно между тем не изменилось), и CPU больше не производит обнаружение фронта.

Обнаруженный фронт индицируется результатом логической операции (RLO) после оценки фронта. Если CPU выявляет фронт сигнала, то он устанавливает RLO в «1» после оценки фронта (то есть ток течет). Если фронта сигнала не обнаружено, то RLO равен «0».

Следовательно, сигнальное состояние «1» после оценки фронта означает «фронт обнаружен» («edge detected»). Присутствие сигнального состояния «1» кратковременно, обычно в течение только одного программного цикла. Так как CPU не обнаруживает (не производит обнаружения) фронт в следующем цикле (если «входной RLO» оценки фронта не меняется), он устанавливает RLO обратно в «0» после оценки фронта.

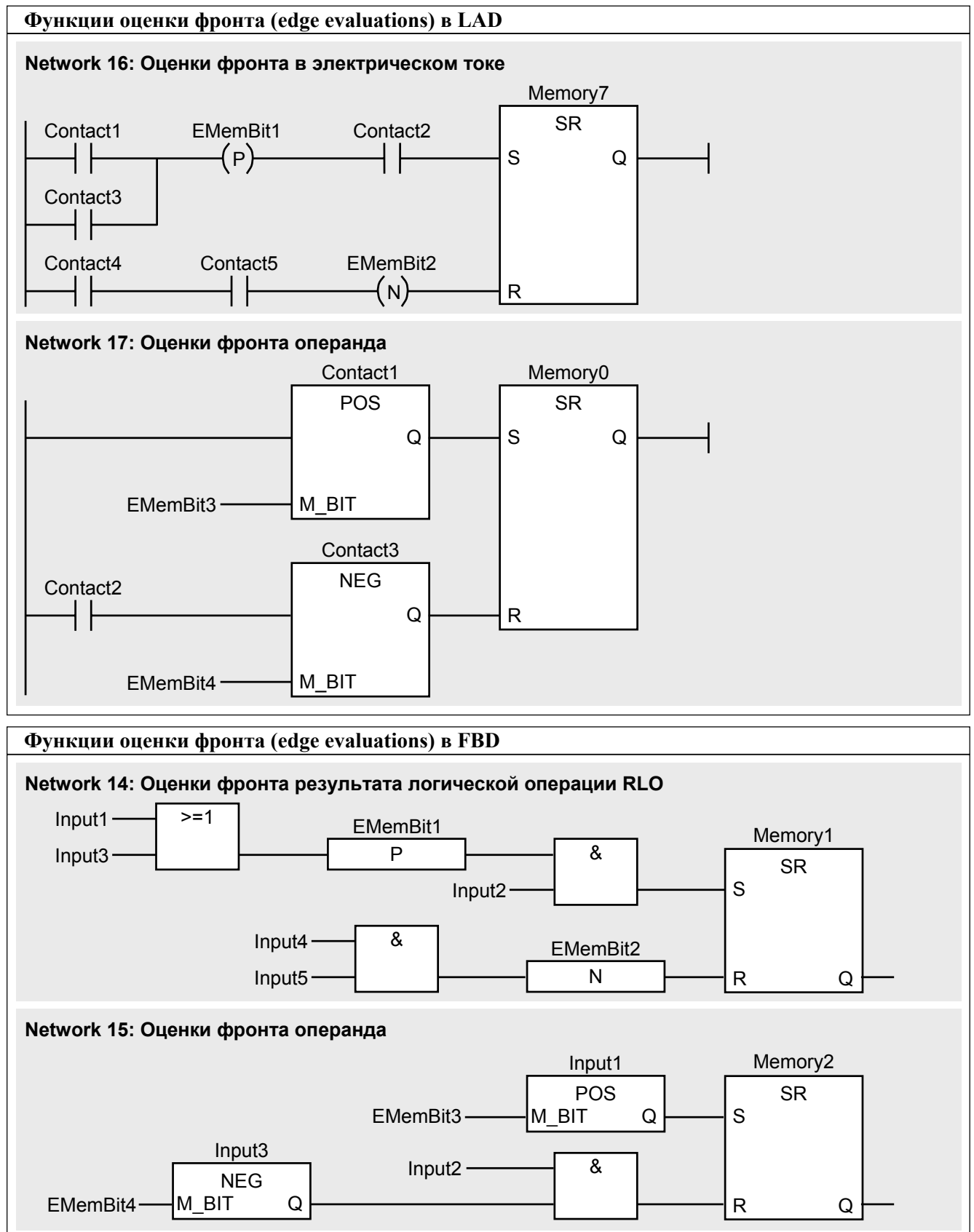
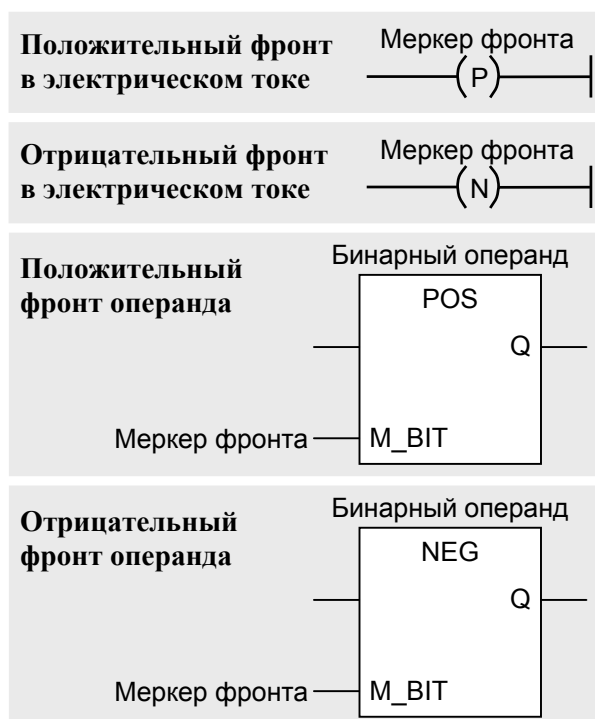


Рисунок 5.6 Функции оценки фронта

Обратите внимание на рабочие параметры оценки фронта, когда CPU запускается. Если фронт не должен быть обнаружен, RLO перед оценкой фронта должен совпадать с сигнальным состоянием меркера фронта при включенном CPU. При определенных обстоятельствах меркер фронта должен быть сброшен в процедуре запуска (в зависимости от требуемого режима работы и используемых операндов).

5.4.2 Функция оценки фронта в LAD

Язык программирования LAD для оценки фронта предоставляет четыре различных элемента:



Вы можете обрабатывать RLO сразу после оценки фронта, то есть, к примеру, сохранять его с помощью катушки установки, комбинировать его с использованием расположенных далее контактов или сохранять его в бинарном операнде (так называемом «импульсном меркере», «pulse memory bit»).

Импульсный меркер применяется, когда RLO из оценки фронта должен использоваться в другом месте программы; это, так сказать, промежуточный буфер для выявленного фронта (импульсный контактный элемент в диаграмме схемы). Операндами, применимыми в качестве импульсного меркера, являются меркеры, биты данных в глобальных блоках данных и биты временных и статических локальных данных.

Оценка фронта тока

Оценка фронта тока отображается катушкой, содержащей Р (для положительного, возрастающего фронта) или N (для отрицательного, спадающего фронта). Над катушкой располагается меркер фронта, бинарный операнд, в котором хранится «старый» RLO из предыдущей оценки фронта. Оценка фронта, как эта, обнаруживает изменение в потоке электроэнергии (электрическом токе) с «ток течет» на «ток не течет» и наоборот.

Пример на рисунке 5.6 в сети 16 (Network 16) показывает оценку положительного и отрицательного фронта. Если параллельная цепь, состоящая из *Contact1* (Контакт1) и *Contact3* (Контакт3), выполняется, оценка фронта испускает короткий импульс с помощью *EMemBit1* (МеркерФронта1). Если *Contact2* (Контакт2) замкнут в этот момент, то *Memory7* (Память7) устанавливается. *Memory7* (Память7) сбрасывается вновь импульсом от *EMemBit2* (МеркерФронта2), если последовательная цепь, состоящая из *Contact4* (Контакт4) и *Contact5* (Контакт5), прерывает электрический ток.

Вы можете запрограммировать оценку фронта с использованием катушки после Т-ветви или в цепи, которая начинается на левой несущей (питающей шине). Она не может быть помещена непосредственно у левой несущей (питающей шины).

Оценка фронта операнда

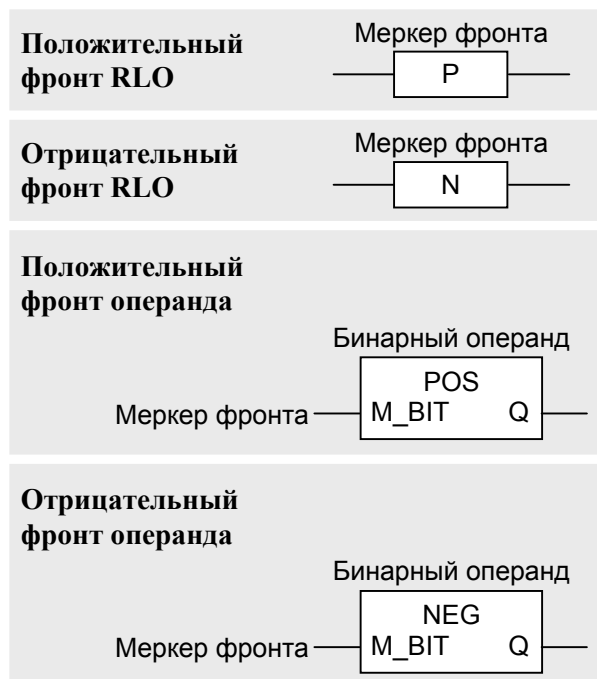
LAD представляет оценку фронта операнда с использованием блочного элемента (box). Над блочным элементом находится операнд, чье изменение сигнального состояния должно быть выявлено. Меркер фронта, который хранит «старое» сигнальное состояние из предыдущего программного цикла, расположен у входа M_BIT.

Элемент оценки фронта имеет немаркированный вход и выход Q и в таком виде «вставляется» в цепь вместо контакта. Если ток течет в немаркированном входе, выход Q при фронте испускает импульс; если в данном входе нет тока, выход Q остается обнуленным. Вы можете вставить эту оценку фронта на место любого контакта, даже в параллельную ветвь, которая не начинается на левой несущей (питающей шине).

Рисунок 5.6 демонстрирует использование оценки фронта операнда на примере сети 17 (Network 17). Оценка фронта в верхней ветви генерирует импульс, если операнд *Contact1* (Контакт1) изменяет свое сигнальное состояние с «0» на «1» (положительный фронт). Этот импульс устанавливает *Memory0* (Память0). Оценка фронта всегда активируется прямым соединением немаркированного входа с левой несущей (питающей шиной). Оценка фронта тока активируется контактом *Contact2* (Контакт2). Если оценка фронта активируется подачей «1» на немаркированный вход, то она испускает импульс при смене сигнального состояния бинарного операнда *Contact3* (Контакт3) с «1» на «0» (отрицательный фронт).

5.4.3 Функция оценки фронта в FBD

Язык программирования FBD для оценки фронта предоставляет четыре различных элемента:



RLO после оценки фронта может быть непосредственно обработан, к примеру, сохранен с помощью катушки установки, использован в комбинации с последующими двоичными функциями или назначен бинарному операнду (так называемому «импульсному меркеру»). Импульсный меркер используется, когда RLO из оценки фронта должен обрабатываться в другом месте программы; это, так сказать, промежуточный буфер для выявленного фронта. Операндами, пригодными для использования в качестве импульсного меркера, могут быть биты данных в глобальных блоках данных и биты временных и статистических локальных данных. После T-ветви оценка фронта не может быть использована.

Оценка фронта RLO

Оценка фронта RLO отображается при помощи блочного элемента, содержащего P (для положительного, возрастающего фронта) или N (для отрицательного, спадающего фронта). Над блочным элементом указывается меркер фронта, бинарный операнд, содержащий «старый» RLO из предыдущего цикла. Оценка фронта как эта обнаруживает изменение RLO внутри схемы с RLO «1» на RLO «0» и наоборот.

Пример, приведенный на рисунке 5.6, сеть 14 (Network 14), показывает оценку положительного и отрицательного фронтов. Когда функция OR, состоящая из *Input1* (*Вход1*) и *Input3* (*Вход3*), выполняется, оценка фронта генерирует короткий импульс с помощью *EMemBit1* (*МеркерФронтa1*). Если *Input2* (*Вход2*) равен «1» в данном случае, то *Memory1* (*Память1*) устанавливается. *Memory1* (*Память1*) вновь сбрасы-

вается импульсом из *EMemBit2* (*МеркерФронта2*), когда функция AND, включающая в себя *Input4* (*Вход4*) и *Input5* (*Вход5*), больше не выполняется.

Оценка фронта операнда

Оценка фронта операнда располагается в начале операции бинарной логики. Над блочным элементом указывается операнд, чье изменение сигнального состояния должно быть оценено. Меркер фронта, который хранит «старое» сигнальное состояние из последнего программного цикла, располагается у входа M_BIT. Выход Q равен «1», когда CPU обнаруживает изменение сигнального состояния операнда.

Сеть 15 (Network 15) на рисунке 5.6 демонстрирует оценку фронта операнда. Верхний элемент оценки фронта POS испускает импульс, когда *Input1* (*Вход1*) переходит из «0» в «1» (положительный фронт). Нижняя оценка фронта NEG генерирует импульс, если бинарный операнд, когда *Input3* (*Вход3*) переходит из «1» в «0» (отрицательный фронт). Когда операнд *Input2* (*Вход2*) равен «1», этот импульс обнуляет *Memory2* (*Память2*).

(Выход) вновь сбрасывается (*Memory (Память)* теперь «1»). Если *Input (Вход)* еще раз будет равен «0», *Memory (Память)* сбрасывается (так как *Output (Выход)* теперь также обнулен). Таким образом, базовое состояние было снова получено после двух входных импульсов и одного выходного.

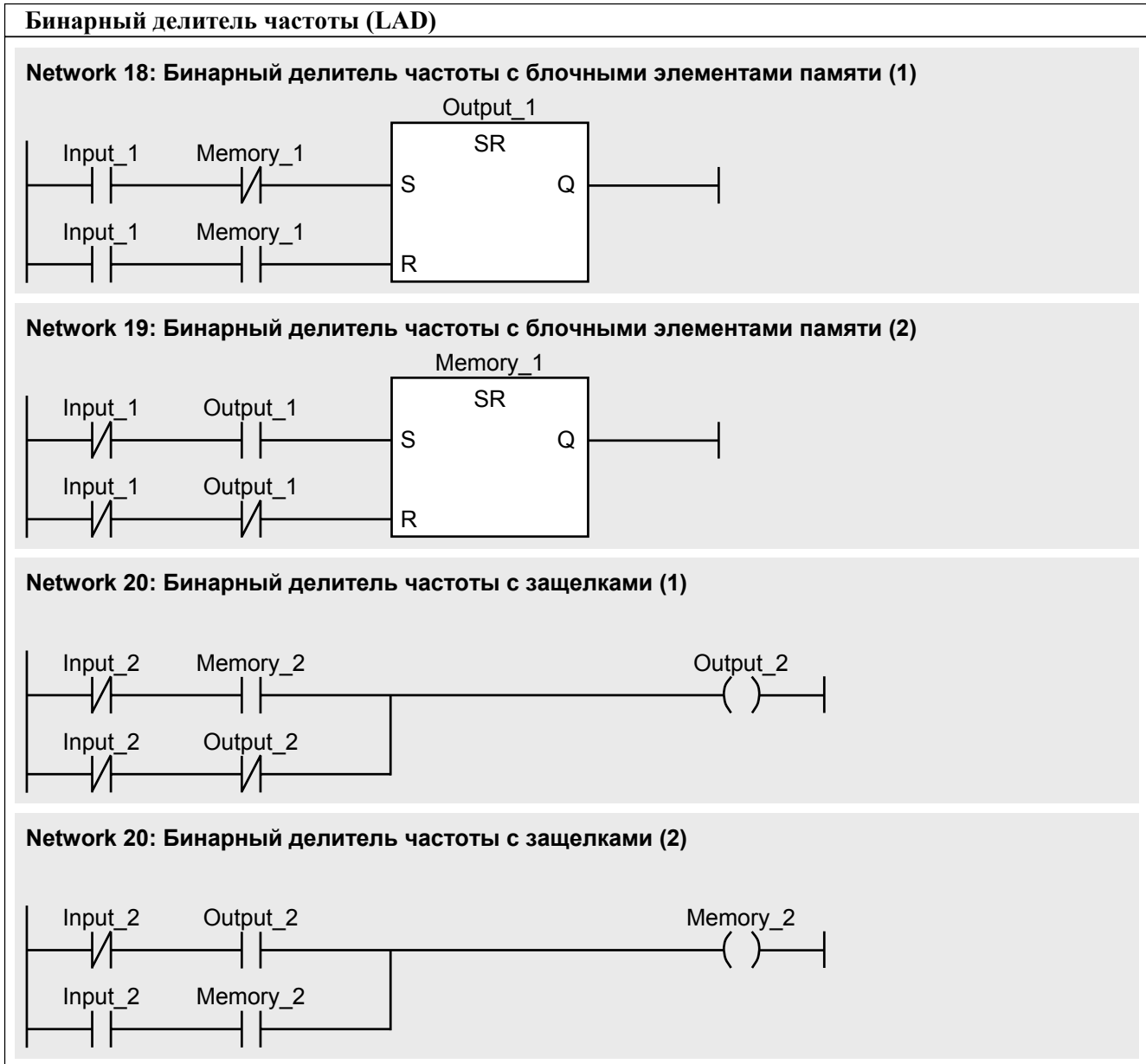


Рисунок 5.8 Примеры бинарного делителя частоты (LAD)

Второй способ решения использует оценку фронта операнда *Input (Вход)* (рисунок 5.9, сети 18 – 20, Network 18 – 20). Если у операнда *Input (Вход)* фронта не обнаружено, то RLO, следующий за оценкой уровня, равен «0», и выполняется инструкция перехода (jump) JCN (подробное описание операторов переходов вы найдете в главе 16 «Функции переходов»). В нашем примере метка перехода называется «bin» и присутствует в сети 20 (Network 20). Именно здесь возобновляется программное сканирование, если фронта не обнаружено. Фактический бинарный делитель частоты

ты имеется в сети 19 (Network 19): если *Output (Выход)* равен «0», то он установлен; если «1», то он обнулен. Эта сеть обрабатывается только тогда, когда обнаруживается фронт операнда *Input (Вход)*. По существу, каждый раз, когда выявляется фронт у *Input (Вход)*, *Output (Выход)* меняет свое сигнальное состояние.

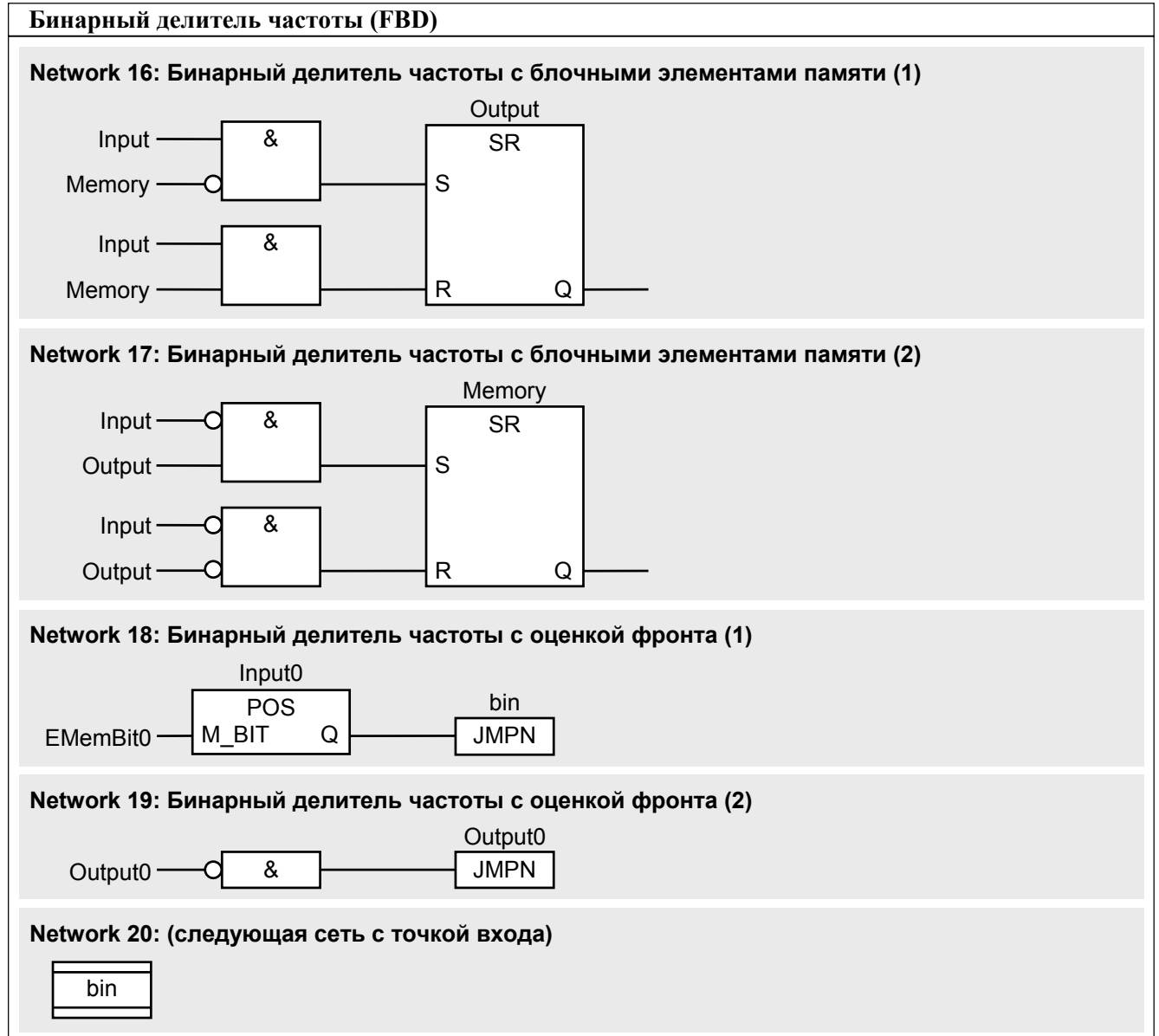


Рисунок 5.9 Примеры бинарных делителей частоты (FBD)

5.6 Пример системы управления конвейером

Следующий пример функционально простейшей системы управления транспортером конвейера иллюстрирует использование операций бинарной логики и функций памяти вместе с входами, выходами и меркерами.

Функциональное описание

- Когда транспортер пуст, контроллер запрашивает детали путем генерирования сигнала «readyload» (ready to load – готов к загрузке);
- Когда выдается сигнал «Start» («Старт»), транспортер запускается и перемещает детали;
- На конце транспортера конвейера датчик «end-of-belt» («конец транспортера»), например, световой барьер (фотоэлемент), обнаруживает детали, и в данной точке мотор транспортера выключается и включает сигнал «ready_remove» (ready to remove – готов к съему детали);
- Когда выдается сигнал «continue» («продолжить»), детали транспортируются дальше, пока датчики «end-belt» (end of belt – конец транспортера) не обнаружат их.

Пример программируется с входами, выходами и меркерами, и может быть запрограммирован в любом блоке в любом месте. В данном случае в качестве блока выбрана функция без значения функции.

Сигналы, символы

Дополняют функциональность системы управления транспортером конвейера несколько сигналов:

- Basic_st
Переводит контроллер в базовое состояние;
- Man_on
Включает транспортер вне зависимости от условий;
- /Stop
Останавливает конвейер на время присутствия «0» (NC-контакт в качестве датчика, «нулевая активность»);
- End_belt
Детали достигли конца транспортера;
- /Mfault
Сигнал ошибки (сбоя) от мотора транспортера (например, выключатель защиты

мотора); разработан как сигнал «нулевой активности», поэтому, к примеру, обрыв провода также генерирует сигнал ошибки.

Мы хотим использовать символическую адресацию, то есть операндам даны имена, которые будем применять при написании программы. Перед вводом программы мы создаем таблицу символов (таблица 5.1), содержащую входы, выходы, меркеры и блоки.

Таблица 5.1 Таблица символов (Symbol Table) для примера «Система управления транспортом конвейера»

Symbol (Символ)	Address (Адрес)	Data Type (Тип данных)	Comment (Комментарий)
Belt_control	FC 11	FC 11	Система управления конвейером
Basic_st	I 0.0	BOOL	Переводит контроллеры в базовое состояние
Man_on	I 0.1	BOOL	Включает мотор транспортера конвейера
/Stop	I 0.2	BOOL	Останавливает мотор транспортера конвейера (нулевая активность)
Start	I 0.3	BOOL	Запускает транспортер конвейера
Continue	I 0.4	BOOL	Уведомление об удалении деталей
Light_barrier1	I 1.0	BOOL	(Световой барьер) сигнал датчика «End of belt» («Конец транспортера») для транспортера 1 (belt 1)
/Mfault	I 2.0	BOOL	Выключатель защиты мотора транспортера 1, нулевая активность
Readyload	Q 4.0	BOOL	Загрузка новых деталей на транспортер (ready to load, готовность к загрузке)
Ready_rem	Q 4.1	BOOL	Удаление деталей с транспортера (ready to remove, готовность к удалению)
Belt_mot1_on	Q 5.0	BOOL	Включает мотор транспортера для транспортера 1
Load	M 2.0	BOOL	Команда загрузки деталей
Remove	M 2.1	BOOL	Команда удаления деталей
EM_Rem_N	M 2.2	BOOL	Меркер фронта для отрицательного фронта «remove»
EM_Rem_P	M 2.3	BOOL	Меркер фронта для положительного фронта «remove»
EM_Loa_N	M 2.4	BOOL	Меркер фронта для отрицательного фронта «load»
EM_Loa_P	M 2.5	BOOL	Меркер фронта для положительного фронта «load»

Программа для LAD

Пример располагается в функциональном блоке, который вы вызываете в организационном блоке OB 1 (выбран из каталога программных элементов «FC Blocks», «Блоки FC») для обработки в CPU.

Здесь будем программировать пример с блочными элементами памяти. В главе 19 «Параметры блока» показан тот же пример с использованием защелок (блокировок). Программа данной главы находится в функциональном блоке с параметрами блока, которые также могут быть вызваны так часто, как это необходимо (для нескольких транспортеров).

При программировании глобальные символы также могут использоваться без апострофов, что не допускает в их составе специальных литер. Если символ содержит специальную литеру (такую, как умлаут или пробел), то он должен быть заключен в апострофы. В компилированном блоке редактор помечает все глобальные символы, заключая их в апострофы.

На рисунке 5.10 показана программа для системы управления конвейером. На дискете, поставляемой с книгой, вы можете найти эту программу в библиотеке «LAD_Book» в функциональном блоке FC 11 в разделе «Conveyor Example» («Пример конвейера»).

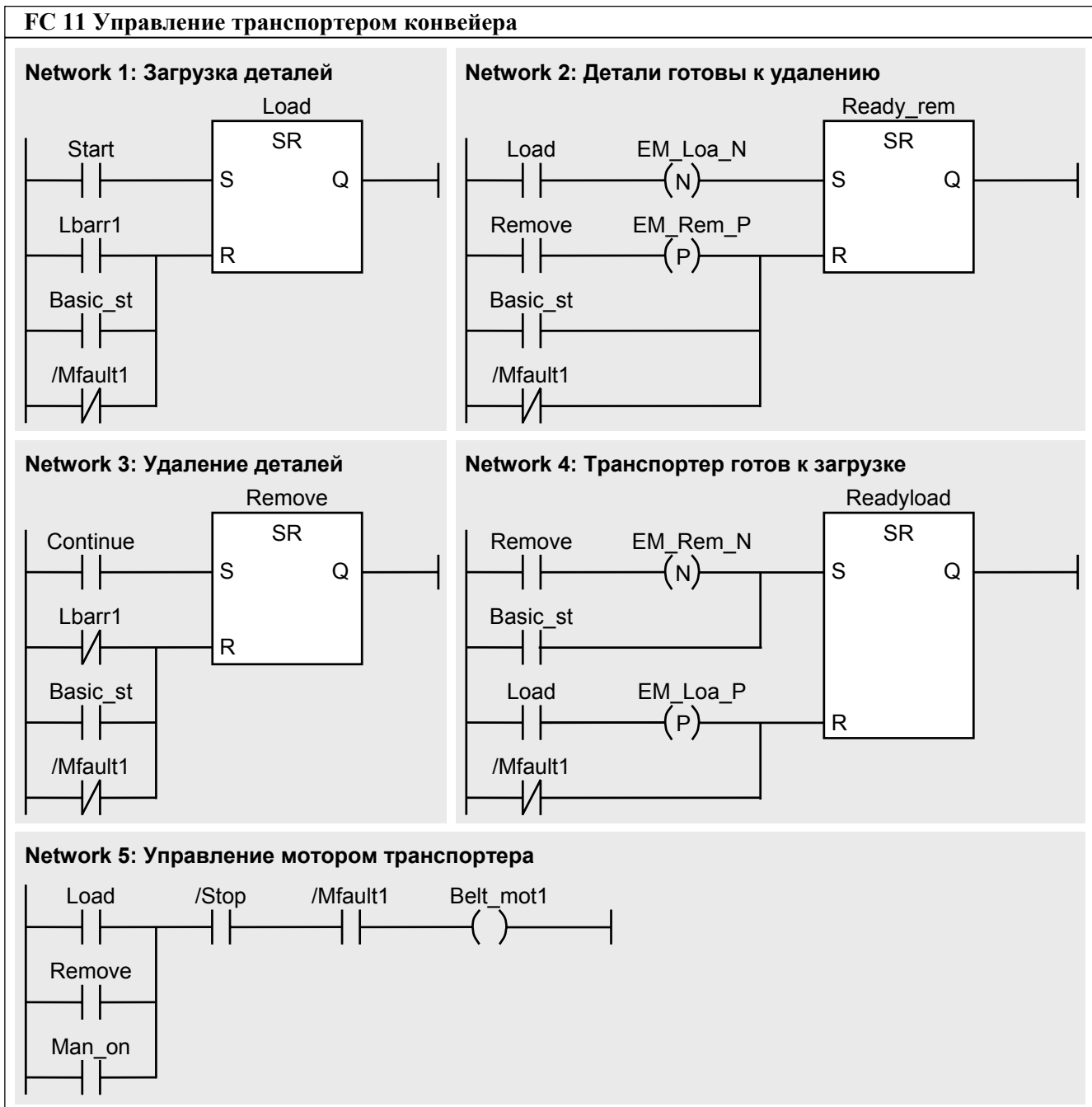


Рисунок 5.10 Образец системы управления конвейером (LAD)

Программа для FBD

Пример располагается в функции, которую вы вызываете в организационном блоке ОВ 1 (выбран из каталога программных элементов «FC Blocks», «Блоки FC») для обработки в CPU.

В главе 19 «Параметры блока» тот же пример приведен с использованием защелок (блокировок). Программа данной главы находится в функциональном блоке с параметрами блока, которые также могут быть вызваны так часто, как это требуется (для нескольких транспортеров).

При программировании глобальные символы также могут использоваться без апострофов, что не допускает в их составе специальных литер. Если символ содержит специальную литеру (такую, как умлаут или пробел), то он должен быть заключен в апострофы. В компилированном блоке редактор помечает все глобальные символы, заключая их в апострофы.

На рисунке 5.11 (а, б) показана программа для системы управления конвейером. На дискете, поставляемой с книгой, вы можете найти эту программу в библиотеке «FBD_Book» в функциональном блоке FC 11 в разделе «Conveyor Example» («Пример конвейера»).

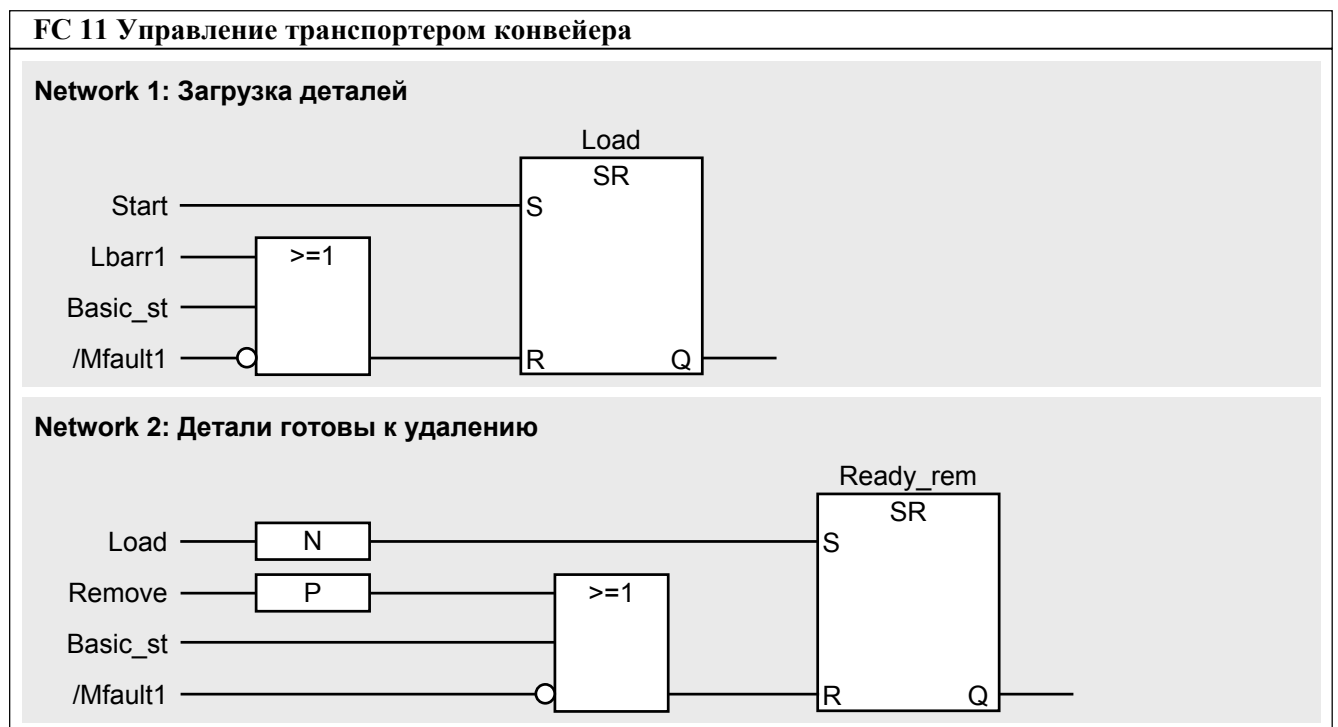


Рисунок 5.11а Пример системы управления конвейером (FBD)

5.5 Бинарный делитель частоты

Бинарный делитель частоты (binary scaler), или счетчик, имеет один вход и один выход. Если сигнал на входе бинарного делителя частоты меняет свое состояние, например с «0» на «1», выход также меняет свое состояние (рисунок 5.7). Это «новое» сигнальное состояние сохраняется до следующего, в нашем случае положительного, изменения сигнального состояния. Только после этого происходит изменение сигнального состояния на выходе. Это означает, что на выходе бинарного делителя частоты входная частота уменьшается на половину.

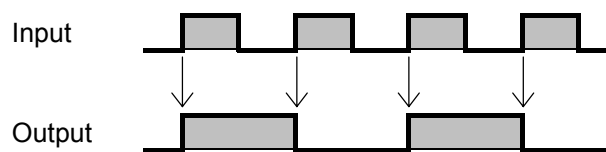


Рисунок 5.7 Импульсная диаграмма бинарного делителя частоты

5.5.1 Решение в LAD

Существует много различных способов решения этой задачи, два из которых представлены ниже.

Первое решение использует функции памяти (рисунок 5.8, сети 8 и 9, Network 8, 9). Если сигнальное состояние операнда *Input_1* (*Вход_1*) «1», то операнд *Output_1* (*Выход_1*) устанавливается; операнд *Memory_1* (*Память_1*) все еще обнулен. Если сигнальное состояние операнда *Input_1* (*Вход_1*) меняется на «0», *Memory_1* (*Память_1*) также устанавливается (*Output_1* (*Выход_1*) теперь «1»). Если снова *Input_1* (*Вход_1*) равен «1», *Output_1* (*Выход_1*) опять обнуляется (*Memory_1* (*Память_1*) теперь «1»). Если *Input_1* (*Вход_1*) еще раз будет равен «0», *Memory_1* (*Память_1*) сбрасывается (так как *Output_1* (*Выход_1*) опять обнулен). Таким образом, базовое состояние было снова получено после двух входных импульсов и одного выходного.

Второе решение применяет функцию блокировки, или защелки (сети 20 и 21, Network 20, 21) в цепных диаграммах. Принцип тот же, что и в первом решении, за исключением того, что условие сброса – как обычно при блокировке – «нулевая активность» («zero active»).

5.5.2 Решение в FBD

Данную задачу можно решить по-разному. Два способа решения приведены ниже.

Первое решение использует функции памяти (рисунок 5.9, сети 16 и 17, Network 16, 17). Если сигнальное состояние операнда *Input* (*Вход*) «1», то операнд *Output* (*Выход*) устанавливается (операнд *Memory* (*Память*) пока сброшен). Если сигнальное состояние операнда *Input* (*Вход*) меняется на «0», то *Memory* (*Память*) также устанавливается (*Output* (*Выход*) теперь «1»). Если снова *Input* (*Вход*) равен «1», то *Output*

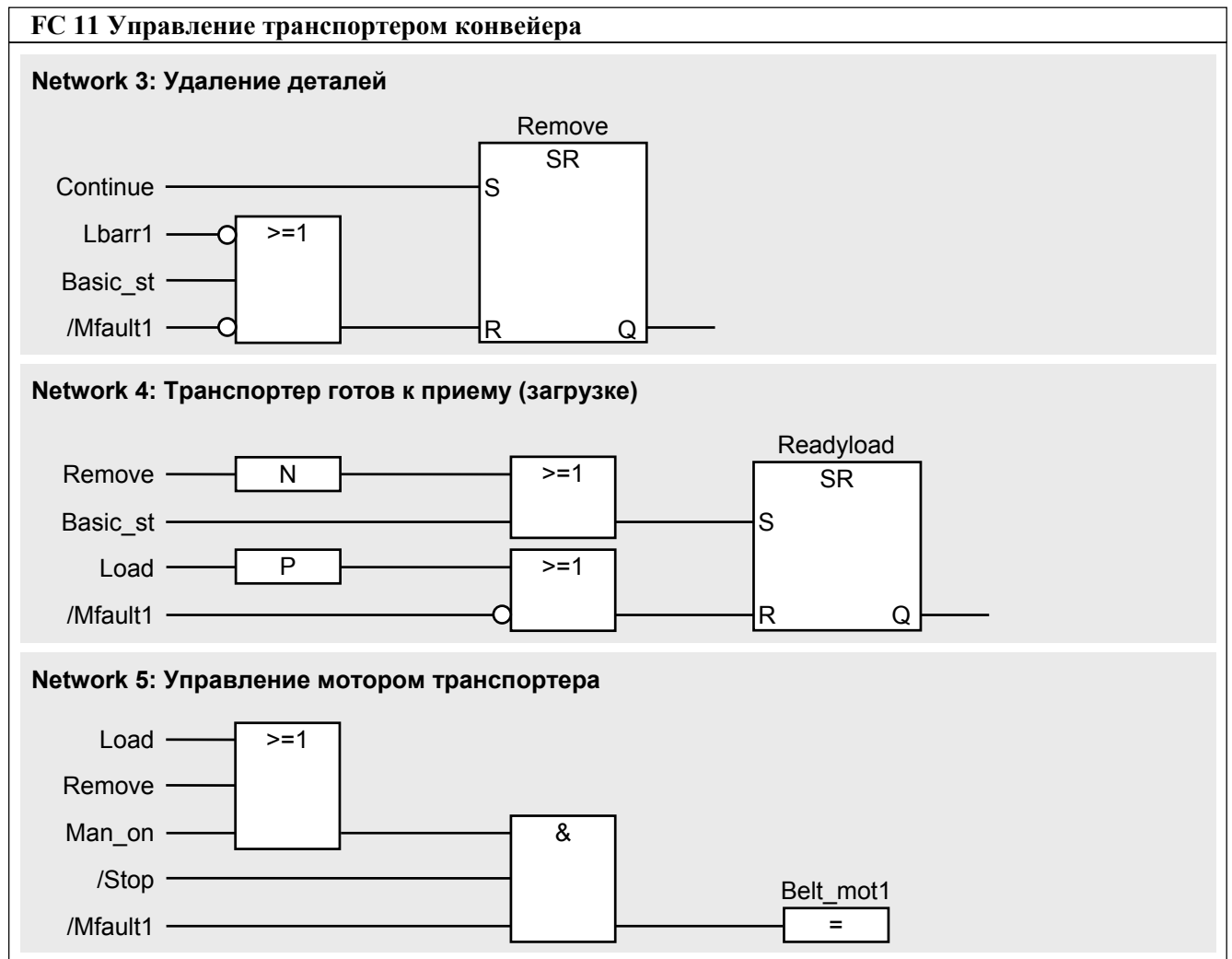


Рисунок 5.116 Пример системы управления конвейером (FBD)

Содержание главы 6

6	<u>Функции передачи</u>	4
6.1	<u>Общие замечания</u>	4
6.2	<u>Блочный элемент MOVE</u>	6
6.2.1	<u>Обработка блочного элемента MOVE</u>	6
6.2.2	<u>Перемещение операндов</u>	9
6.2.3	<u>Передача констант</u>	11
6.3	<u>Системные функции для передачи данных</u>	12
6.3.1	<u>Указатель типа ANY</u>	12
6.3.2	<u>Копирование области данных</u>	13
6.3.3	<u>Непрерывное копирование области данных</u>	14
6.3.4	<u>Заполнение области данных</u>	14

6 Функции передачи

Языки программирования LAD и FBD предоставляют следующие move-функции (функции пересылки, перемещения данных):

- Блочный элемент MOVE
Копирует операнды и переменные простых типов данных;
- SFC 20 BLKMOV
Копирует область данных;
- SFC 21 FILL
Заполняет область данных;
- SFC 81 UBLKMOV
Непрерывное копирование области данных.

SFC – это системные функции из стандартной библиотеки *Standard Library* программы *System Function Blocks* (*Системные функциональные блоки*).

6.1 Общие замечания

Функции пересылки данных используются для копирования информации между системной памятью (system memory), пользовательской памятью (user memory) и областью пользовательских данных модулей (рисунок 6.1). Информация передается через внутренний регистр CPU, который функционирует как промежуточное запоминающее устройство (промежуточная память).

Регистр называется аккумулятором 1 (accumulator 1). Перемещение информации из памяти в аккумулятор 1 называется «загрузка» («loading»), а перемещение из аккумулятора 1 в память называется передачей или пересылкой («transferring»). Блочный элемент MOVE содержит оба пути передачи. Он перемещает информацию со входа IN в аккумулятор 1 (загружает) и сразу после этого – из аккумулятора 1 в операнд на выходе (передает).

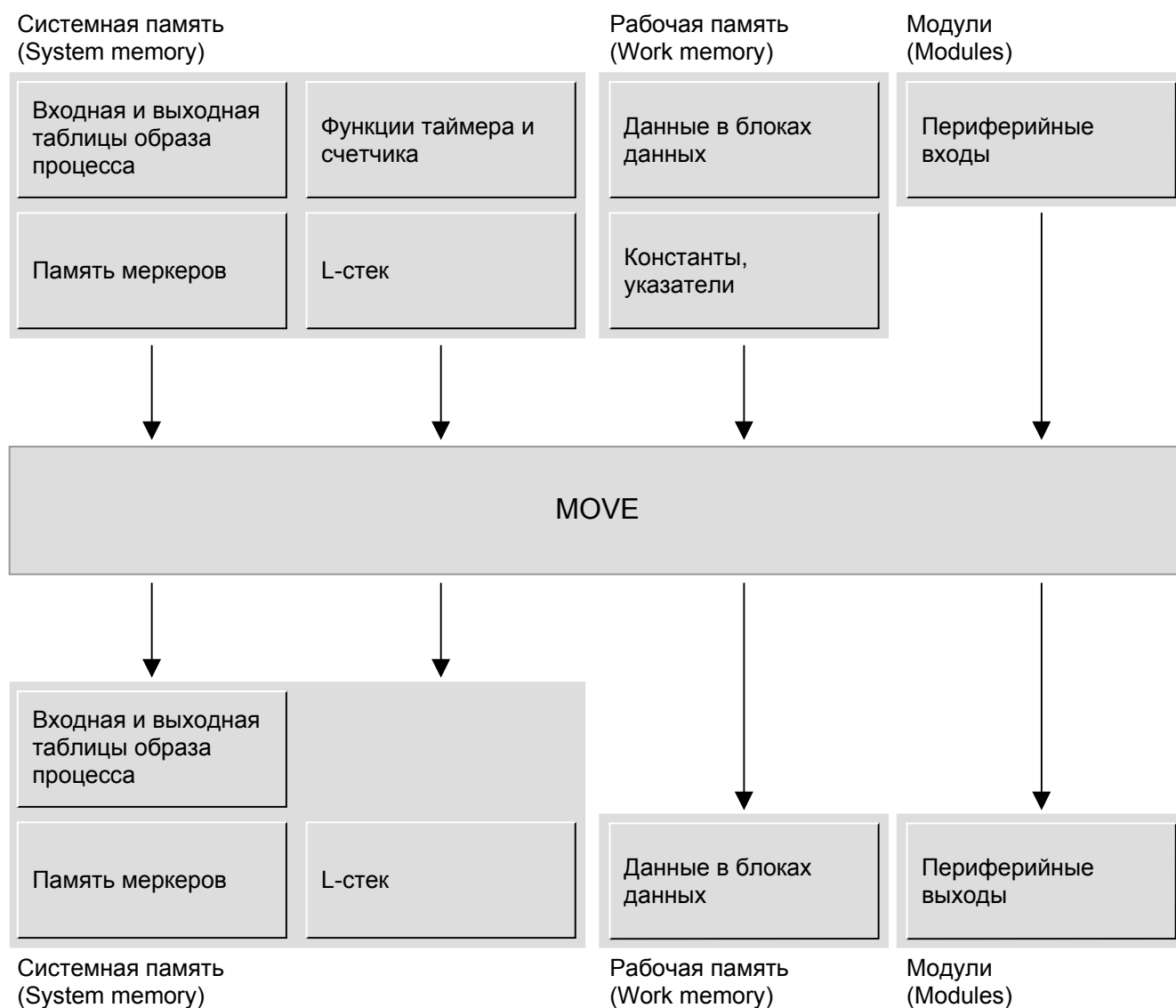


Рисунок 6.1 Области памяти для загрузки (Loading) и передачи (Transferring)

6.2 Блочный элемент MOVE

6.2.1 Обработка блочного элемента MOVE

Представление

В дополнение к разрешающему входу (enable input) EN и разрешающему выходу ENO блочный элемент MOVE имеет вход IN и выход OUT. На входе IN и выходе OUT вы можете задействовать все цифровые операнды и цифровые переменные простых типов данных (за исключением BOOL – логического типа). Переменные у входа IN и выхода OUT могут иметь разные типы данных.

Блочный элемент MOVE



Содержимое и назначение битов в форматах данных подробно обсуждены в параграфе 3.5 «Переменные, константы и типы данных».

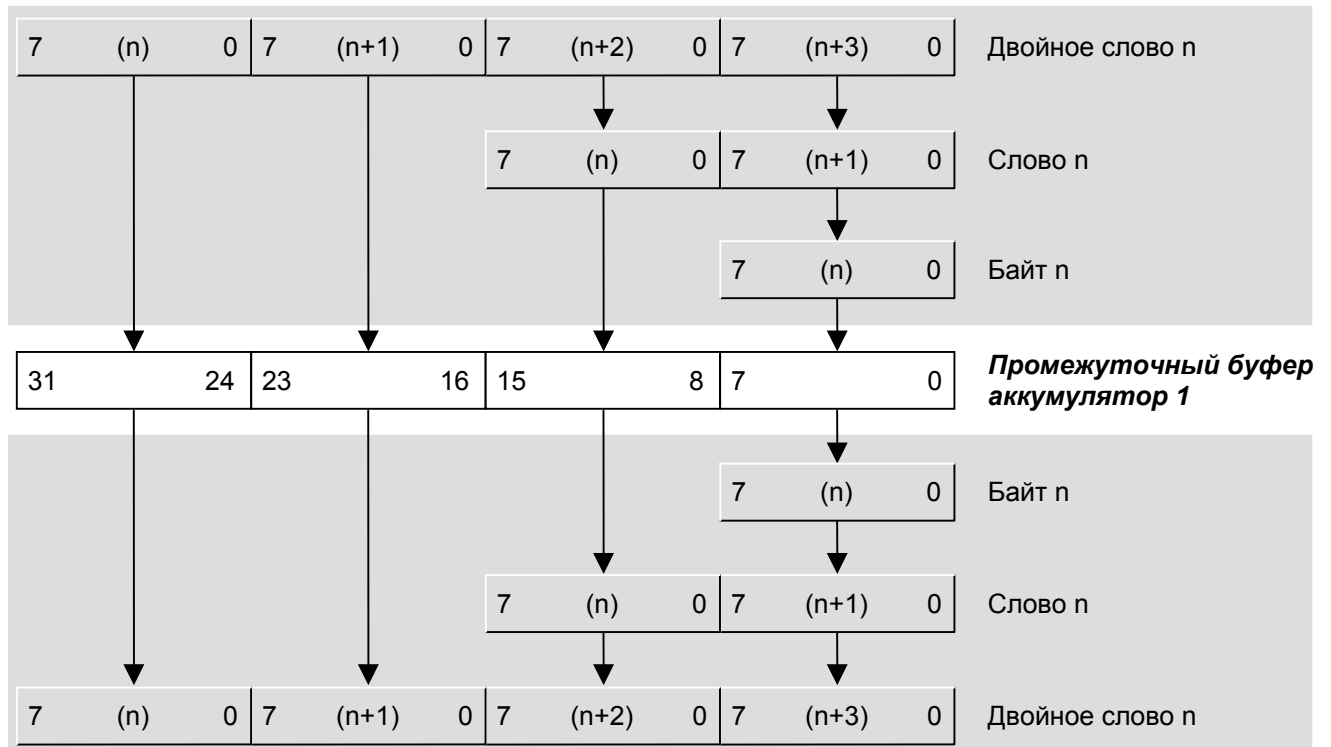
Блочный элемент MOVE, который можно использовать в пошаговом программировании, вы найдете в каталоге программных элементов (Program Element Catalog), выбрав команду меню View → Catalog (Вид → Каталог) [Ctrl + K] или Insert → Program Elements (Вставка → Программные элементы) в разделе «Shift» («Сдвиг»).

Различные размеры операндов

Размеры операндов (байт, слово, двойное слово) на входе и выходе блочного элемента MOVE могут быть разными. Если операнд на входе «меньше», чем на выходе, он передается в выходной операнд с правым выравниванием (значение передаваемого операнда заполняет правые биты результата) и заполнением нулями левой части. Если входной операнд «больше» выходного операнда, то передается только правая часть операнда, соответствующая выходному операнду.

Рисунок 6.2 объясняет этот факт. Байт или слово на входе загружается с правым выравниванием в аккумулятор 1, а оставшаяся часть заполняется нулями. Байт или слово на выходе OUT извлекается из аккумулятора 1 с правым выравниванием.

Вход IN



Выход OUT

Рисунок 6.2 Передача операндов различных размеров

Функция

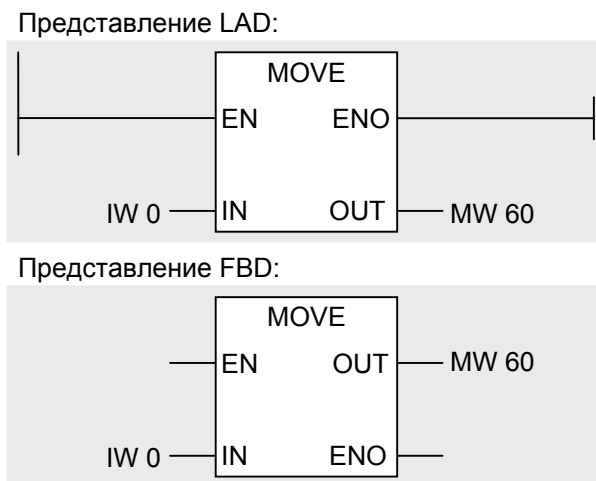
Блочный элемент MOVE перемещает информацию операнда на входе IN в операнд на выходе OUT. Блочный элемент MOVE только тогда совершает передачу информации, когда разрешающий вход (enable input) равен «1» или не используется, и когда главное реле управления (master control relay, MCR) не активировано.

Если EN = «1» и MCR активировано, то на выход OUT записывается нуль. При наличии «0» на разрешающем входе операнд на выходе OUT остается неизменным. MOVE об ошибке не сообщает.

ЕСЛИ EN == "1" или не используется		ИНАЧЕ
ТО		
ENO := "1"		ENO := "0"
ЕСЛИ MCR активно		
ТО	ИНАЧЕ	
OUT := 0	OUT := 1	

Пример

Содержимое входного слова (input word) IW 0 передается в слово памяти (memory word) MW 60.



Блочный элемент MOVE в цепи (LAD)

Перед входом EN и после выхода ENO вы можете выстроить контакты последовательно или параллельно.

Блочный элемент MOVE должен быть помещен только на ветвь, соединенную непосредственно с левой направляющей (питающей шиной). Эта ветвь также может иметь контакты перед входом EN, и это не должна быть верхняя ветвь. С помощью прямого соединения с левой направляющей (питающей шиной) вы можете подключить блочные элементы MOVE параллельно. При параллельном соединении блочных элементов вам необходима катушка, чтобы завершить цепь. Если вы не предусмотрели оценку ошибки, назначьте катушке «фиктивный» операнд, например, бит временных локальных данных.

Вы можете соединить блочные элементы MOVE последовательно. При этом выход ENO предшествующего элемента соединяется с входом EN следующего.

Если вы сконструировали несколько блочных элементов MOVE в одной цепи (параллельно у левой направляющей, или питающей шины, и далее продолжив последовательно), то блочные элементы самой верхней ветви обрабатываются первыми слева направо, затем элементы параллельной ветви слева направо и так далее.

Примеры функций передачи вы можете найти на дискете, прилагаемой к книге (библиотека «LAD_Book» в функциональном блоке FB 106 программы «Basic Functions», «Основные функции»).

Блочный элемент MOVE в логической схеме (FBD)

Если вы хотите обработать блочный элемент MOVE в зависимости от определенных условий, вы можете запрограммировать бинарные логические схемы перед входом EN. Вы можете соединить выход ENO с двоичными входами других функций; к примеру, можно скомпоновать блочные элементы MOVE последовательно, где выход ENO предыдущего элемента соединен с входом EN последующего блочного элемента.

EN и ENO не должны быть жестко смонтированы.

Примеры функций передачи вы можете найти на дискете, прилагаемой к книге, в библиотеке «FBD_Book», FB 106 в программе «Basic Functions» («Основные функции»).

6.2.2 Перемещение операндов

Кроме упомянутых в данной главе операндов, вы можете также пересылать значения таймеров и счетчиков (см. главу 7 «Таймеры» и главу 8 «Счетчики»). Параграф 18.2 «Функции блока для блоков данных» рассказывает об использовании операндов данных.

Передача входов

IB n	Передача входного байта
IW n	Передача входного слова
ID n	Передача входного двойного слова

В некоторых CPU загрузка из и передача во входную таблицу образа процесса допустимы только для входных байтов, которые также доступны как входной модуль (см. параграф 1.5.2 «Образ процесса»).

Передача выходов

QB n	Передача выходного байта
QW n	Передача выходного слова
QD n	Передача выходного двойного слова

В некоторых CPU загрузка из выходной таблицы образа процесса и передача в нее допустимы только для выходных байтов, которые также доступны как выходной модуль (см. параграф 1.5.2 «Образ процесса»).

Передача из I/O (периферийных входов/выходов)

- PIB n Загрузка байта периферийного входа
- PIW n Загрузка слова периферийного входа
- PID n Загрузка двойного слова периферийного входа
- PQB n Передача байта в периферийный выход
- PQW n Передача слова в периферийный выход
- PQD n Передача двойного слова в периферийный выход

При передаче данных в области периферийных входов/выходов (в I/O-области) вы можете получить доступ к различным операндам в зависимости от направления передачи. Вы должны определить I/O-входы (входы PI) при входе IN блочного элемента MOVE и I/O-выходы (выходы PQ) при выходе OUT.

Когда происходит передача из I/O (периферийного входа/выхода) (операция загрузки), доступ к входным модулям осуществляется как к периферийным входам (peripheral inputs, PI). Адресоваться могут только доступные модули. Обратите внимание, что прямая загрузка из I/O-модулей (модулей периферийных входов/выходов) может пересылать значение, отличное от загружаемого из входов модуля с теми же адресами. Пока сигнальное состояние входов соответствует значениям в начале программного цикла (когда CPU обновил образ процесса), значения, загруженные непосредственно из I/O-модулей, являются текущими значениями.

Периферийные выходы (peripheral outputs, PQ) используются для передач в I/O. Доступ возможен только к тем адресам, по которым расположены I/O-модули. Передача в I/O-модули, имеющие таблицу образа процесса, одновременно обновляет эту выходную таблицу образа процесса, поэтому между одинаково адресуемыми выходами и периферийными выходами различий нет.

Передача меркеров

- MB n Передача байта меркерной памяти
- MW n Передача слова меркерной памяти
- MD n Передача двойного слова меркерной памяти

Передача из и в область адресов памяти меркеров всегда допустима, так как вся память меркеров расположена в CPU. Заметьте, что у CPU разных версий размеры области памяти меркеров различны.

Передача временных локальных данных

LB n	Передача байта локальных данных
LW n	Передача слова локальных данных
LD n	Передача двойного слова локальных данных

Перемещение из и в L-стек всегда разрешено. За информацией по этому вопросу обратитесь к параграфу 18.1.5 «Временные локальные данные».

6.2.3 Передача констант

Определить константы можно только на входе IN блочного элемента MOVE.

Передача констант простых типов данных

Фиксированное значение, или константа, может быть переслано в операнд. Иначе говоря, эта константа может быть передана в нескольких различных форматах. В параграфе 3.5.4 «Простые типы данных» вы найдете обзор различных имеющихся форматов. Все константы, которые могут быть переданы с помощью блочного элемента MOVE, принадлежат простым типам данных. Примеры:

V#16#F1	Передача двузначного шестнадцатеричного числа
- 1000	Передача целого числа (типа INT)
5.0	Передача вещественного числа (типа REAL)
S5T#2s	Передача таймера S5
TOD#8:30	Передача времени суток

Передача указателей

Указатели (pointers) – это особый вид констант, используемый для вычисления адресов в стандартных блоках. Вы можете использовать блочный элемент MOVE для сохранения этих указателей в операндах.

P#1.0	Передача внутреннего указателя области (area-internal pointer)
P#M2.1	Передача указателя пересечения областей (area-crossing pointer)

6.3 Системные функции для передачи данных

Для передачи данных доступны следующие системные функции:

- SFC 20 BLKMOV
Копирует область данных;
- SFC 21 FILL
Заполняет область данных;
- SFC 81 UBLKMOV
Непрерывное копирование области данных.

Эти системные функции имеют два параметра типа ANY (таблица 6.1). В принципе, вы можете назначить этим параметрам любой операнд, любую переменную или любую абсолютно адресуемую область. Если вы используете переменную сложного типа данных, то это может быть только «вся» переменная (целиком); компоненты переменной (отдельный массив или структурные компоненты, например) недопустимы. Для определения абсолютно адресуемой области используйте указатель типа ANY (Любой).

Таблица 6.1 Параметры для SFC 20, 21 и 81

SFC	Параметр	Объявление	Тип данных	Содержимое, характеристика
20	SRCBLK	INPUT	ANY	Исходная область, из которой копируются данные
	RET_VAL	OUTPUT	INT	Информация об ошибке
	DSTBLK	OUTPUT	ANY	Область назначения, в которую копируются данные
21	BVAL	INPUT	ANY	Копируемая исходная область
	RET_VAL	OUTPUT	INT	Информация об ошибке
	BLK	OUTPUT	ANY	Область назначения, в которую копируется исходная область (включая несколько копий)
81	SRCBLK	INPUT	ANY	Исходная область, из которой копируются данные
	RET_VAL	OUTPUT	INT	Информация об ошибке
	DSTBLK	OUTPUT	ANY	Область назначения, в которую копируются данные

6.3.1 Указатель типа ANY

Когда требуется определить область операнда, адресуемую абсолютно, в качестве параметра блока типа ANY, используется указатель данного типа. Общий формат ANY-указателя следующий:

R#[DataBlock]Operand Type Quantity (R#[БлокДанных]Операнд Тип Количество)

Примеры:

P#M16.0 BYTE 8	Область из 8 байт, начинающаяся с MB 16
P#DB11.DBX30.0 INT 12	Область из 12 слов в DB 11, начинающаяся с DBB 30
P#I18.0 WORD 1	Входное слово IW 18
P#I1.0 BOOL 1	Вход I 1.0

Обратите внимание, что адрес операнда в указателе типа ANY всегда должен быть адресом бита.

Определить постоянный ANY-указатель имеет смысл, когда вы хотите получить доступ к области данных, для которой переменных не объявлено. В принципе, ANY-параметру вы можете назначить переменные или операнды. Например, 'P#I1.0 BOOL 1' идентично 'I 1.0' или соответствующему символическому адресу.

6.3.2 Копирование области данных

Системная функция **SFC 20 BLKMOV** копирует содержимое исходной области (параметр SLCBLK) в область назначения (параметр DSTBLK) в направлении роста адресов (пошагового увеличения).

Могут быть назначены следующие фактические параметры:

- Любые переменные из областей операндов для входов (I), выходов (Q), меркеров (M) и блоков данных (переменные из глобальных блоков данных и из экземплярных блоков данных);
- Переменные из временных локальных данных (использование типа данных ANY определяется особыми обстоятельствами);
- Абсолютно адресуемые области данных, которые требуют определения ANY-указателя.

SFC 20 нельзя использовать для копирования таймеров или счетчиков, для копирования информации из или в модули (область операнда P) или копирования системных блоков данных (блоков SDB).

В случае входов и выходов определенная область копируется независимо от того, указывают фактически адреса на модули входов или выходов или нет. Вы также можете определить переменную или область из блока данных в загрузочной памяти (блок данных, запрограммированный с ключевым словом UNLINKED, Несвязанный) в качестве исходной области.

Исходная область и область назначения не могут перекрываться. Если исходная область и область назначения имеют разные размеры, передача происходит, исходя из длины наименьшей из двух областей.

Пример (рисунки 6.3а, 6.3б, сеть 4, Network 4): начиная с байта памяти (меркеров) MB 64, 16 байт копируются в блок данных DB 124, начиная от DBB 0.

6.3.3 Непрерывное копирование области данных

Системная функция **SFC 81 UBLKMOV** копирует содержимое исходной области (параметр SRCBLK) в область назначения (параметр DSTBLK) в направлении возрастания адресов (пошагового увеличения). Функция копирования непрерывна, создает возможность увеличенных временных интервалов откликов на прерывания. Скопировано может быть максимально 512 байт.

Могут быть назначены следующие фактические параметры:

- Любые переменные из областей операндов для входов (I), выходов (Q), меркеров (M) и блоков данных (переменные из глобальных блоков данных и из экземплярных блоков данных);
- Переменные из временных локальных данных (использование типа данных ANY определяется особыми обстоятельствами);
- Абсолютно адресуемые области данных, которые требуют определения ANY-указателя.

SFC 81 не может использоваться для копирования таймеров или счетчиков, для копирования информации из или в модули (область операнда P) или для копирования системных блоков данных (блоков SDB) или блоков данных в загрузочной памяти (блок данных, запрограммированный с ключевым словом UNLINKED, Несвязанный).

В случае входов и выходов определенная область копируется независимо от того, указывают их адреса на модули входов или выходов или нет.

Исходная область и область назначения не могут перекрываться. Если исходная область и область назначения имеют разные размеры, передача происходит, исходя из длины наименьшей из двух областей.

6.3.4 Заполнение области данных

Системная функция **SFC 21 FILL** копирует установленное значение (исходную область) в область памяти (область назначения) с целью полной перезаписи области назначения. Передача производится в направлении возрастания адресов (пошагового увеличения). Могут быть назначены следующие фактические параметры:

- Любые переменные из областей операндов для входов (I), выходов (Q), меркеров (M) и блоков данных (переменные из глобальных блоков данных и из экземплярных блоков данных);

- Абсолютно адресуемые области данных, которые требуют определения ANY-указателя;
- Переменные из временных локальных данных (использование типа данных ANY определяется особыми обстоятельствами).

SFC 21 не может использоваться для копирования таймеров или счетчиков, для копирования информации из или в модули (область операнда P) или системных блоков данных (блоков SDB).

В случае входов и выходов определенная область копируется независимо от того, указывают фактически адреса на модули входов или выходов или нет.

Исходная область и область назначения не могут перекрываться. Область назначения всегда полностью перезаписывается, даже когда исходная область больше области назначения, или когда длина области назначения не кратна целому числу размеров исходных областей.

Пример (рисунки 6.3а, 6.3б, сеть 5, Network 5): содержимое байта памяти (меркеров) MB 80 копируется 16 раз в блок данных DB 124, начиная с DBB 16.

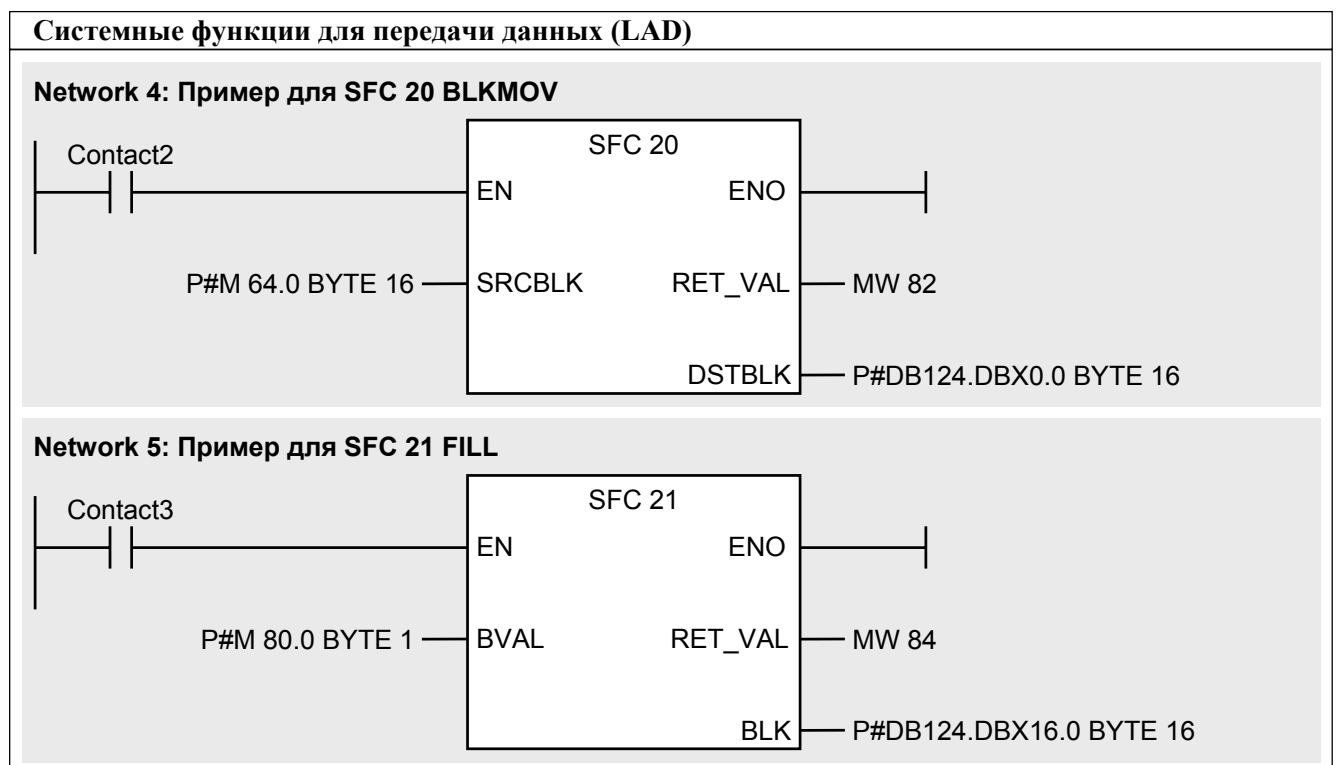


Рисунок 6.3а Примеры для SFC 20 BLKMOV и SFC 21 FILL

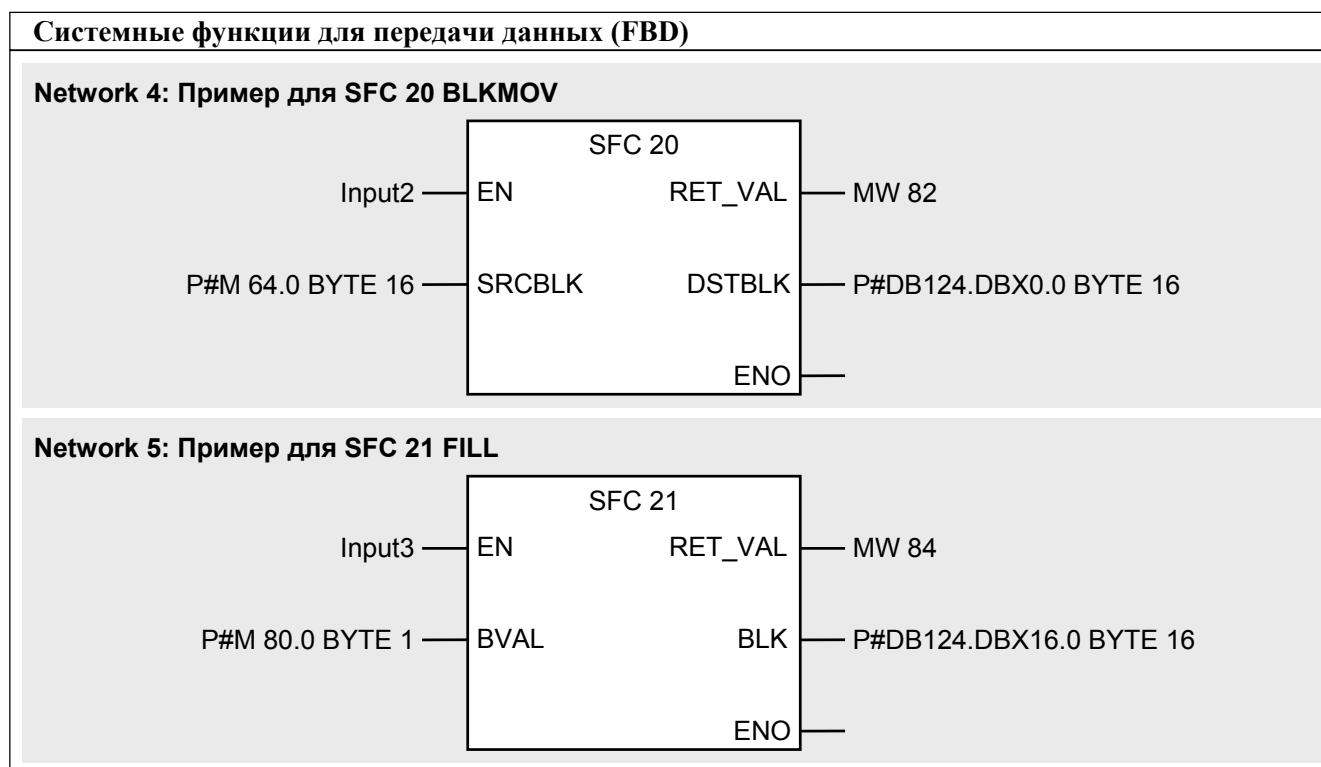


Рисунок 6.36 Примеры для SFC 20 BLKMOV и SFC 21 FILL

Содержание главы 7

7	<u>Таймеры</u>	3
7.1	<u>Программирование таймера</u>	4
7.1.1	<u>Общее представление о таймере</u>	4
7.1.2	<u>Запуск таймера</u>	6
7.1.3	<u>Определение длительности времени</u>	6
7.1.4	<u>Сброс таймера</u>	8
7.1.5	<u>Опрос таймера</u>	8
7.1.6	<u>Последовательность операций с таймерами</u>	9
7.1.7	<u>Блочный элемент таймера в цепи (LAD)</u>	10
7.1.8	<u>Блочный элемент таймера в логической схеме (FBD)</u>	10
7.2	<u>Импульсный таймер</u>	11
7.3	<u>Расширенный импульсный таймер</u>	13
7.4	<u>Таймер задержки включения</u>	15
7.5	<u>Таймер задержки включения с запоминанием</u>	17
7.6	<u>Таймер задержки выключения</u>	19
7.7	<u>IEC-таймеры</u>	21
7.7.1	<u>Импульсный таймер SFB 3 TP</u>	22
7.7.2	<u>Таймер задержки включения SFB 4 TON</u>	22
7.7.3	<u>Таймер задержки выключения SFB 5 TOF</u>	23

7 Таймеры

Таймеры (Timers) позволяют программно реализовать последовательности синхронизации, такие как интервалы ожидания и наблюдения, измерение интервалов или генерирование импульсов.

Существуют следующие типы таймеров:

- Импульсные таймеры (Pulse timers);
- Расширенные импульсные таймеры (Extended pulse timers);
- Таймеры задержки включения (On-delay timers);
- Таймеры задержки включения с запоминанием (Retentive on-delay timers);
- Таймеры задержки выключения (Off-delay timers).

Вы можете запрограммировать таймер полностью как блочный элемент или с помощью отдельных программных элементов. При запуске таймера вы определяете требуемый тип таймера, и как долго он должен работать; также вы можете сбросить таймер. Таймер контролируется путем запроса его состояния («Timer running», «Таймер работает») или текущего значения таймера, которое вы можете получить из таймера в двоичном или двоично-десятичном (BCD) коде.

Блочный элемент таймера

(в примере: импульсный таймер)



Наименование	Тип данных	Описание
S	BOOL	Вход запуска
TV	S5TIME	Спецификация длительности времени
R	BOOL	Вход сброса
BI	WORD	Текущее значение времени в двоичном представлении
BCD	WORD	Текущее значение времени в BCD-представлении
Q	BOOL	Состояние таймера

Рисунок 7.1 Таймер в виде блочного элемента (timer box)

7.1 Программирование таймера

7.1.1 Общее представление о таймере

Вы можете выполнять следующие операции над таймером:

- Запустить таймер, задав значение времени;
- Обнулить (сбросить) таймер;
- Проверить (двоичное) состояние таймера;
- Прочитать (цифровое) значение таймера в бинарном виде;
- Прочитать (цифровое) значение таймера в BCD-виде.

Блочный элемент для таймера содержит все эти операции таймера в форме функциональных входов и функциональных выходов (рисунок 7.1). Над блочным элементом расположен абсолютный или символический адрес таймера. В самом блочном элементе в качестве заголовка указан режим таймера (S_PULSE означает «Start pulse timer» или «Запуск импульсного таймера»). Назначения для входов S и TV обязательны, в то время как для других – нет.

Отдельные программные элементы в LAD

Также можно запрограммировать таймер с использованием отдельных программных элементов (рисунок 7.2). В этом случае запуск таймера осуществляется посредством катушки. Режим таймера указан в катушке (SP = start pulse timer или запуск импульсного таймера), а под катушкой расположено значение в формате S5TIME, определяющее длительность (duration). Чтобы обнулить таймер, используйте катушку сброса. NO- и NC-контакты можно применить для проверки состояния таймера. Наконец, вы можете сохранить текущее значение времени в двоичной форме в операнде размером в слово с помощью блочного элемента MOVE.

Отдельные программные элементы в FBD

Вы также можете запрограммировать таймер с помощью отдельных программных элементов (рисунок 7.3). В данном случае таймер стартует посредством простого блочного элемента, содержащего режим таймера (SP = start pulse timer или запуск импульсного таймера). Под элементом находится значение в формате S5TIME, определяющее длительность. Чтобы обнулить таймер, используйте блочный элемент сброса. Сканировать состояние таймера можно непосредственно или в инвертированном виде с использованием любого бинарного входа. Наконец, вы можете сохранить текущее значение времени в двоичной форме в операнде размером в слово с помощью блочного элемента MOVE.

Запуск таймера с определенным значением времени (катушка запуска с рабочими характеристиками)	
Сброс таймера (катушка сброса)	
Проверка состояния таймера (NO-контакт, NC-контакт)	
Чтение значения времени в двоичной форме (блочный элемент MOVE)	

Рисунок 7.2 Отдельные элементы таймера (LAD)

Для создания программ с помощью пошагового программирования элементы-таймеры вы можете найти в каталоге программных элементов (Program Element Catalog), вызываемого командой меню View → Catalog (Вид → Каталог) [Ctrl + K] или Insert → Program Elements (Вставка → Программные элементы), в разделе «Timers» («Таймеры»).

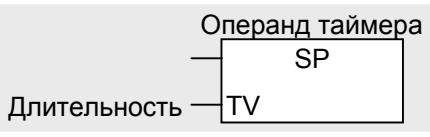
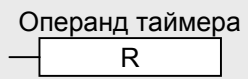
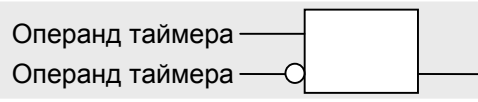
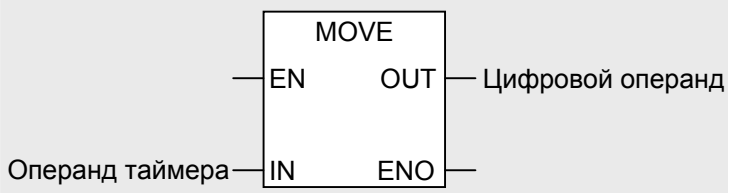
Запуск таймера с определенным значением времени (блочный элемент запуска с рабочими характеристиками)	
Сброс таймера (блочный элемент сброса)	
Проверка состояния таймера (прямой или инвертированный двоичный вход)	
Чтение значения времени в двоичной форме (блочный элемент MOVE)	

Рисунок 7.3 Отдельные элементы таймера (FBD)

7.1.2 Запуск таймера

Таймер стартует, когда результат логической операции (RLO) меняется на входе запуска (start input) или перед катушкой / блочным элементом запуска. Такая смена сигнала всегда требуется, чтобы запустить таймер. В случае таймера задержки выключения RLO должен поменяться с «1» на «0»; все другие таймеры стартуют при смене RLO с «0» на «1».

Вы можете запустить в одном из пяти различных режимов (рисунок 7.4). Однако, нет смысла использовать любой данный таймер более, чем в одном режиме.

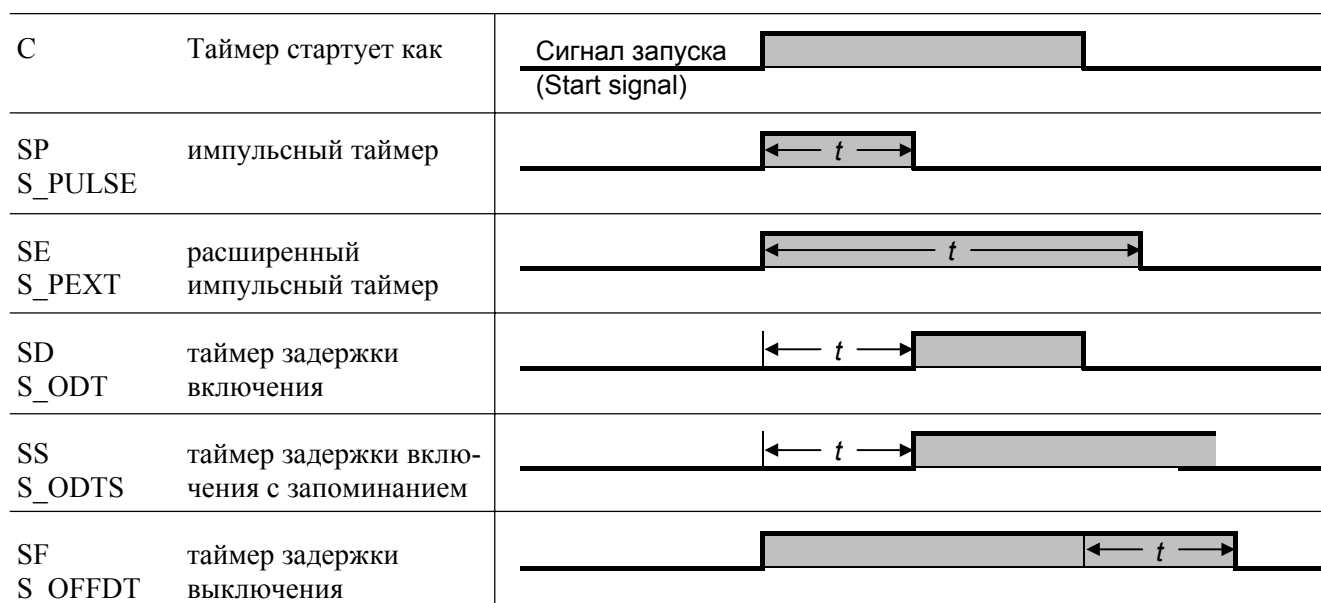


Рисунок 7.4 Рабочие характеристики таймера

7.1.3 Определение длительности времени

Таймер в качестве длительности принимает значение, указанное под катушкой / блочным элементом, или значение на входе TV. Вы можете задать длительность как константу, как операнд размером в слово или как переменную типа S5TIME.

Определение длительности как константы

S5TIME#10s	Длительность 10 с
S5T#1m10ms	Длительность 1 м + 10 мс

Длительность задается в часах, минутах, секундах и миллисекундах. Область значений простирается от S5TIME#10s до S5TIME#2h46min30s (что соответствует 9990 с). Промежуточные значения округляются до 10 мс. Для определения константы вы можете использовать S5TIME# или S5T#.

Определение длительности как операнда или переменной

MW 20 Операнд размером в слово, содержащий длительность
 “Time1” Переменная типа S5TIME

Значение в 16-битном операнде должно соответствовать типу данных S5TIME (см. ниже «Структура значения длительности времени»).

Структура значения длительности времени

Структурно длительность состоит из значения времени (time value) и временной базы (time base): длительность = значение времени × временная база. Длительность представляет собой интервал времени, в течение которого таймер активен («таймер работает»). Значение времени – это число циклов, которые обрабатывает таймер. Временная база определяет интервал, за который CPU должен изменить значение времени (рисунок 7.5).

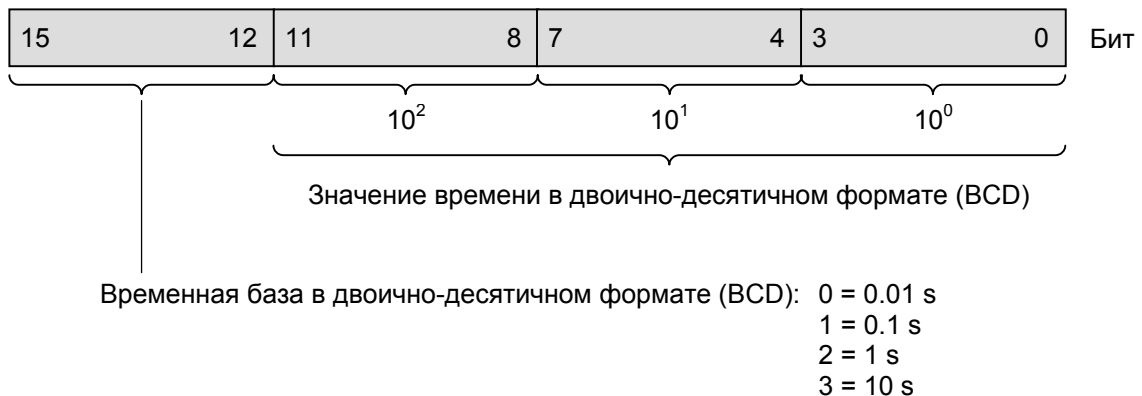


Рисунок 7.5 Описание битов в значении длительности

Также можно задать длительность времени непосредственно в 16-битном операнде. Чем меньше временная база, тем точнее фактическая длительность. Например, если вы хотите реализовать длительность, равную одной секунде, то вы можете указать одну из трех спецификаций:

Длительность = 2001_{hex} Временная база 1 с

Длительность = 1010_{hex} Временная база 100 мс

Длительность = 0100_{hex} Временная база 10 мс

В данном случае последнее предпочтительнее.

При запуске таймера CPU принимает запрограммированное значение времени. Операционная система обновляет таймеры в фиксированные интервалы независимо от пользовательской программы, то есть она уменьшает значение времени всех актив-

ных таймеров согласно временной базе. Когда таймер достигает нуля, он «заканчивается». Затем CPU устанавливает состояние таймера (сигнальное состояние «0» или «1» в зависимости от режима, или «типа», таймера) и приостанавливает всю дальнейшую деятельность до рестарта таймера. Если вы определили нулевую длительность (0 мс или W#16#0000) при запуске таймера, то таймер остается активным, пока CPU не обработает его и не обнаружит, что время истекло.

Таймеры обновляются асинхронно по отношению к сканированию программы. В результате возможна ситуация, при которой состояние таймера в начале цикла отличается от состояния в конце цикла. Если вы используете таймеры только в одной точке программы и в предлагаемом порядке (см. ниже), то асинхронное обновление предотвратит появление ошибок.

7.1.4 Сброс таймера

LAD: таймер сбрасывается, когда электрический ток течет на входе сброса (reset input) или в катушке сброса (reset coil) (когда «RLO» равен «1»). Пока таймер остается сброшенным, сканирование с помощью NO-контакта вернет «0», с помощью NC-контакта – «1».

FBD: таймер обнуляется, когда на входе сброса присутствует «1». Пока таймер остается сброшенным, прямое сканирование состояния таймера возвратит «0», а инвертированное сканирование вернет «1».

Сброс таймера устанавливает данный таймер и временную базу в нуль. Вход R блочного элемента таймера должен быть назначен (то есть на этом входе должно присутствовать фактическое значение).

7.1.5 Опрос таймера

Проверка состояния таймера (LAD)

Состояние таймера можно получить на выходе Q блочного элемента таймера. Узнать состояние таймера вы можете также с помощью NO-контакта (соответствует выходу Q) или с помощью NC-контакта. Результаты считывания с помощью NO-контакта или выхода Q различаются в зависимости от типа таймера (см. описание типов таймера ниже). Как и в случае с входами, например, проверка с помощью NC-контакта генерирует в точности противоположный результат считывания по сравнению с результатом, полученным с помощью NO-контакта. Выход Q не должен использоваться в блочном элементе таймера.

Проверка состояния таймера (FBD)

Состояние таймера доступно на выходе Q блочного элемента таймера. Проверить состояние таймера вы также можете с использованием двоичного выхода функции (соответствует выходу Q). Результаты считывания таймера зависят от типа исполь-

зуемого таймера (см. описание типов таймера ниже). Выход Q может не использоваться в блочном элементе таймера.

Считывание значения времени (time value)

Выходы BI и BCD предоставляют значение времени таймера в двоичном (BI) и двоично-десятичном (BCD) виде. Это значение является текущим в момент считывания (если таймер активен, то значение времени отсчитывается от установленного значения в сторону уменьшения до нуля). Значение сохраняется в заданном операнде (передается как и при использовании блочного элемента MOVE). В блочном элементе таймера эти выходы можно не назначать.

Прямое считывание значения времени

Значение времени доступно в двоично-десятичном коде и может быть получено из таймера в этой форме. В таком случае временная база теряется и заменяется значением «0». Значение соответствует положительному числу в формате INT (Целое). Обратите внимание, что считанное значение времени не является длительностью! Вы также можете запрограммировать непосредственное чтение значения времени с помощью блочного элемента MOVE.

Закодированное считывание значения времени

Также вы можете вывести бинарное значение времени из таймера в «закодированной» форме. Здесь и значение времени, и временная база могут быть получены в двоично-десятичном виде. BCD-значение структурировано также, как для спецификации значения времени (см. выше).

7.1.6 Последовательность операций с таймерами

При программировании таймера не нужно использовать все доступные для этого операции. Вы должны применять только те операции, которые требуются для выполнения конкретной функции. Обычно к этим операциям относятся операции для запуска таймера и для проверки состояния таймера.

Чтобы таймер работал, как описано в этой главе, рекомендуется соблюдать следующий порядок при программировании с использованием отдельных программных элементов:

- Запуск таймера (Start);
- Сброс таймера (Reset);
- Считывание значения времени или длительности;
- Проверка состояния таймера.

При программировании ненужные элементы пропустите. Если вы соблюдаете порядок, показанный выше, и таймер запущен и сброшен «одновременно», то таймер запустится и немедленно обнулится. Последующая проверка таймера не обнаружит тот факт, что таймер был запущен.

7.1.7 Блочный элемент таймера в цепи (LAD)

Контакты перед входом запуска (start input) и входом сброса (reset input) вы можете соединить последовательно и параллельно, как и после выхода Q.

Сам блочный элемент таймера может быть расположен после T-ветви и в ветви, которая непосредственно подключена к левой питающей шине (несущей). В этой ветви также могут быть контакты перед входом запуска, и это не должна быть самой верхней ветвью.

Примеры представления и группировки таймеров вы можете найти в библиотеке «LAD_Book» на прилагаемой к книге дискете, FB 107 программы «Basic Functions» («Основные функции»).

7.1.8 Блочный элемент таймера в логической схеме (FBD)

Перед входами запуска и сброса, как и после выхода Q, можно запрограммировать бинарные функции и функции для работы с памятью.

Блочный элемент таймера и отдельные элементы для запуска и сброса таймера могут также быть запрограммированы после T-ветви.

Примеры представления и группировки таймеров вы можете найти в библиотеке «FBD_Book» на дискете, поставляемой с книгой, FB 107 программы «Basic Functions» («Основные функции»).

7.2 Импульсный таймер

Запуск импульсного таймера

Диаграмма на рисунке 7.6 поясняет характеристики таймера, когда он запускается как импульсный таймер, и когда он сбрасывается. Описание имеет силу, если при программировании с использованием отдельных элементов вы соблюдаете порядок, показанный в параграфе 7.1.6 «Последовательность операций с таймерами».

- ① Когда сигнальное состояние на входе запуска (start input) таймера меняется с «0» на «1» (положительный фронт), таймер запускается. Он работает на протяжении запрограммированной длительности, пока сигнальное состояние на входе запуска равно «1». Во время работы таймера выход Q выдает сигнальное состояние «1».

Имея стартовое значение в качестве начальной точки, значение времени отсчитывается в обратном порядке до нуля согласно временной базе.

- ② Если сигнальное состояние на входе запуска таймера меняется на «0» до истечения времени, то таймер останавливается. Выход Q становится равным «0». Значение времени показывает, сколько времени работал бы таймер еще, не будь он преждевременно прерван.

Сброс импульсного таймера

Сброс импульсного таймера имеет статический эффект и получает приоритет перед запуском таймера (рисунок 7.6).

- ③ Сигнальное состояние «1» на входе сброса (reset input) активного таймера обнуляет этот таймер. Выход Q в этом случае равен «0». Значению времени и временной базе также присваиваются нули. Если сигнальное состояние входа сброса переходит из «1» в «0», в то время как сигнальное состояние при входе установки еще равно «1», то таймер остается неизменным.
- ④ Сигнальное состояние «1» на входе сброса неактивного таймера никакого воздействия не оказывает.
- ⑤ Если сигнальное состояние на входе запуска переходит из «0» в «1» (положительный фронт), пока сигнал сброса еще присутствует, таймер стартует, но немедленно сбрасывается (на диаграмме показано линией). Если проверка состояния таймера была запрограммирована после сброса, то кратковременный запуск таймера не воздействует на проверку.

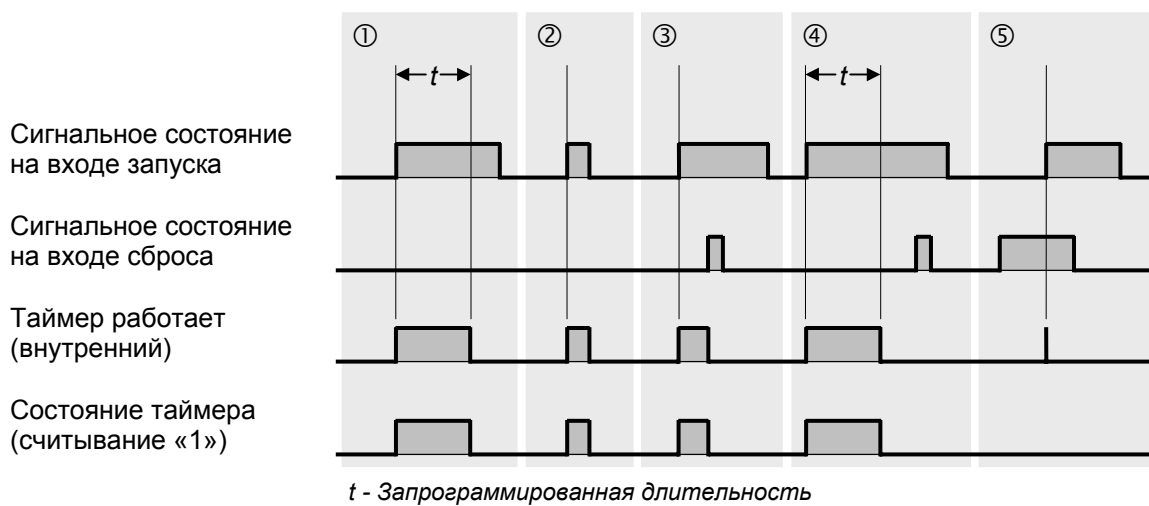


Рисунок 7.6 Рабочие характеристики при запуске и сбросе импульсного таймера

7.3 Расширенный импульсный таймер

Запуск расширенного импульсного таймера

Диаграмма на рисунке 7.7 описывает рабочие характеристики таймера после его запуска, и когда он сброшен. Описание применимо, если при программировании с использованием отдельных элементов вы соблюдаете порядок, показанный в параграфе 7.1.6 «Последовательность операций с таймерами».

- ❶❷ Когда сигнальное состояние на входе запуска таймера меняется с «0» на «1» (положительный фронт), таймер запускается. Он работает в течение запрограммированной длительности, даже когда сигнальное состояние на входе запуска переходит обратно в «0». Считывание с ожиданием сигнального состояния «1» (состояние таймера) возвращает результат проверки «1», пока таймер работает.

Имея стартовое значение в качестве начальной точки, значение времени отсчитывается в порядке уменьшения до нуля согласно временной базе.

- ❸ Если сигнальное состояние на входе запуска меняется с «0» на «1» (положительный фронт) во время работы таймера, он перезапускается с запрограммированным значением времени (таймер «повторно запускается»). Таймер может быть перезапущен любое количество раз, не доходя до завершения заданного времени.

Сброс расширенного импульсного таймера

Сброс расширенного импульсного таймера имеет статический эффект и получает приоритет перед запуском таймера (рисунок 7.7).

- ❹ Сигнальное состояние «1» на входе запуска работающего таймера сбрасывает его. Считывание с ожиданием сигнального состояния «1» (состояние таймера) возвращает результат проверки «0» для обнуления таймера. Значение времени и временная база также обнуляются.
- ❺ «1» на входе сброса неактивного таймера влияния не оказывает.
- ❻❼ Если сигнальное состояние на входе запуска переходит из «0» в «1» (положительный фронт) при наличии сигнала сброса, то таймер запускается, но немедленно сбрасывается (на диаграмме показано линией). Если проверка состояния таймера запрограммирована после сброса, то кратковременный запуск таймера не повлияет на проверку таймера.

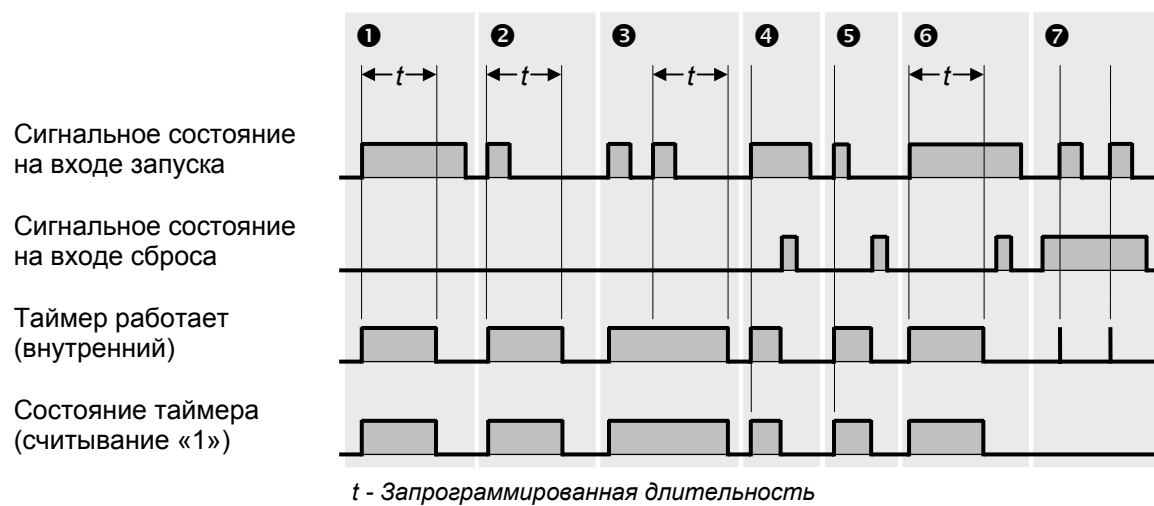


Рисунок 7.7 Рабочие характеристики при запуске и сбросе расширенного импульсного таймера

7.4 Таймер задержки включения

Запуск таймера задержки включения

На диаграмме, представленной на рисунке 7.8, описываются рабочие характеристики таймера после его запуска, и когда он сброшен. Описание применимо, если при программировании с использованием отдельных элементов вы соблюдаете порядок, показанный в параграфе 7.1.6 «Последовательность операций с таймерами».

- ① Когда сигнальное состояние на входе запуска таймера изменяется с «0» на «1» (положительный фронт), таймер запускается. Он работает в течение запрограммированной длительности. Операции считывания с ожиданием сигнального состояния «1» возвращают результат «1», если время должным образом истекло, и на входе запуска по-прежнему присутствует сигнальное состояние «1» (задержка включения).

Имея стартовое значение в качестве начальной точки, значение времени отсчитывается в направлении уменьшения до нуля согласно временной базе.

- ② Если во время работы таймера сигнальное состояние на входе запуска изменяется с «1» на «0», то таймер останавливается. В таких случаях считывание с ожиданием сигнального состояния «1» (состояние таймера) всегда возвращает результат проверки «1». Значение времени показывает количество оставшегося времени.

Обнуление таймера задержки включения

Сброс таймера задержки включения имеет статический эффект и получает приоритет перед запуском таймера (рисунок 7.8).

- ③④ Сигнальное состояние «1» на входе сброса обнуляет работающий и недействующий таймер. Считывание с ожиданием сигнального состояния «1» (состояние таймера) в этом случае возвращает результат проверки «0», даже если таймер не работает, и на входе запуска все еще присутствует сигнальное состояние «1». Значение времени и временная база также обнуляются.

Смена сигнального состояния на входе сброса с «1» на «0» при сигнальном состоянии «1» на входе запуска влияния на таймер не оказывает.

- ⑤ Если сигнальное состояние на входе запуска переходит из «0» в «1» (положительный фронт) при наличии сигнала сброса, то таймер запускается, но немедленно сбрасывается (на диаграмме это показано линией). Если проверка состояния таймера при программировании предусмотрена после сброса, то кратковременный запуск таймера не повлияет на проверку.

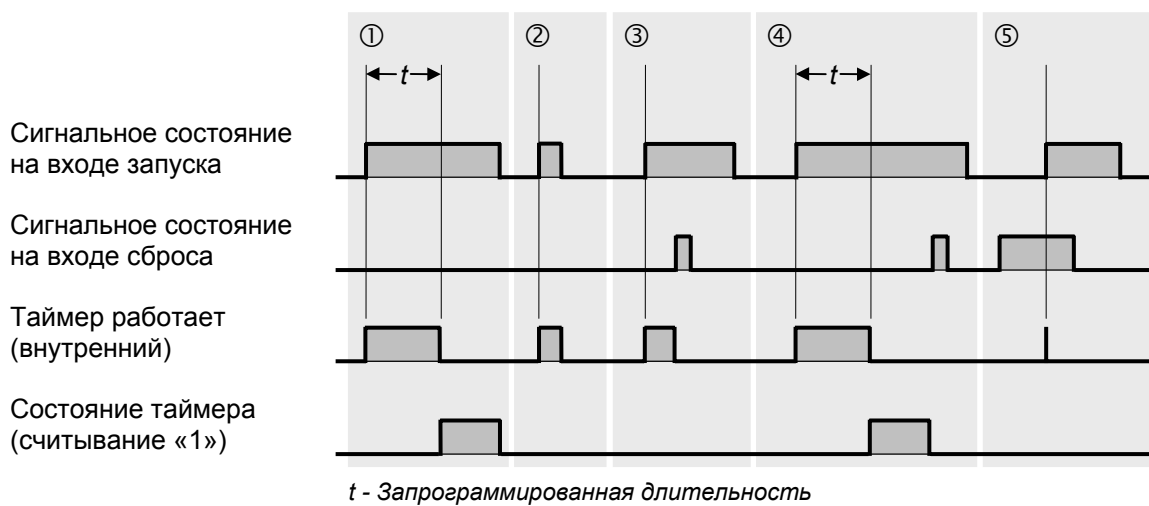


Рисунок 7.8 Рабочие характеристики при запуске и сбросе таймера задержки включения

7.5 Таймер задержки включения с запоминанием

Запуск таймера задержки включения с запоминанием

На диаграмме, изображенной на рисунке 7.9, описываются рабочие характеристики таймера после его запуска, и когда он сброшен. Описание применимо, если при программировании с использованием отдельных элементов вы соблюдаете порядок, показанный в параграфе 7.1.6 «Последовательность операций с таймерами».

- ❶❷ Когда сигнальное состояние на входе запуска таймера переходит из «0» в «1» (положительный фронт), таймер запускается. Он работает в течение запрограммированной длительности, даже когда сигнальное состояние на входе запуска возвращается к «0». По истечении времени считывание с ожиданием сигнального состояния «1» (состояние таймера) возвращает результат считывания «1» вне зависимости от сигнального состояния на входе запуска. Результат считывания «0» не возвращается до сброса таймера независимо от сигнального состояния на входе запуска.

Используя стартовое значение в качестве начальной точки, значение времени отсчитывается в направлении уменьшения согласно временной базе.

- ❸ Если сигнальное состояние на входе запуска меняется с «0» на «1» (положительный фронт) во время работы таймера, то таймер перезапускается с запрограммированным значением времени (таймер «повторно вызывается»). Он может быть перезапущен любое количество раз, при этом истечение заданного времени не обязательно.

Сброс таймера задержки включения с запоминанием

Обнуление таймера задержки включения с запоминанием имеет статический эффект и получает приоритет перед запуском таймера (рисунок 7.9).

- ❹❺ Сигнальное состояние «1» на входе сброса обнуляет таймер независимо от сигнального состояния на входе запуска. Считывание с ожиданием сигнального состояния «1» (состояние таймера) возвращает результат проверки «0». Значение времени и временная база также устанавливаются в нуль.
- ❻ Если сигнальное состояние на входе запуска переходит из «0» в «1» (положительный фронт) при наличии сигнала сброса, то таймер стартует, но немедленно сбрасывается (на диаграмме это отмечено линией). Если проверка состояния таймера запрограммирована после сброса, кратковременный запуск таймера на проверку не повлияет.

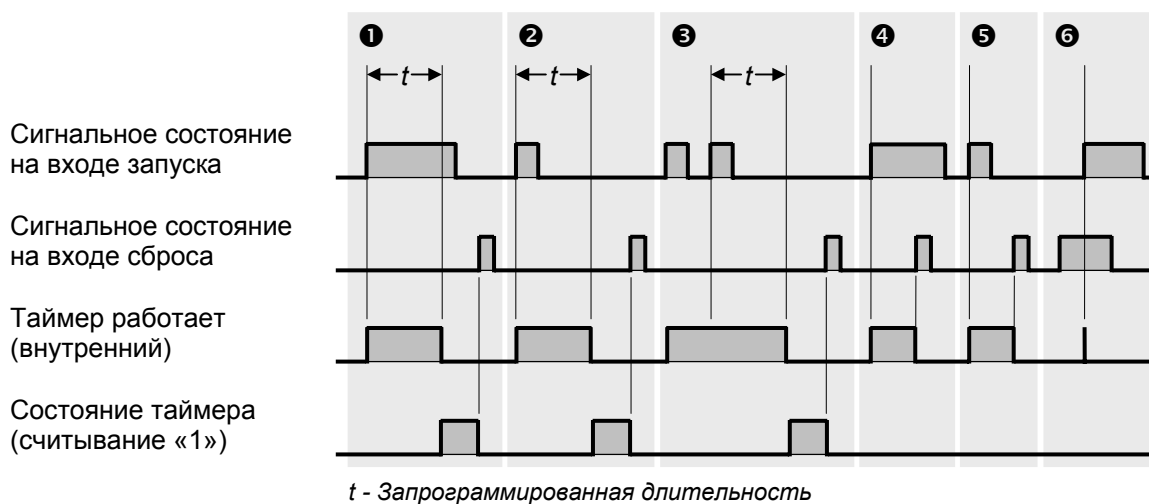


Рисунок 7.9 Рабочие характеристики при запуске и сбросе таймера задержки включения с запоминанием

7.6 Таймер задержки выключения

Запуск таймера задержки выключения

Диаграмма, представленная на рисунке 7.10, описывает рабочие характеристики таймера после его запуска, и когда он сброшен. Описание применимо, если при программировании с использованием отдельных элементов вы соблюдаете порядок, показанный в параграфе 7.1.6 «Последовательность операций с таймерами».

- ①③ Таймер запускается, когда сигнальное состояние на входе запуска таймера изменяется с «1» на «0» (отрицательный фронт). Он функционирует в течение запрограммированной длительности. Считывания с ожиданием сигнального состояния «1» (состояние таймера) возвращают результат проверки «1», когда сигнальное состояние на входе запуска «1», или когда таймер работает (задержка выключения).

Используя стартовое значение в качестве начальной точки, значение времени отсчитывается в направлении уменьшения согласно временной базе.

- ② Если сигнальное состояние на входе запуска меняется с «0» на «1» (положительный фронт) во время работы таймера, то он сбрасывается. Перезапускается таймер только при наличии на входе запуска отрицательного фронта.

Сброс таймера задержки выключения

Обнуление таймера задержки выключения имеет статический эффект и получает приоритет перед запуском таймера (рисунок 7.10).

- ④ Сигнальное состояние «1» на входе сброса таймера во время работы таймера обнуляет его. В этом случае результат считывания с ожиданием сигнального состояния «1» (состояние таймера) равен «0». Значение времени и временная база также обнулены.
- ⑤⑥ Сигнальное состояние «1» на входе запуска и на входе сброса обнуляет бинарный выход таймера. Считывание с ожиданием сигнального состояния «1» (состояние таймера) в этом случае возвращает сигнальное состояние «0». Если сигнальное состояние на входе сброса теперь снова переходит в «0», выход таймера возвращается в состояние «1».
- ⑦ Если сигнальное состояние на входе запуска переходит из «1» в «0» (отрицательный фронт) при наличии сигнала сброса, то таймер стартует, но немедленно сбрасывается (на диаграмме это показано линией). Тогда считывание с ожиданием сигнального состояния «1» (состояние таймера) возвращает результат проверки «0».

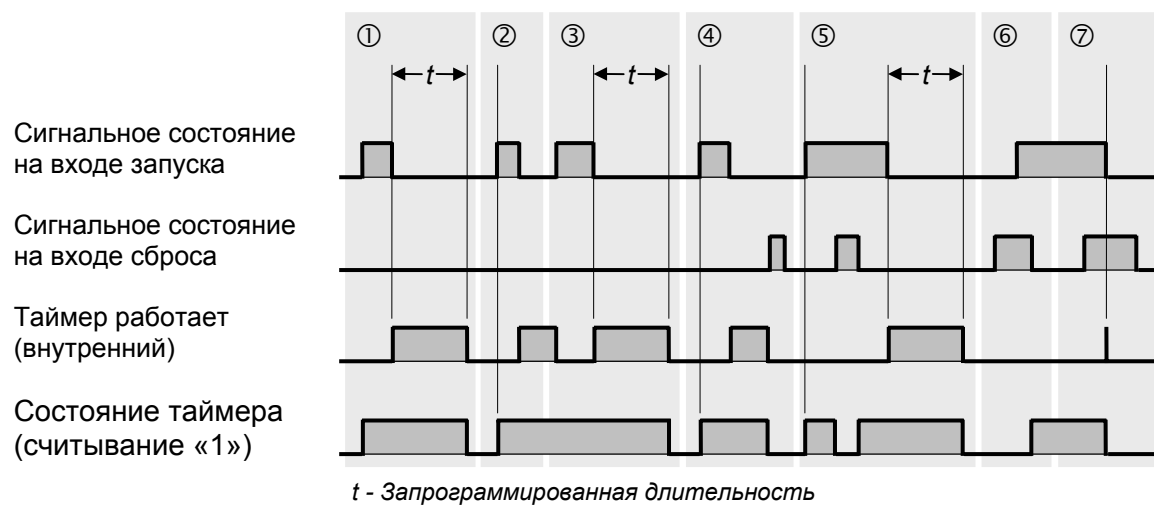


Рисунок 7.10 Рабочие характеристики при запуске и сбросе таймера задержки выключения

7.7 IEC-таймеры

IEC-таймеры встроены в операционную систему CPU как системные функциональные блоки (блоки SFB).

В некоторых CPU доступны следующие таймеры:

- SFB 3 TP
Импульсный таймер (pulse timer);
- SFB 4 TON
Таймер задержки включения (on-delay timer);
- SFB 5 TOF
Таймер задержки выключения (off-delay timer).

Рисунок 7.11 показывает рабочие характеристики этих таймеров.

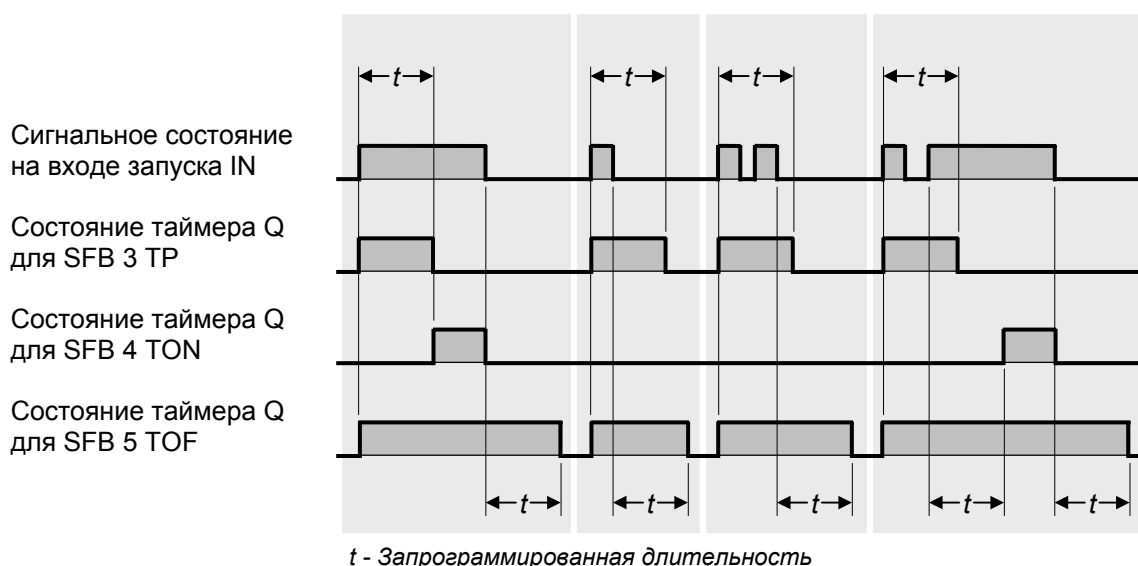


Рисунок 7.11 Рабочие характеристики IEC-таймеров

Эти блоки SFB вызываются с экземплярным блоком данных или используются как локальные экземпляры в функциональном блоке. Описание интерфейса для автономного программирования вы найдете в стандартной библиотеке под именем *Standard Library* в программе *System Function Blocks* (Системные функциональные блоки).

Примеры вызовов вы сможете найти на прилагаемой к книге дискете в библиотеках «LAD_Book» и «FBD_Book», функциональный блок FB 107 в программе «Basic Functions» («Основные функции»).

7.7.1 Импульсный таймер SFB 3 TP

Параметры IEC-таймера SFB 3 TP приведены в таблице 7.1.

Таблица 7.1 Параметры IEC-таймеров

Имя	Объявление	Тип данных	Описание
IN	INPUT	BOOL	Вход запуска
PT	INPUT	TIME	Длина импульса или длительность задержки
Q	OUTPUT	BOOL	Состояние таймера
ET	OUTPUT	TIME	Истекшее время (сколько времени прошло)

Когда RLO на входе запуска таймера переходит из «0» в «1», таймер запускается. Он работает в течение запрограммированной длительности вне зависимости от любых последующих изменений в RLO на входе запуска. Сигнальное состояние на выходе Q работающего таймера равно «1».

Выход ET выдает длительность времени для выхода Q. Эта длительность начинается с T#0s и заканчивается на установленной длительности PT. Когда PT истекает, ET остается установленным в значение истекшего времени до перехода входа IN обратно в «0». Если вход IN переходит в «0» до истечения PT, то выход ET переходит в T#0s, текущая длительность PT обнуляется.

Чтобы повторно инициализировать таймер, просто запустите его с PT = T#0s.

Таймер SFB 3 TP активен в режимах CPU START и RUN. Он сбрасывается (инициализируется) при выполнении «холодного» рестарта.

7.7.2 Таймер задержки включения SFB 4 TON

Параметры IEC-таймера SFB 4 TON показаны в таблице 7.1.

Таймер запускается, когда RLO на входе запуска таймера меняется с «0» на «1». Он работает в течение запрограммированной длительности. Выход Q показывает сигнальное состояние «1», когда время истекло, и пока на входе запуска остается сигнальное состояние «1». Если RLO на входе запуска таймера меняется с «1» на «0» до окончания заданного времени, то таймер сбрасывается. Следующий положительный фронт вновь запускает таймер.

Выход ET выдает длительность времени для таймера. Эта длительность начинается с T#0s и заканчивается на установленной длительности PT. Когда PT заканчивается, ET остается установленным в значение истекшего времени до перехода входа IN в «0». Если вход IN обнуляется до истечения PT, то ET немедленно переходит в T#0s.

Чтобы повторно инициализировать таймер, просто запустите его с PT = T#0s.

Таймер SFB 4 TON активен в режимах CPU START и RUN. Он сбрасывается при «холодном» рестарте.

7.7.3 Таймер задержки выключения SFB 5 TOF

Параметры IEC-таймера SFB 5 TOF перечислены в таблице 7.1.

На выходе Q присутствует сигнальное состояние «1», когда RLO на входе запуска таймера меняется с «0» на «1». Таймер запускается, когда RLO на входе запуска вновь обнуляется. Пока таймер работает, выход Q сохраняет сигнальное состояние «1».

Выход Q сбрасывается, когда время истекает. Если RLO на входе запуска переходит опять в «1» до истечения времени, то таймер сбрасывается, и выход Q остается установленным в «1».

Выход ET выдает длительность времени для таймера. Эта длительность начинается с T#0s и заканчивается на установленной длительности PT. Когда время, заданное на входе PT, заканчивается, ET остается установленным в значение истекшего времени до перехода входа IN в «1». Если вход IN устанавливается в «1» до истечения PT, то ET немедленно переходит в T#0s.

Чтобы повторно инициализировать таймер, просто запустите его с PT = T#0s.

Таймер SFB 5 TOF активен в режимах CPU START и RUN. Он сбрасывается при «холодном» рестарте.

Содержание главы 8

8	<u>Счетчики</u>	4
8.1	<u>Программирование счетчика</u>	4
8.2	<u>Установка и сброс счетчиков</u>	9
8.3	<u>Вычисление</u>	10
8.4	<u>Опрос счетчика</u>	11
8.5	<u>IES-счетчики</u>	12
8.5.1	<u>Счетчик прямого счета SFB 0 CTU</u>	12
8.5.2	<u>Счетчик обратного счета SFB 1 CTD</u>	13
8.5.3	<u>Счетчик прямого/обратного счета SFB 2 CTUD</u>	13
8.6	<u>Пример счетчика деталей</u>	15

8 Счетчики

Счетчики (counters) позволяют использовать CPU в вычислительных задачах. Счетчики могут вести счет по возрастанию (прямой счет) и по убыванию (обратный счет). Область счета охватывает три разряда (от 000 до 999). Счетчики располагаются в системной памяти CPU; количество счетчиков определяется версией CPU.

Вы можете запрограммировать счетчик полностью в виде блочного элемента или с использованием отдельных программных элементов. Вы можете задать для счетчика определенное начальное значение или сбросить его, вести прямой и обратный счет. Счетчик сканируется путем считывания его состояния (нулевое или ненулевое значение счета) или текущего значения счетчика (значения счета, count value), которое вы можете получить либо в двоичном виде, либо в двоично-десятичном коде.

8.1 Программирование счетчика

Со счетчиками производятся следующие операции:

- Установка счетчика, задание значения счетчика;
- Прямой счет;
- Обратный счет;
- Сброс счетчика;
- Считывание состояния (двоичного) счетчика;
- Считывание (числового) значения счетчика в двоичной форме;
- Считывание (числового) значения счетчика в двоично-десятичной форме.

Представление счетчика в виде блочного элемента

Блочный элемент счетчика содержит все операции счета в форме функциональных входов и функциональных выходов (рисунок 8.1). Над блочным элементом расположен абсолютный или символический адрес счетчика. В блочном элементе в качестве заголовка указывается тип счетчика (S_CUD означает «up-down counter», «счетчик прямого и обратного счета»). Назначение для первого входа (на примере CU) обязательно; для остальных входов и выходов назначения можно не делать.

Имеется три исполнения блочных элементов счетчика: счетчик прямого и обратного счета (up-down counter, S_CUD), счетчик только прямого счета (up counter, S_CU) и счетчик обратного счета (down counter, S_CD). Функциональные различия объясняются ниже.

Блочный элемент счетчика

(в примере: счетчик прямого и обратного счета)



Наименование	Тип данных	Описание
CU	BOOL	Вход прямого счета
CD	BOOL	Вход обратного счета
S	BOOL	Вход установки
PV	WORD	Вход предустановки
R	BOOL	Вход сброса
CV	WORD	Текущее значение в двоичном виде
CV_BCD	WORD	Текущее значение в двоично-десятичном виде
Q	BOOL	Состояние счетчика

Рисунок 8.1 Счетчик в виде блочного элемента (counter box)

Для создания программ с помощью пошагового программирования элементы-счетчики вы можете найти в каталоге программных элементов (Program Element Catalog), вызываемого командой меню View → Catalog (Вид → Каталог) [Ctrl + K] или Insert → Program Elements (Вставка → Программные элементы), в разделе «Counters» («Счетчики»).

Представление счетчика с использованием отдельных элементов (LAD)

Запрограммировать счетчик вы также можете, используя отдельные элементы (рисунок 8.2).

Установка и счет в этом случае производятся посредством катушек. Катушка установки счетчика содержит операцию вычисления (подсчета) (SC = Set Counter, установка счетчика); под катушкой находится значение счетчика в формате WORD, которое используется для установки счетчика.

В катушках для вычисления CU (count up) означает прямой счет, CD (count down) – обратный счет. Чтобы сбросить счетчик, используйте катушку для сброса, а NO- или NC-контакт – для проверки состояния счетчика.

Наконец, вы можете переслать текущее значение счетчика в двоичном виде с помощью блочного элемента MOVE.

Счетчик прямого счета (катушка прямого счета)	Операнд счетчика —(CU)—
Сброс обратного счета (катушка обратного счета)	Операнд счетчика —(CD)—
Установка счетчика, определение значения счетчика (катушка установки счетчика со значением счета)	Операнд счетчика —(SC)— Значение счета
Сброс счетчика (катушка сброса)	Операнд счетчика —(R)—
Проверка состояния счетчика (NO-контакт, NC-контакт)	Операнд счетчика — — Операнд счетчика — / —
Чтение значения счета в двоичной форме (блочный элемент MOVE)	

Рисунок 8.2 Отдельные элементы счетчика (LAD)

Блочный элемент счетчика в цепи (LAD)

Перед входами счетчика, входами запуска и сброса, а также после выхода Q вы можете сгруппировать контакты последовательно и параллельно.

Блочный элемент счетчика может быть помещен после Т-ветви или в участке цепи, которая непосредственно соединена левой направляющей (шиной питания). Данная ветвь также может иметь контакты перед входами и не должна быть самой верхней.

Дополнительные примеры представления и компоновки счетчиков вы можете найти на дискете, прилагаемой к книге, в библиотеке «LAD_Book», функциональном блоке FB 108 программы «Basic Functions» («Основные функции»).

Представление счетчика с использованием отдельных элементов (FBD)

Запрограммировать счетчик вы также можете, используя отдельные элементы (рисунок 8.3).

Установка и подсчет в этом случае производятся посредством простых блочных элементов. Блочный элемент установки счетчика содержит операцию вычисления (SC = Set Counter, установка счетчика); на входе PV находится значение счетчика в формате WORD, которое используется для установки счетчика.

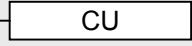

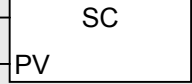
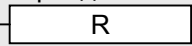


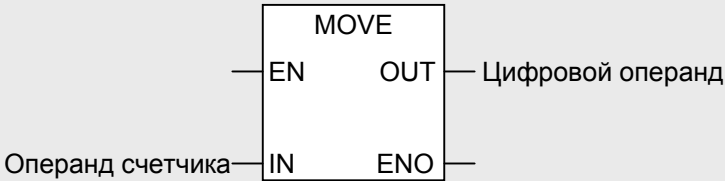
Счетчик прямого счета (блочный элемент прямого счета)	Операнд счетчика — 
Счетчик обратного счета (блочный элемент обратного счета)	Операнд счетчика — 
Установка счетчика, определение значения счетчика (блочный элемент установки счетчика со значением счета)	Операнд счетчика —  Значение счета — PV
Сброс счетчика (блочный элемент сброса)	Операнд счетчика — 
Проверка состояния счетчика (прямой или инвертированный двоичный вход)	Операнд счетчика —  Операнд счетчика — 
Чтение значения счета в двоичной форме (блочный элемент MOVE)	 Операнд счетчика — IN ENO — EN OUT — Цифровой операнд

Рисунок 8.3 Отдельные элементы счетчика (FBD)

В блочных элементах для вычислений CU (count up) означает прямой счет, CD (count down) – обратный счет. Чтобы сбросить счетчик, используйте блочный элемент для сброса, а прямой или инвертированный двоичный вход функции – для проверки состояния счетчика.

Наконец, вы можете переслать текущее значение счетчика в двоичном виде с помощью блочного элемента MOVE.

Блочный элемент счетчика в логической цепи (FBD)

Перед входами счетчика, входами запуска и сброса, а также после выхода Q вы можете поместить бинарные функции и функции для работы с памятью.

Блочный элемент счетчика и отдельные элементы для счета, установки и сброса счетчика также могут быть размещены после T-ветви.

Дополнительные примеры представления и компоновки счетчиков вы можете найти на дискете, прилагаемой к книге, в библиотеке «FBD_Book», функциональном блоке FB 108 программы «Basic Functions» («Основные функции»).

Последовательность операций вычисления

При программировании счетчика вы можете не использовать все доступные для него операции. Достаточно операций, требующихся для выполнения нужных функций.

Например, чтобы запрограммировать счетчик обратного счета, вам потребуются только операции для установки счетчика в начальное значение, обратного счета и проверки состояния счетчика.

Для того, чтобы рабочие характеристики счетчика соответствовали описанию этой главы, рекомендуется соблюдать следующий порядок при программировании с использованием отдельных программных элементов:

- Счет (прямой или обратный в любом порядке);
- Установка счетчика;
- Сброс счетчика;
- Проверка счета;
- Проверка состояния счетчика.

Любые ненужные отдельные элементы пропускайте. Если подсчет, установка и сброс счетчика происходят «одновременно» при программировании операций в указанном порядке, то счет будет сначала соответственно изменен, затем обнулен следующей операцией сброса. Последующая проверка, таким образом, покажет нулевое значение счетчика и состояние счетчика «0».

Если вычисление и установка происходят «одновременно» при программировании операций в указанном порядке, то счет будет сначала соответственно изменен, затем установлен в запрограммированное значение счета, которое сохранится оставшуюся часть цикла.

Порядок операций счета по возрастанию и убыванию не важен.

8.2 Установка и сброс счетчиков

Установка счетчиков

Счетчик устанавливается, когда RLO меняется с «0» на «1» на входе установки S или перед катушкой установки или блочным элементом установки. Для установки счетчика всегда требуется положительный фронт. «Установка счетчика» («Set counter») означает, что счетчик устанавливается в начальное значение. Область значения – от 0 до 999.

Определение значения счета счетчика

Когда счетчик устанавливается, он принимает в качестве значения счета значение на входе PV или значение под катушкой установки. Вы можете определить значение счетчика как константы, операнда длиной в слово или переменной типа WORD.

Определение значения счета как константы

S#100	Значение счета 100
W#16#0100	Значение счета 100

Значение счета содержит три разряда из области 000 ... 999. Допустимы только положительные BCD-значения; счетчик не обрабатывает отрицательные значения. Для идентификации константы вы можете использовать S# или W#16# (только вместе с десятичными числами).

Определение значения счета как операнда или переменной

MW 56	Операнд размером в слово, содержащий значение счета
“Count value”	Переменная типа WORD.

Сброс счетчиков (LAD)

Счетчик сбрасывается, когда ток протекает на входе сброса или в катушке сброса (когда присутствует RLO «1»). В этом случае проверка счетчика NO-контактом вернет результат считывания «0», а проверка NC-контактом возвратит результат считывания «1». Сброс счетчика устанавливает его значение счета в «нуль». Вход R блочного элемента счетчика может быть свободным (неподключенным).

Сброс счетчиков (FBD)

Счетчик сбрасывается, когда «1» присутствует на входе сброса. Прямое сканирование состояния счетчика вернет «0», а инвертированное сканирование возвратит «1». Сброс счетчика устанавливает его значение счета в «нуль». Вход R блочного элемента счетчика может быть свободным (неподключенным).

8.3 Вычисление

Частота счета счетчика определяется временем выполнения вашей программы! Для того, чтобы вести счет, CPU должен обнаружить изменение в состоянии входного импульса, то есть входной импульс (или его отсутствие) должен присутствовать в течение, по крайней мере, одного программного цикла. Таким образом, чем дольше время исполнения программы, тем меньше частота ведения счета.

Счет по возрастанию

Счетчик приступает к прямому счету, когда RLO меняется с «0» на «1» на входе прямого счета CU (up count input) или перед катушкой прямого счета, или блочным элементом. Для прямого счета всегда требуется положительный фронт.

При прямом счете каждый положительный фронт увеличивает счет на единицу до достижения верхней границы интервала 999. Каждый следующий положительный фронт влияния на ведение прямого счета не оказывает. Переноса не происходит.

Счет по убыванию

Счетчик начинает вести счет по убыванию при смене RLO с «0» на «1» на входе обратного счета CD (down count input) или перед катушкой обратного счета, или блочным элементом. Для обратного счета всегда требуется положительный фронт.

При обратном счете каждый положительный фронт уменьшает счет на единицу до достижения нижней границы интервала 0. Каждый последующий положительный фронт для ведения обратного счета не используется. Отрицательных значений счета не возникает.

Различные блочные элементы счетчика

Редактор предоставляет три различных блочных элемента счетчика:

- S_CUD Счетчик прямого/обратного счета;
- S_CU Счетчик прямого счета;
- S_CD Счетчик обратного счета.

Эти блочные элементы счетчика отличаются только типом и количеством входов счетчика. S_CUD имеет входы для обоих направлений вычисления, а S_CU оснащен только входом прямого счета, и S_CD имеет только вход обратного счета.

Вы всегда должны подключать первый вход блочного элемента счетчика. Если вы не подключаете второй вход (S_CD) блочного элемента S_CUD, то этот элемент приобретает характеристики счетчика S_CU.

8.4 Опрос счетчика

Проверка состояния счетчика (LAD)

Состояние счетчика подается на выход Q блочного элемента счетчика. Состояние счетчика также можно проверить с использованием NO-контакта (соответствует выходу Q) или NC-контакта.

Выход Q содержит «1» (ток течет из выхода), когда текущее значение счета больше нуля. На выходе Q содержится «0», если текущее значение счета равно нулю. Выход Q в блочном элементе счетчика может быть не подключен.

Проверка состояния счетчика (FBD)

Состояние счетчика подается на выход Q блочного элемента счетчика. Состояние счетчика также можно опросить непосредственно (соответствует выходу Q) с помощью двоичного входа функции или в инвертированном виде.

Выход Q содержит «1», когда текущее значение счета больше нуля. На выходе Q содержится «0», если текущее значение счета равно нулю. Выход Q блочного элемента счетчика может быть не подключен.

Считывание значения счета (LAD и FBD)

Выходы CV и CV_BCD предоставляют текущее значение счета счетчика в двоичной форме (CV) или в двоично-десятичном коде (BCD). Значение счета, полученное при данной операции, является текущим в момент считывания.

Значение сохраняется в определенном операнде (пересылается так же, как при использовании блочного элемента MOVE). Эти выходы в блочном элементе счетчика можно не подключать (оставить свободными).

Прямое считывание значения счета

Значение счета доступно в двоичном виде и может быть получено из счетчика в этой форме. Значение соответствует положительному числу в формате INT. Прямое считывание значения счета может быть также запрограммировано с использованием блочного элемента MOVE.

Кодированное считывание значения счета

Вы также можете получить бинарное значение счета из счетчика в «кодированной» форме. Двоично-десятичное (BCD) значение структурировано так же как и при определении значения счета (см. выше).

8.5 ИЕС-счетчики

ИЕС-счетчики встроены в операционную систему CPU в качестве системных функциональных блоков (блоки SFB). В соответствующих CPU имеются следующие счетчики:

- SFB 0 STU
Счетчик прямого счета;
- SFB 1 CTD
Счетчик обратного счета;
- SFB 2 CTUD
Счетчик прямого/обратного счета.

Вы можете вызывать эти блоки SFB с экземплярным блоком данных или использовать их как локальные экземпляры в другом функциональном блоке.

Описание интерфейса для автономного программирования вы найдете в стандартной библиотеке под именем *Standard Library* в программе *System Function Blocks* (Системные функциональные блоки).

Примеры вызовов вы сможете найти на прилагаемой к книге дискете в библиотеках «LAD_Book» и «FBD_Book», функциональный блок FB 108 в программе «Basic Functions» («Основные функции»).

8.5.1 Счетчик прямого счета SFB 0 STU

ИЕС-счетчик SFB 0 STU имеет параметры, приведенные в таблице 8.1.

Таблица 8.1 Параметры ИЕС-счетчиков

Наименование	Наличие в SFB			Объявление	Тип данных	Описание
	0	-	2			
CU	0	-	2	INPUT	BOOL	Вход прямого счета
CD	-	1	2	INPUT	BOOL	Вход обратного счета
R	0	-	2	INPUT	BOOL	Вход сброса
LOAD	-	1	2	INPUT	BOOL	Вход загрузки
PV	0	1	2	INPUT	INT	Предустановленное значение
Q	0	1	-	OUTPUT	BOOL	Состояние счетчика
QU	-	-	2	OUTPUT	BOOL	Состояние счетчика прямого счета
QD	-	-	2	OUTPUT	BOOL	Состояние счетчика обратного счета
CV	0	1	2	OUTPUT	INT	Текущее значение счета

Когда сигнальное состояние на входе прямого счета CU меняется с «0» на «1» (положительный фронт), текущее значение счета увеличивается на 1 и подается на выход CV. При первом вызове (с сигнальным состоянием «0» на входе сброса R) счет соответствует значению по умолчанию на входе PV. Когда значение счета достигает верхней границы 32767, оно больше не инкрементируется, и вход CU воздействия не оказывает.

Значение счета обнуляется, когда сигнальное состояние на входе сброса R равно «1». Пока вход R содержит «1», положительный фронт на CU влияния не оказывает. Когда значение на CV больше или равно значению на PV, выход Q установлен в «1».

SFB 0 STU выполняется в режимах CPU START и RUN. Он сбрасывается при «холодном» перезапуске.

8.5.2 Счетчик обратного счета SFB 1 CTD

ИЕС-счетчик SFB 1 CTD имеет параметры, указанные в таблице 8.1.

Когда сигнальное состояние на входе обратного счета CD переходит от «0» к «1» (положительный фронт), текущее значение счета уменьшается на 1 и подается на выход CV. При первом вызове (с сигнальным состоянием «0» на входе загрузки LOAD) счет соответствует значению по умолчанию на входе PV. Когда значение счета достигает нижней границы -32768 , оно больше не декрементируется, и CD воздействия не оказывает.

Когда на входе загрузки LOAD устанавливается «1», значение счета счетчика приобретает значение по умолчанию PV. Пока вход загрузки LOAD содержит «1», положительный фронт на CD влияния не оказывает.

Когда значение на CV меньше или равно нулю, выход Q установлен в «1».

SFB 1 CTD выполняется в режимах CPU START и RUN. Он сбрасывается при «холодном» рестарте.

8.5.3 Счетчик прямого/обратного счета SFB 2 CTUD

ИЕС-счетчик SFB 2 CTUD имеет параметры, данные в таблице 8.1.

Когда сигнальное состояние на входе прямого счета CU меняется с «0» на «1» (положительный фронт), текущее значение счета увеличивается на 1 и подается на выход CV. Когда сигнальное состояние на входе обратного счета CD переходит от «0» к «1» (положительный фронт), текущее значение счета уменьшается на 1 и подается на выход CV. Если оба входа показывают положительный фронт, то текущее значение счета не изменяется.

Если текущее значение счета достигает верхнего предела 32767, то оно более не инкрементируется при положительном фронте на входе прямого счета CU. В этом случае CU далее влияния не оказывает. Если текущее значение счета достигает нижней

границы -32768 , то оно больше не декрементируется при положительном фронте на входе обратного счета CD. И тогда CD воздействия не оказывает.

Когда вход загрузки LOAD устанавливается в «1», значение счета устанавливается в значение по умолчанию PV. Пока вход загрузки LOAD содержит «1», положительный фронт на двух счетных входах действия не оказывает.

Счетчик обнуляется, когда вход сброса R устанавливается в «1». Пока вход R содержит «1», положительный фронт сигнала на двух счетных входах и сигнальное состояние «1» на входе загрузки LOAD влияния не оказывают.

Выход QU устанавливается в «1», когда значение на CV больше или равно значению на PV.

Выход QD устанавливается в «1», когда значение на CV меньше или равно нулю.

SFB 2 CTUD работает в режимах CPU START и RUN. Он сбрасывается при «холодном» рестарте.

8.6 Пример счетчика деталей

Пример иллюстрирует использование таймеров и счетчиков. Он программируется с применением входов, выходов и меркеров, так что он может быть запрограммирован в любой точке любого блока. В данном случае используется функция без параметров; таймеры и счетчик представлены в виде законченных блоков.

Тот же пример, запрограммированный как функциональный блок с параметрами и с использованием отдельных элементов, вы найдете в главе 19 «Параметры блоков».

Описание действия

Детали перемещаются на транспортере конвейера. Световой барьер обнаруживает и считает детали. После установленного количества счетчик посылает сигнал «Finished» («Завершено»). Счетчик оборудован схемой наблюдения (мониторинга). Если сигнальное состояние светового барьера не меняется за определенный промежуток времени, то монитор (устройство наблюдения) посылает сигнал.

Вход «Set» («Установка») передает начальное значение (количество, которое необходимо подсчитать) в счетчик. Положительный фронт светового барьера уменьшает счетчик на единицу. По достижении нулевого значения счетчик посылает сигнал «Finished» («Завершено»). Необходимым условием является раздельное размещение деталей (через промежутки) на транспортере.

Вход «Set» («Установка») также устанавливает сигнал «Active» («Активный»). Контроллер отслеживает изменение сигнального состояния светового барьера только в активном состоянии. Когда подсчет завершается, и последнее сосчитанное изделие прошло световой барьер, «Active» («Активный») сбрасывается.

В активном состоянии положительный фронт светового барьера запускает таймер со значением времени «Duration1» («Durat1», «Длительность1») как импульсный таймер с запоминанием (retentive pulse timer). Если вход запуска таймера обрабатывается с «0» в следующем цикле, то он все еще продолжает работу. Новый положительный фронт вновь «включает» таймер, то есть перезапускает его. Следующий положительный фронт для рестарта таймера генерируется, когда световой барьер посылает сигнал отрицательного фронта. В этом случае таймер запускается со значением времени «Duration2» («Durat2», «Длительность2»). Если световой барьер в настоящий момент занят на период времени, превышающий «Durat1», или свободен на период времени, превышающий «Durat2», таймер останавливается и выдает сигнал «Fault» («Сбой»). При первой активации таймер запускается со значением времени «Durat2».

Сигналы, символы

Сигнал «Set» («Установка») активирует счетчик и монитор (систему наблюдения). Световой барьер управляет счетчиком, «активным» состоянием, выбором значения времени и началом (повторным запуском) времени наблюдения посредством положительных и отрицательных фронтов. Определение положительного и отрицательного фронта светового барьера требуется часто, и в качестве рабочей (временной, сверхоперативной) памяти подходят временные локальные данные. Временные локальные данные являются внутриблочными (локальными для блока) переменными. Они объявляются (описываются) в блоках (не в таблице символов).

В примере меркеры импульса, используемые для определения фронта, хранятся во временных локальных данных. (Меркерам фронта их сигнальные состояния требуются также в следующем цикле, и, следовательно, они не должны быть временными локальными данными).

Мы выбираем символическую адресацию, то есть операнды являются назначенными именами, которые затем используются для программирования. Перед вводом программы мы создаем таблицу символов (таблица 8.2), содержащую входы, выходы, меркеры, таймеры, счетчики и блоки.

Таблица 8.2 Таблица символов для примера счетчика деталей

Symbol (Символ)	Address (Адрес)	Data Type (Тип данных)	Comment (Комментарий)
Counter_control	FC 12	FC 12	Счетчик и управление монитором деталей
Acknowl	I 0.6	BOOL	Подтверждение сбоя
Set	I 0.7	BOOL	Установка счетчика, активация монитора
Lbarr1	I 1.0	BOOL	Сигнал датчика транспортера 1 конвейера «End_of_belt» («Конец транспортера»)
Finished	Q 4.2	BOOL	Набранное количество деталей
Fault	Q 4.3	BOOL	Отклик монитора
Active	M 3.0	BOOL	Счетчик и монитор активны
EM_LB_P	M 3.1	BOOL	Меркер фронта для положительного фронта светового барьера
EM_LB_N	M 3.2	BOOL	Меркер фронта для отрицательного фронта светового барьера
EM_Ac_P	M 3.3	BOOL	Меркер фронта для положительного фронта «Monitor active» («Монитор активен»)
EM_ST_P	M 3.4	BOOL	Меркер фронта для положительного фронта «Set» («Установка»)
Quantity	MW 4	WORD	Количество деталей
Durat1	MW 6	S5TIME	Время наблюдения для занятого светового барьера
Durat2	MW 8	S5TIME	Время наблюдения для незанятого светового барьера
Count	C 1	COUNTER	Функция счетчика для деталей
Monitor	T 1	TIMER	Функция таймера для монитора

Программа

Программа расположена в функции, которую вы вызываете в CPU в организационном блоке OB 1 (выбирается из каталога программных элементов, раздела «FC Blocks», «Блоки FC»). При программировании также могут быть использованы глобальные символы без кавычек, не содержащих тем самым специальных литер. Если символ включает специальную литеру (умлаут или пробел, например), то он должен быть заключен в кавычки. В компилированном блоке редактор показывает все глобальные символы в кавычках.

На рисунках 8.4 и 8.5 показана программа для счетчика деталей. Эту программу вы можете найти на дискете, прилагаемой к книге, в библиотеках «LAD_Book» и «FBD_Book» в функции FC 12 в разделе «Conveyor Example» («Пример конвейера»).

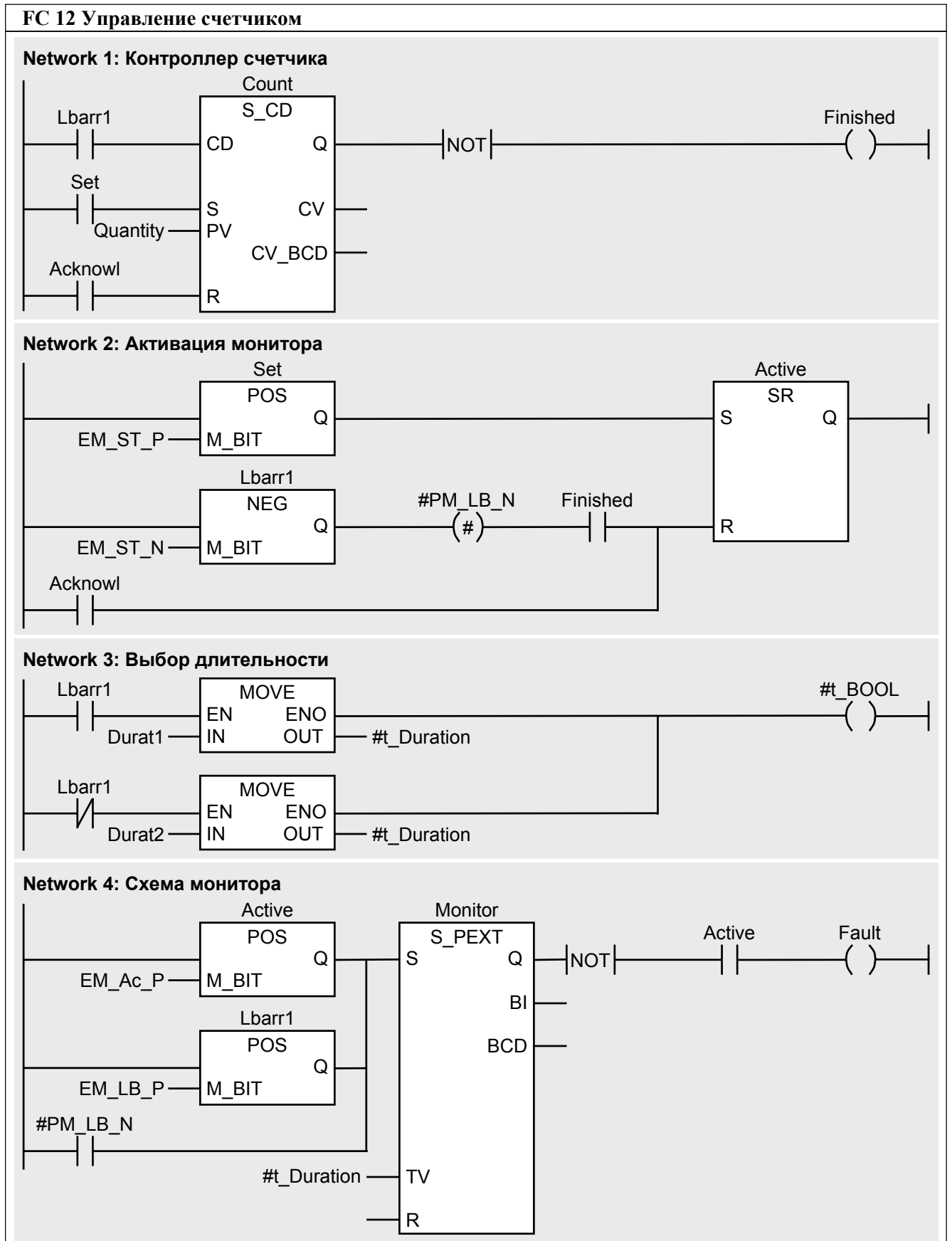


Рисунок 8.4 Программирование примера для счетчика деталей (LAD)

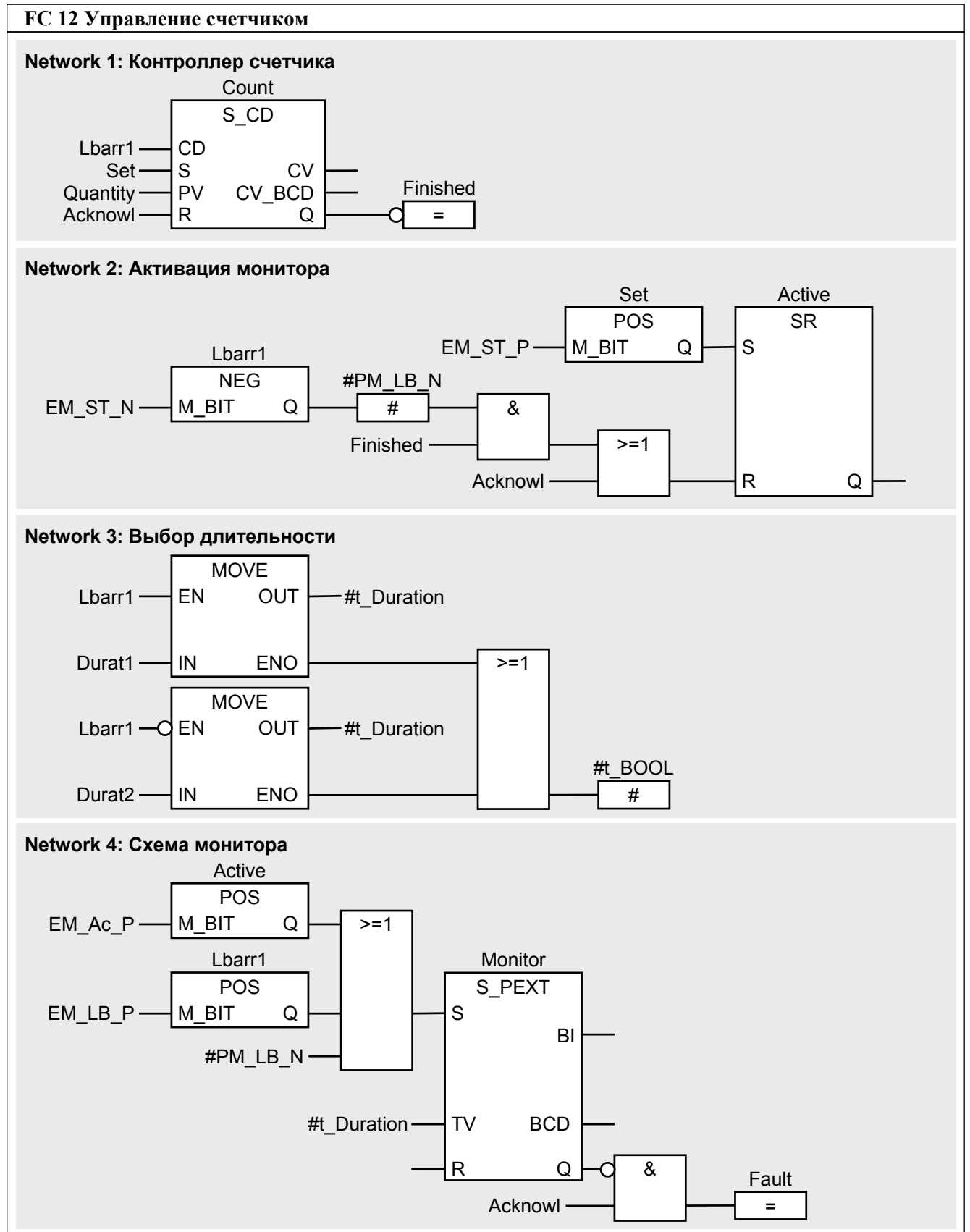


Рисунок 8.5 Программирование примера для счетчика деталей (FBD)

Числовые функции

Числовые функции обрабатывают числовые значения преимущественно типов данных INT (целое), DINT (двойное целое) и REAL (вещественное) и, таким образом, расширяют функциональность PLC.

Функции сравнения формируют бинарный результат сравнения двух значений. Они работают с типами данных INT, DINT и REAL.

Арифметические функции используются для осуществления вычислений в вашей программе. Все основные арифметические действия производятся над типами данных INT, DINT и REAL.

Математические функции расширяют вычислительные возможности за основные арифметические функции, включая, например, тригонометрические функции.

До и после проведения вычислений вы можете привести числовые значения к требуемому типу данных, используя **функции преобразования**.

Функции сдвига позволяют выравнивать содержимое переменной путем сдвига вправо или влево.

С помощью **битовых логических операций (числовой логики)** вы можете маскировать цифровые значения, выделяя (указывая) отдельные биты и устанавливая их в «1» или «0».

Операции числовой логики работают главным образом со значениями, хранящимися в блоках данных. Это могут быть блоки глобальных данных или экземплярные блоки данных, если используются статические локальные данные. Параграф 18.2 «Функции для блоков данных» демонстрирует управление блоками данных и представляет методы адресации операндов данных.

9 Функции сравнения

Операции сравнения: равно, неравно, больше, больше или равно, меньше и меньше или равно

10 Арифметические функции

Основные арифметические функции, работающие с типами данных INT, DINT и REAL

11 Математические функции

Тригонометрические функции; обратные тригонометрические функции; возведение в квадрат; извлечение квадратного корня; возведение в степень и логарифмы

12 Функции преобразования

Преобразование из INT/DINT в BCD и обратно; преобразование из DINT в REAL и обратно с различными формами округления; (поразрядное) дополнение до единицы; инвертирование (отрицание) и генерирование абсолютного значения

13 Функции сдвига

Сдвиг вправо и влево, по слову и двойному слову, сдвиг с корректным знаком; ротация (циклический сдвиг) вправо и влево

14 Побитовые логические операции

AND, OR и исключающее OR; комбинирование слов и двойных слов

Содержание главы 9

<u>9</u>	<u>Функции сравнения</u>	3
<u>9.1</u>	<u>Обработка функции сравнения</u>	3
<u>9.2</u>	<u>Описание функций сравнения</u>	6

9 Функции сравнения

Функции сравнения сравнивают две числовых переменных, относящихся к типам данных INT, DINT и REAL, на предмет равенства, неравенства, больше, больше или равно, меньше, меньше или равно. После операции сравнения выдается ее результат в виде двоичного значения (таблица 9.1).

Таблица 9.1 Обзор функций сравнения

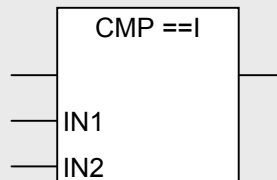
Функция сравнения	Сравнение в соответствии с типом данных		
	INT	DINT	REAL
Сравнение «равно»	CMP ==I	CMP ==D	CMP ==R
Сравнение «не равно»	CMP <>I	CMP <>D	CMP <>R
Сравнение «больше»	CMP >I	CMP >D	CMP >R
Сравнение «больше или равно»	CMP >=I	CMP >=D	CMP >=R
Сравнение «меньше»	CMP <I	CMP <D	CMP <R
Сравнение «меньше или равно»	CMP <=I	CMP <=D	CMP <=R

9.1 Обработка функции сравнения

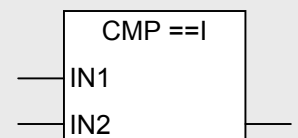
Блочный элемент сравнения

(в примере: операция сравнения «равно» между числами INT)

Представление LAD:



Представление FBD:



Представление (LAD)

Кроме (немаркированного) двоичного входа блочный элемент функции сравнения имеет два входа IN1 и IN2 и (немаркированный) двоичный выход. «Заголовок» в блочном элементе идентифицирует операцию сравнения (CMP означает compare, сравнение) и тип выполняемого сравнения (к примеру, под CMP ==I подразумевается сравнение двух чисел типа INT на предмет равенства).

В цепи вы можете компаратор (блок сравнения) поставить вместо контакта. Немаркированные вход и выход служат для соединения других (двоичных) программных элементов.

Сравниваемые значения подаются на входы IN1 и IN2, результат сравнения – на выходе. Успешное сравнение эквивалентно замкнутому контакту («ток» протекает че-

рез компаратор). Если сравнение не успешно, то контакт разомкнут. Выход компаратора всегда должен быть подключен.

Представление (FBD)

Блочный элемент сравнения оснащен двумя входами IN1 и IN2 и немаркированным двоичным выходом. «Заголовок» в блочном элементе идентифицирует операцию выполняемого сравнения (к примеру, под CMP ==I подразумевается сравнение двух чисел типа INT на предмет равенства).

Сравниваемые значения подаются на входы IN1 и IN2, результат сравнения – на выходе. Если сравнение успешно, то выход компаратора показывает сигнальное состояние «1», иначе на выходе «0». Он всегда должен быть подключен.

Типы данных

Тип данных входов в функции сравнения зависит от этой функции. Например, входы типа REAL в функции сравнения CMP >R (сравнение «больше» чисел REAL). Переменные должны быть того же типа, что и входы. При использовании операндов с абсолютными адресами размеры операндов должны соответствовать типам данных. Например, можно использовать операнд размером в слово для типа данных INT.

Назначения битов в форматах данных вы можете найти в параграфе 3.5.4 «Простые типы данных».

Сравнение между числами REAL не истинно, если один или оба числа REAL недействительны (не являются числами типа REAL). Также устанавливаются биты состояния OS и OV. Как функции сравнения устанавливают остальные биты состояния, вы можете выяснить в главе 15 «Биты состояния».

Примеры

Рисунок 9.1 представляет пример для каждого типа данных. Функция сравнения выполняет сравнение в соответствии с определенными характеристиками, включая случай, когда типы данных не объявлены при использовании операндов с абсолютными адресами.

В случае пошагового программирования функции сравнения вы можете найти в каталоге программных элементов (Program Element Catalog), вызываемого командой меню View → Catalog (Вид → Каталог) [Ctrl + K] или Insert → Program Elements (Вставка → Программные элементы), в разделе «Comparator» («Компаратор»).

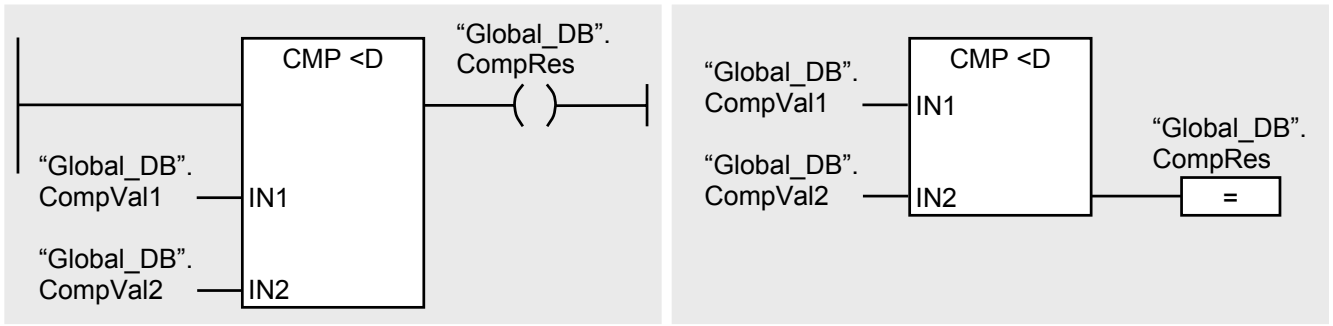
Функция сравнения в цепи (LAD)

Вы можете использовать функцию сравнения в цепи вместо контакта.

Сравнение в соответствии с INT Меркер M 99.0 сбрасывается, если значение в слове MW 92 равно 120; иначе он остается без изменений.



Сравнение в соответствии с DINT Переменная «CompRes» в блоке данных «Global_DB» устанавливается, если переменная «CompVal1» меньше «CompVal2»; иначе она сбрасывается.



Сравнение в соответствии с REAL Если переменная #Act_Value больше или равна переменной #Calibrat, то #NewCali устанавливается; иначе #NewCali остается без изменений.

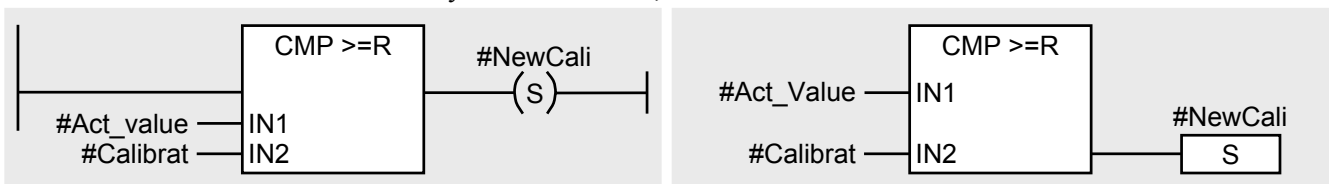


Рисунок 9.1 Примеры функции сравнения

Можно подсоединить контакты до и после функции сравнения последовательно и параллельно. Сами блочные элементы сравнения также могут быть соединены последовательно или параллельно. В случае последовательного соединения функций сравнения оба сравнения должны быть успешно выполнены, чтобы в цепи протекал ток. В случае параллельного подключения компараторов только одно условие сравнения должно выполняться, чтобы в параллельной схеме протекал ток.

Библиотека «LAD_Book» на дискете, поставляемой с книгой, содержит другие примеры представления и компоновки функций сравнения (FB 109 в программе «Digital Functions», «Числовые функции»).

Функция сравнения в логической схеме (FBD)

Вы можете поместить функцию сравнения на любой двоичный вход программного элемента. Результат сравнения впоследствии может комбинироваться с бинарными функциями. Библиотека «FBD_Book» на дискете, поставляемой с книгой, содержит примеры представления и компоновки функций сравнения (FB 109 в программе «Digital Functions», «Числовые функции»).

9.2 Описание функций сравнения

Сравнение «равно»

Сравнение «равно» интерпретирует содержимое входных переменных в соответствии с типами данных, определенными в функции сравнения, и проверяет, равны ли два значения. Условие сравнения выполняется («ток» протекает через выход компаратора или RLO равен «1»), когда две переменные имеют одинаковые значения.

Если, в случае сравнения переменных типа REAL, одна или обе входных переменных недействительны, то сравнение не успешно. При этом устанавливаются биты состояния OV и OS.

Сравнение «не равно»

Сравнение «не равно» интерпретирует содержимое входных переменных в соответствии с типами данных, определенными в функции сравнения, и проверяет, различны ли два значения. Условие сравнения выполняется («ток» на выходе компаратора течет или RLO равен «1»), когда две переменные имеют разные значения.

Если, в случае сравнения переменных типа REAL, одна или обе входных переменных недействительны, то сравнение не успешно. При этом устанавливаются биты состояния OV и OS.

Сравнение «больше»

Сравнение «больше» интерпретирует содержимое входных переменных в соответствии с типами данных, определенными в функции сравнения, и проверяет, является ли значение на входе IN1 больше значения на входе IN2. Если это имеет место, то условие сравнения выполняется («ток» протекает через выход компаратора или RLO равен «1»).

Если, в случае сравнения переменных типа REAL, одна или обе входных переменных недействительны, то сравнение не успешно. При этом устанавливаются биты состояния OV и OS.

Сравнение «больше или равно»

Сравнение «больше или равно» интерпретирует содержимое входных переменных в соответствии с типами данных, определенными в функции сравнения, и производит проверку, больше или равно значение на входе IN1 значению на входе IN2. Если это на самом деле так, то условие сравнения выполняется («ток» на выходе компаратора течет или RLO равен «1»).

Если, в случае сравнения переменных типа REAL, одна или обе входных переменных недействительны, то сравнение не успешно. При этом устанавливаются биты состояния OV и OS.

Сравнение «меньше»

Сравнение «меньше» интерпретирует содержимое входных переменных в соответствии с типами данных, определенными в функции сравнения, и проверяет, является ли значение на входе IN1 меньше значения на входе IN2. Если это так, то условие сравнения выполняется («ток» протекает через выход компаратора или RLO равен «1»).

Если, в случае сравнения переменных типа REAL, одна или обе входных переменных недействительны, то сравнение не успешно. При этом устанавливаются биты состояния OV и OS.

Сравнение «меньше или равно»

Сравнение «меньше или равно» интерпретирует содержимое входных переменных в соответствии с типами данных, определенными в функции сравнения, и проверяет, меньше или равно значение на входе IN1 значению на входе IN2. Если это так, то условие сравнения выполняется («ток» протекает через выход компаратора или RLO равен «1»).

Если, в случае сравнения переменных типа REAL, одна или обе входных переменных недействительны, то сравнение не успешно. При этом устанавливаются биты состояния OV и OS.

Содержание главы 10

<u>10</u>	<u>Арифметические функции</u>	4
<u>10.1</u>	<u>Обработка арифметической функции</u>	<u>4</u>
<u>10.2</u>	<u>Вычисления с типом данных INT</u>	<u>8</u>
<u>10.3</u>	<u>Вычисления с типом данных DINT</u>	<u>9</u>
<u>10.4</u>	<u>Вычисления с типом данных REAL</u>	<u>11</u>

10 Арифметические функции

Арифметические функции комбинируют два значения в соответствии с основными арифметическими операциями сложения, вычитания, умножения и деления. Вы можете применять арифметические функции к переменным типов INT, DINT и REAL (таблица 10.1).

Таблица 10.1 Обзор арифметических функций

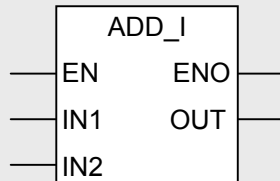
Арифметическая функция	С типом данных		
	INT	DINT	REAL
Сложение	ADD_I	ADD_DI	ADD_R
Вычитание	SUB_I	SUB_DI	SUB_R
Умножение	MUL_I	MUL_DI	MUL_R
Деление с частным в качестве результата	DIV_I	DIV_DI	DIV_R
Деление с остатком в качестве результата	-	MOD_DI	-

10.1 Обработка арифметической функции

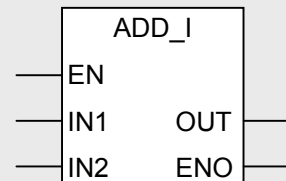
Представление

Арифметический блочный элемент
(в примере: сложение в соответствии с INT)

Представление LAD:



Представление FBD:



Кроме разрешающего входа (enable input) EN и разрешающего выхода (enable output) ENO блочный элемент арифметической функции имеет два входа IN1 и IN2 и выход OUT. «Заголовок» в блочном элементе идентифицирует исполняемое арифметическое действие (ADD_I, например, означает сложение чисел типа INT).

Комбинируемые значения подаются на входы IN1 и IN2, результат вычисления находится на выходе OUT. Входы и выход могут иметь различные типы данных в зависимости от арифметической функции. К примеру, в случае арифметической функции ADD_R (сложение чисел типа REAL) входы и выход отнесены к типу REAL. Используемые переменные должны быть того же типа данных, что и входы или выход. Если вы используете для операндов абсолютные адреса, то размеры операндов

должны соответствовать типам данных. Так, вы можете применить операнд размером в слово для типа данных INT.

Описание отдельных битов в каждом формате данных вы можете найти параграфе 3.5.4 «Простые типы данных».

Функция

Арифметическая функция выполняется, если на разрешающем входе присутствует «1» (во входе EN течет ток). Если во время вычисления возникает ошибка, то разрешающий выход устанавливается в «0»; в противном случае он устанавливается в «1». Если выполнение функции не разрешено (EN = «0»), то вычисление не производится, и ENO также обнуляется.

Если главное реле управления (MCR) активировано, то выход OUT устанавливается в нуль, когда арифметическая функция обрабатывается (EN = «1»). MCR не влияет на выход ENO.

ЕСЛИ EN == "1" или не используется		ИНАЧЕ
ТО		
OUT := IN1 Cfst IN2		ENO := "0"
ЕСЛИ возникла ошибка		
ТО	ИНАЧЕ	
ENO := "0"	ENO := "1"	

Cfst – арифметическая функция

Во время выполнения арифметической функции могут возникнуть следующие ошибки:

- Выход за пределы диапазона (переполнение) в вычислениях с типами INT и DINT;
- Исчезновение значащих разрядов и переполнение в вычислениях с типом REAL;
- Недействительное (недопустимое) число REAL в вычислениях с типом REAL.

За информацией о том, как арифметические функции устанавливают различные биты состояния, обратитесь к главе 15 «Биты состояния».

Примеры

На рисунке 10.1 приведены примеры для каждого типа данных. Арифметическая функция выполняет вычисление в соответствии с определенными параметрами, даже если не были объявлены типы данных при использовании операндов с абсолютными адресами.

В случае пошагового программирования арифметические функции вы можете найти в каталоге программных элементов (Program Element Catalog), вызываемого командой меню View → Catalog (Вид → Каталог) [Ctrl + K] или Insert → Program Elements (Вставка → Программные элементы), в разделе «Integer Math» («Вычисления с целыми числами», вычисления с типами данных INT и DINT) и в разделе «Floating-Point Math» («Вычисления с числами с плавающей запятой», вычисления с типом данных REAL).

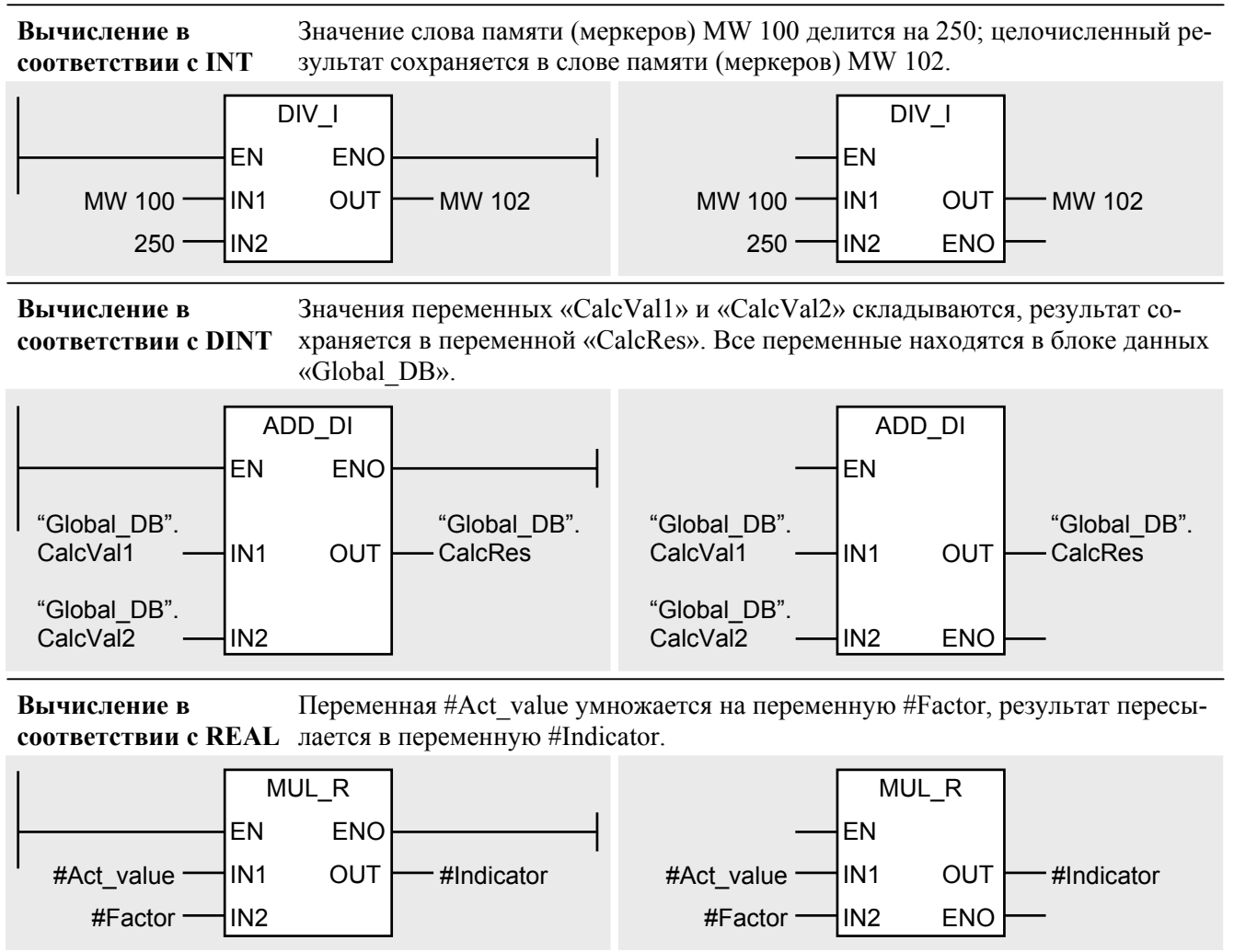


Рисунок 10.1 примеры арифметических функций

Арифметические функции в цепи (LAD)

Вы можете поставить контакты, соединив их последовательно или параллельно, перед входом EN и после выхода ENO.

Сам арифметический блочный элемент может быть помещен после Т-ветви и в ветви, ведущей непосредственно к левой питающей направляющей (шине питания). Эта ветвь также может иметь контакты перед входом EN и не может быть самой верхней ветвью.

Прямое соединение с левой питающей направляющей означает, что вы можете подключить арифметические блочные элементы параллельно. Когда блочные элементы подключаются параллельно, требуется катушка для завершения цепи. Если вы не производите проверку на наличие ошибки, назначьте катушке «пустой» операнд, к примеру, бит временных локальных данных.

Можно соединить арифметические блочные элементы последовательно. Если выход ENO предыдущего блочного элемента соединен с входом EN следующего, то последний срабатывает только, если предыдущий элемент был обработан без ошибок. Если вы хотите использовать результат из предыдущего блочного элемента в качестве входного значения для следующего блока, то переменные из области временных локальных данных предоставляют удобные промежуточные буферы.

Если вы компонуете несколько арифметических блочных элементов в одной цепи (параллельно левой питающей направляющей и в дальнейшем последовательно), блочные элементы самой верхней ветви обрабатываются слева направо, затем элементы второй ветви слева направо и так далее.

Библиотека «LAD_Book» на дискете, прилагаемой к книге, содержит примеры представления и группировки арифметических функций (FB 110 в программе «Digital Functions», «Числовые функции»).

Арифметические функции в логической цепи (FBD)

EN и ENO могут быть не назначены. Если требуется обработать арифметический блочный элемент согласно определенным условиям, то можно скомпоновать логические операции перед входом EN. Можно соединить выход ENO с бинарными входами других функций. Например, вы можете выстроить арифметические блочные элементы последовательно, таким образом, выход ENO предыдущего блочного элемента будет подключен к входу EN следующего. Если вы хотите использовать результат вычислений из предыдущего блочного элемента в качестве входного значения для последующего элемента, то переменные из области временных локальных данных предоставляют удобные промежуточные буферы.

Библиотека «FBD_Book» на дискете, прилагаемой к книге, содержит примеры представления и группировки арифметических функций (FB 110 в программе «Digital Functions», «Числовые функции»).

10.2 Вычисления с типом данных INT

Сложение с типом INT

Функция `ADD_I` интерпретирует значения на входах `IN1` и `IN2` как числа типа данных `INT` (целочисленный тип). Она складывает два числа и сохраняет сумму на выходе `OUT`.

После выполнения вычисления биты состояния `CC0` и `CC1` указывают на отрицательную, нулевую или положительную сумму. Биты состояния `OV` и `OS` индицируют любой выход за границы диапазона.

Вычитание с типом INT

Функция `SUB_I` интерпретирует значения на входах `IN1` и `IN2` как числа типа данных `INT`. Она вычитает значение на `IN2` из значения на `IN1` и сохраняет разность на выходе `OUT`.

После вычисления биты состояния `CC0` и `CC1` показывают, какова разность - отрицательная, нулевая или положительная. Биты состояния `OV` и `OS` сигнализируют о любом нарушении границ диапазона.

Умножение с типом INT

Функция `MUL_I` интерпретирует значения на входах `IN1` и `IN2` как числа типа данных `INT`. Она умножает два значения и сохраняет произведение на выходе `OUT`.

После выполнения вычисления биты состояния `CC0` и `CC1` указывают, каково произведение - отрицательное, равно нулю или положительное. Биты состояния `OV` и `OS` сигнализируют о любом нарушении границ диапазона типа `INT`.

Деление с типом INT

Функция `DIV_I` интерпретирует значения на входах `IN1` и `IN2` как числа типа данных `INT`. Она делит значение на входе `IN1` (делимое) на значение на входе `IN2` (делитель) и передает частное на выход `OUT`. Этот результат деления является целым. Частное равно нулю, если делимое равно нулю, а делитель не равен нулю, или если абсолютное значение делимого меньше абсолютного значения делителя. Частное отрицательное, если делитель отрицательный.

После вычисления биты состояния `CC0` и `CC1` показывают, частное отрицательное, равно нулю или положительное. Биты состояния `OV` и `OS` индицируют любой выход за границы диапазона. Деление на ноль выдает нулевое частное и устанавливает биты состояния `CC0`, `CC1`, `OV` и `OS` в «1».

10.3 Вычисления с типом данных DINT

Сложение с типом DINT

Функция ADD_DI интерпретирует значения на входах IN1 и IN2 как числа типа данных DINT (двойное целое). Она складывает два числа и сохраняет сумму на выходе OUT.

После выполнения вычисления биты состояния CC0 и CC1 указывают на отрицательную, нулевую или положительную сумму. Биты состояния OV и OS индицируют любой выход за границы диапазона.

Вычитание с типом DINT

Функция SUB_DI интерпретирует значения на входах IN1 и IN2 как числа типа данных DINT. Она вычитает значение на IN2 из значения на IN1 и сохраняет разность на выходе OUT.

После вычисления биты состояния CC0 и CC1 показывают, какова разность - отрицательная, нулевая или положительная. Биты состояния OV и OS сигнализируют о любом нарушении границ диапазона.

Умножение с типом DINT

Функция MUL_DI интерпретирует значения на входах IN1 и IN2 как числа типа данных DINT. Она умножает два значения и сохраняет произведение на выходе OUT.

После выполнения вычисления биты состояния CC0 и CC1 указывают, каково произведение - отрицательное, равно нулю или положительное. Биты состояния OV и OS сигнализируют о любом нарушении границ диапазона типа INT.

Деление с типом DINT, в качестве результата – частное

Функция DIV_DI интерпретирует значения на входах IN1 и IN2 как числа типа данных DINT. Она делит значение на входе IN1 (делимое) на значение на входе IN2 (делитель) и сохраняет частное на выходе OUT. Этот результат деления является целым. Частное равно нулю, если делимое равно нулю, а делитель не равен нулю, или если абсолютное значение делимого меньше абсолютного значения делителя. Частное отрицательное, если делитель меньше нуля.

После вычисления биты состояния CC0 и CC1 показывают, частное отрицательное, равно нулю или положительное. Биты состояния OV и OS индицируют любой выход за границы диапазона. Деление на ноль выдает нулевое частное и устанавливает биты состояния CC0, CC1, OV и OS в «1».

Деление с типом DINT, в качестве результата – остаток

Функция MOD_DI интерпретирует значения на входах IN1 и IN2 как числа типа данных DINT. Она делит значение на входе IN1 (делимое) на значение на входе IN2 (делитель) и сохраняет остаток на выходе OUT. Остаток – это то, что осталось от деления; он не соответствует десятичным разрядам. Если делимое отрицательное, то остаток также меньше нуля.

После вычисления биты состояния CC0 и CC1 показывают, каков остаток - отрицательный, равен нулю или положительный. Биты состояния OV и OS индицируют любой выход за границы диапазона. Деление на ноль возвращает нулевой остаток и устанавливает биты состояния CC0, CC1, OV и OS в «1».

10.4 Вычисления с типом данных REAL

Числа типа REAL (вещественные) представляются внутренне как числа с плавающей десятичной точкой (запятой) с двумя диапазонами чисел. Один диапазон с полной точностью («нормализованные» числа с плавающей точкой) и один диапазон с ограниченной точностью («ненормализованные» числа с плавающей точкой; обратитесь также к параграфу 3.5.4 «Простые типы данных»). CPU S7-400 производят вычисления в обоих диапазонах, CPU S7-300 – только в диапазоне с полной точностью. Если CPU S7-300 выполняют вычисление, результат которого находится в диапазоне с ограниченной точностью, то в качестве результата возвращается нуль, и сообщается о выходе из диапазона.

Сложение с типом REAL

Функция ADD_R интерпретирует значения на входах IN1 и IN2 как числа типа данных REAL. Она складывает два числа и сохраняет сумму на выходе OUT.

После выполнения вычисления биты состояния CC0 и CC1 указывают на отрицательную, нулевую или положительную сумму. Биты состояния OV и OS индицируют любой выход за границы диапазона.

В случае недопустимого вычисления (одно из входных значений является недействительным числом типа REAL, или вы пытаетесь сложить $+\infty$ и $-\infty$) ADD_R возвращает недействительное значение на выход OUT и устанавливает биты состояния CC0, CC1, OV и OS в «1».

Вычитание с типом REAL

Функция SUB_R интерпретирует значения на входах IN1 и IN2 как числа типа данных REAL. Она вычитает значение на входе IN2 из значения на входе IN1 и сохраняет разность на выходе OUT.

После вычисления биты состояния CC0 и CC1 показывают, какова разность - отрицательная, нулевая или положительная. Биты состояния OV и OS сигнализируют о любом нарушении границ диапазона.

В случае недопустимого вычисления (одно из входных значений является недействительным числом типа REAL, или вы пытаетесь вычесть $+\infty$ из $-\infty$) SUB_R возвращает недействительное значение на выход OUT и устанавливает биты состояния CC0, CC1, OV и OS в «1».

Умножение с типом REAL

Функция MUL_R интерпретирует значения на входах IN1 и IN2 как числа типа данных REAL. Она умножает два значения и сохраняет произведение на выходе OUT.

После выполнения вычисления биты состояния CC0 и CC1 указывают, каково произведение - отрицательное, равно нулю или положительное. Биты состояния OV и OS сигнализируют о любом нарушении границ диапазона.

В случае недопустимого вычисления (одно из входных значений является действительным числом типа REAL, или вы пытаетесь умножить ∞ на 0) MUL_R возвращает действительное значение на выход OUT и устанавливает биты состояния CC0, CC1, OV и OS в «1».

Деление с типом REAL

Функция DIV_R интерпретирует значения на входах IN1 и IN2 как числа типа данных REAL. Она делит значение на входе IN1 (делимое) на значение на входе IN2 (делитель) и передает частное на выход OUT.

После вычисления биты состояния CC0 и CC1 показывают, частное отрицательное, равно нулю или положительное. Биты состояния OV и OS индицируют любой выход за границы диапазона. Деление на ноль выдает нулевое частное и устанавливает биты состояния CC0, CC1, OV и OS в «1».

В случае недопустимого вычисления (одно из входных значений является действительным числом типа REAL, или вы пытаетесь поделить ∞ на ∞ или 0 на 0) DIV_R возвращает действительное значение на выход OUT и устанавливает биты состояния CC0, CC1, OV и OS в «1».

Содержание главы 11

<u>11</u>	<u>Математические функции</u>	4
<u>11.1</u>	<u>Обработка математической функции</u>	4
<u>11.2</u>	<u>Тригонометрические функции</u>	8
<u>11.3</u>	<u>Обратные тригонометрические функции</u>	9
<u>11.4</u>	<u>Различные математические функции</u>	10

11 Математические функции

В LAD и FBD представлены следующие математические функции:

- Синус, косинус, тангенс;
- Арксинус, арккосинус, арктангенс;
- Возведение в квадрат, извлечение квадратного корня;
- Экспоненциальная функция по основанию e , натуральный логарифм.

Все математические функции работают с числами типа данных REAL.

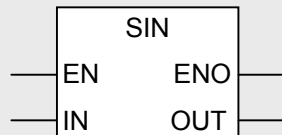
11.1 Обработка математической функции

Представление

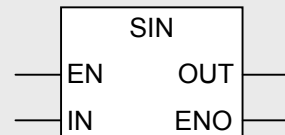
Блочный элемент математической функции имеет вход IN и выход OUT, а также разрешающий вход EN и разрешающий выход ENO. «Заголовок» в блочном элементе идентифицирует выполняемую математическую функцию (например, SIN означает синус).

Математический блочный элемент (в примере: синус)

Представление LAD:



Представление FBD:



Входное значение подается на вход IN, и результат математической функции выводится на выход OUT. Вход и выход относятся к типу данных REAL. Операнды, адресуемые с использованием абсолютных адресов, должны иметь размер двойного слова.

Обратитесь к параграфу 3.5.4 «Простые типы данных» за описанием битов формата REAL.

Функция

Математическая функция выполняется, если на разрешающем входе (EN) присутствует «1», или если через вход EN проходит «электрический ток». Если при вычислении возникнет ошибка, то разрешающий выход устанавливается в «0»; иначе он ус-

танавливается в «1». Если исполнение функции не разрешено (EN = «0»), то вычисление не производится, и ENO также становится равным «0».

ЕСЛИ EN == "1" или не используется		ИНАЧЕ
ТО		
OUT := Mfct (IN)		ENO := "0"
ЕСЛИ возникла ошибка		
ТО	ИНАЧЕ	
ENO := "0"	ENO := "1"	

Mfct – математическая функция

Если главное реле управления (MCR) активно, то когда математическая функция обрабатывается (EN = «1»), выход OUT установлен в ноль. MCR не воздействует на ENO.

В математической функции могут возникнуть следующие ошибки:

- Выход за пределы диапазона (исчезновение значащих разрядов и переполнение);
- Недействительное (недопустимое) число типа REAL в качестве входного значения.

В главе 15 «Биты состояния» объясняется, как математические функции устанавливают биты состояния.

Примеры

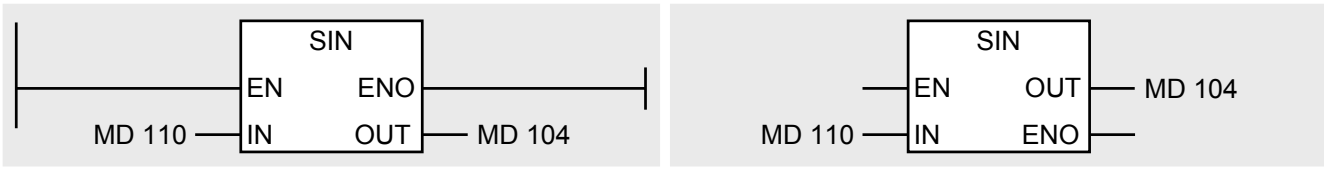
Рисунок 11.1 содержит три примера математических функций. Математическая функция выполняет вычисление в соответствии с типом данных REAL, даже если при использовании операндов с абсолютной адресацией типы данных не были объявлены.

При пошаговом программировании математические функции вы можете найти в каталоге программных элементов (Program Element Catalog), вызываемого командой меню View → Catalog (Вид → Каталог) [Ctrl + K] или Insert → Program Elements (Вставка → Программные элементы), в разделе «Floating-Point Math» («Вычисления с числами с плавающей запятой»).

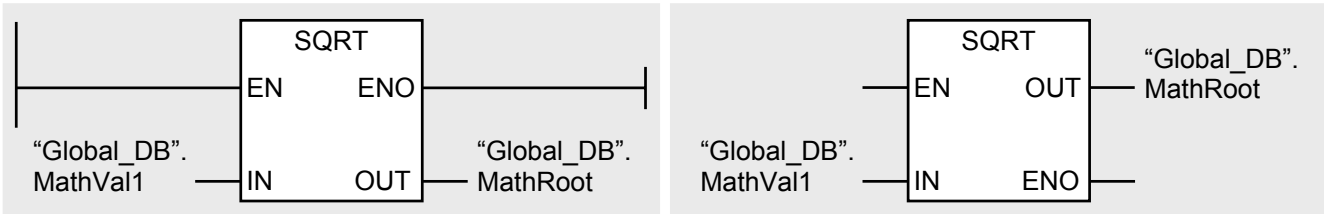
Математическая функция в цепи (LAD)

Вы можете размещать контакты и соединять их последовательно и параллельно перед входом EN и после выхода ENO.

Синус Значение двойного слова памяти (меркеров) MD 110 содержит величину в радианах. От этого значения берется синус и сохраняется в двойном слове памяти (меркеров) MD 104.



Квадратный корень Квадратный корень извлекается из значения переменной «MathVal1» и сохраняется в переменной «MathRoot».



Экспонента Переменная #Result содержит число e, возведенное в степень #Exponent.



Рисунок 11.1 Примеры математических функций

Сам математический блочный элемент может быть вставлен после T-ветви или находится на ветви, соединенной непосредственно с левой направляющей (питающей шиной). Эта ветвь также может иметь контакты перед входом EN и не может быть самой верхней ветвью.

Прямое подключение к левой питающей направляющей позволяет вам скомпоновать математические блочные элементы параллельно. Когда блочные элементы подключаются параллельно, требуется катушка для завершения цепи. Если вы не производите проверку на наличие ошибки, назначьте катушке «пустой» операнд, например, бит временных локальных данных.

Можно соединить математические блочные элементы последовательно. Если выход ENO предыдущего блочного элемента соединен с входом EN следующего, то последний срабатывает только при условии, если предыдущий элемент был обработан без ошибок. Если вы хотите использовать результат из предыдущего блочного элемента в качестве входного значения для следующего блока, то переменные из области временных локальных данных предоставляют удобные промежуточные буферы.

Если вы компоуете несколько математических блочных элементов в одной цепи (параллельно левой питающей направляющей и в дальнейшем последовательно), блочные элементы самой верхней ветви обрабатываются слева направо, затем элементы второй ветви слева направо и так далее.

Библиотека «LAD_Book» на дискете, прилагаемой к книге, содержит примеры представления и группировки математических функций (FB 111 в программе «Digital Functions», «Числовые функции»).

Математическая функция в логической схеме (FBD)

EN и ENO не должны быть назначены. Если требуется обработать математический блочный элемент в зависимости от определенных условий, то можно запрограммировать бинарные логические операции перед входом EN. Можно соединить выход ENO с бинарными входами других функций. Например, вы можете соединить математические блочные элементы последовательно, таким образом, выход ENO предыдущего блочного элемента будет подключен к входу EN следующего. Если вы хотите использовать результат вычислений предыдущего блочного элемента в качестве входного значения для последующего элемента, то переменные из области временных локальных данных предоставляют удобные промежуточные буферы.

Библиотека «FBD_Book» на дискете, прилагаемой к книге, содержит примеры представления и группировки математических функций (FB 111 в программе «Digital Functions», «Числовые функции»).

11.2 Тригонометрические функции

Тригонометрические функции

SIN Синус,
 COS Косинус и
 TAN Тангенс

принимают на входе значение угла в радианах как число типа REAL.

Для размера угла по соглашению используются две величины: градусы от 0° до 360° и радианы от 0 до 2π (где $\pi = +3.141593e+00$). Они обе могут быть пропорционально конвертированы. Например, радианная мера для угла 90° (прямого) равна $\pi/2$ или $+1.570796e+00$. В случае значений, больших 2π ($+6.283185e+00$), 2π или несколько 2π вычитается из него, пока входное значение тригонометрической функции не станет меньше 2π .

Пример (рисунок 11.2а или 11.3, сеть 4, Network 4): вычисление неактивной (холостой) мощности (idle power) $P_s = U I \sin \varphi$.

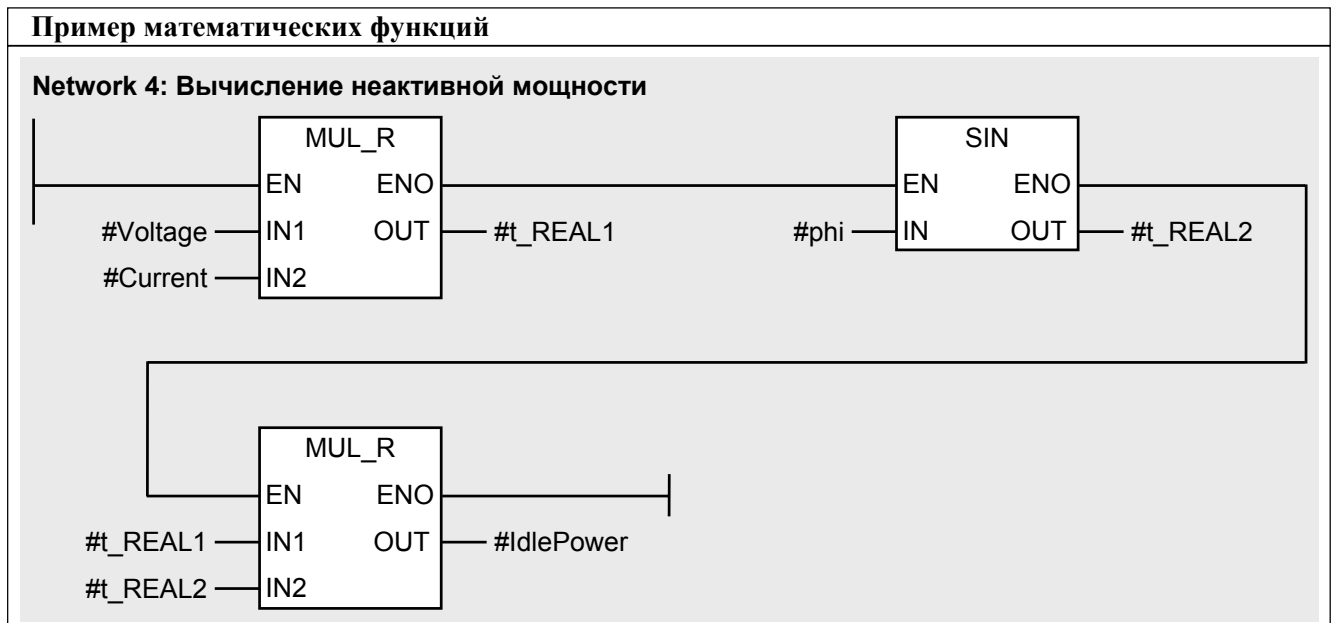


Рисунок 11.2а Пример математических функций (LAD)

11.3 Обратные тригонометрические функции

Обратные тригонометрические функции (аркфункции)

ASIN Арксинус,
ACOS Арккосинус и
ATAN Арктангенс.

Аркфункции являются обратными функциями соответствующих тригонометрических функций. Они допускают на входе IN числа типа REAL определенного диапазона и возвращают угол в радианной мере (таблица 11.1).

Если на входе IN присутствует значение не из допустимого диапазона, то аркфункция возвращает недействительное число типа REAL, и ENO = «0», а также она устанавливает биты состояния CC0, CC1, OV и OS в «1».

Таблица 11.1 Диапазон аркфункций

Функция	Допустимый диапазон	Возвращаемое значение
ASIN (арксинус)	от -1 до +1	от $-\pi/2$ до $+\pi/2$
ACOS (арккосинус)	от -1 до +1	от 0 до π
ATAN (арктангенс)	Весь диапазон	от $-\pi/2$ до $+\pi/2$

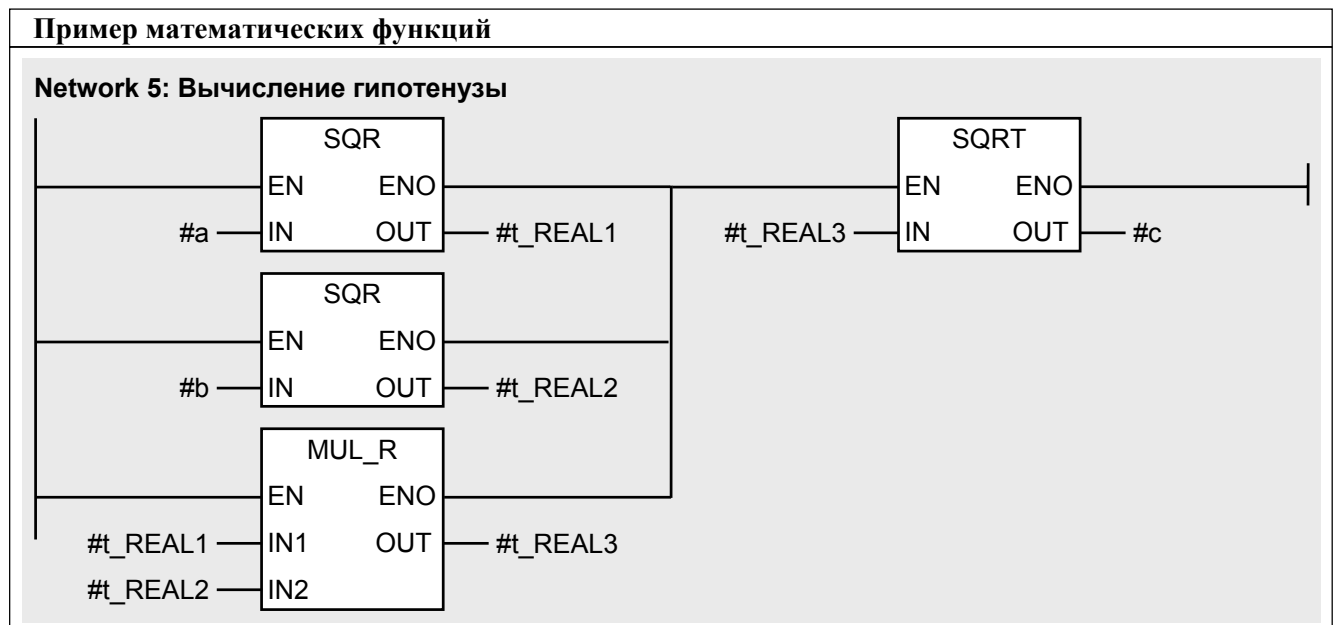


Рисунок 11.26 Пример математических функций (LAD)

11.4 Различные математические функции

Применять также можно следующие математические функции:

SQR	Вычисление квадрата числа,
SQRT	Вычисление квадратного корня числа,
EXP	Вычисление экспоненты по основанию e ,
LN	Нахождение натурального логарифма (логарифма по основанию e).

Вычисление квадрата

Функция SQR возводит в квадрат значение на входе IN и сохраняет результат на выходе OUT.

Пример: см. «Вычисление квадратного корня».

Вычисление квадратного корня

Функция SQRT извлекает квадратный корень значения на входе IN и сохраняет результат на выходе OUT. Если значение на входе IN меньше нуля, то SQRT устанавливает биты состояния CC0, CC1, OV и OS в «1» и возвращает недействительное число REAL. Если значение на входе IN -0 (минус ноль), возвращается -0 .

Пример: $c = \sqrt{a^2 + b^2}$

Рисунок 11.2б (приведенный выше) или рисунок 11.3, сеть 5 (Network 5): сначала находятся квадраты переменных a и b , затем они складываются. Завершается пример извлечением квадратного корня из суммы. Временные локальные данные используются в качестве промежуточной памяти.

(Если вы объявили b или c как локальную переменную, вы должны предварить ее знаком #, чтобы редактор распознал ее как локальную переменную; если b или c является глобальной переменной, она должна быть заключена в кавычки.)

Вычисление экспоненты по основанию e

Функция EXP вычисляет экспоненциальное значение по основанию e (равному $2.718282e+00$) и значению на входе IN (e^{IN}) и сохраняет результат на выходе OUT.

Вы можете вычислять любое экспоненциальное значение, используя формулу $a^b = e^{b \ln a}$.

Нахождение натурального логарифма

Функция LN находит натуральный логарифм по основанию e ($= 2.718282e+00$) числа на входе IN и сохраняет его на выходе OUT. Если значение на входе меньше или равно нулю, то LN устанавливает биты состояния CC0, CC1, OV и OS в «1» и возвращает недействительное число REAL.

Натуральный логарифм является обратной функцией экспоненциальной функции: если $y = e^x$, то $x = \ln y$.

Чтобы найти любой логарифм, воспользуйтесь формулой

$$\log_b a = \frac{\log_n a}{\log_n b}$$

Здесь b или n – любое основание. Если взять $n = e$, то можно найти логарифм по любому основанию, используя натуральный логарифм:

$$\log_b a = \frac{\ln a}{\ln b}$$

В особом случае для основания 10 формула следующая:

$$\lg a = \frac{\ln a}{\ln 10} = 0.4342945 \ln a$$

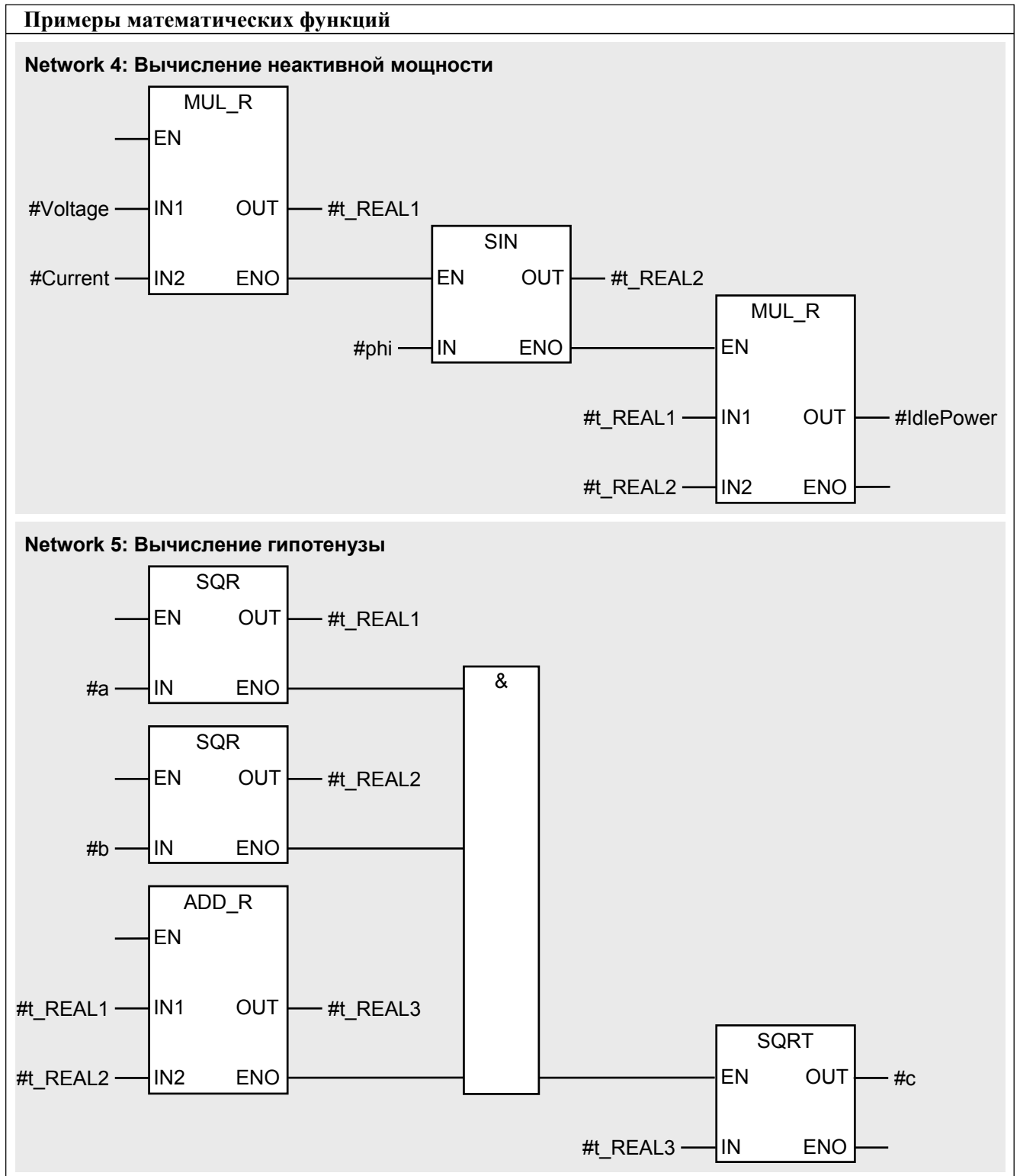


Рисунок 11.3 Примеры математических функций (FBD)

Содержание главы 12

<u>12</u>	<u>Функции преобразования</u>	4
12.1	<u>Обработка функции преобразования</u>	4
12.2	<u>Преобразование чисел типов INT и DINT</u>	8
12.3	<u>Преобразование чисел типа BCD</u>	10
12.4	<u>Преобразование чисел типа REAL</u>	11
12.5	<u>Различные функции преобразования</u>	13

12 Функции преобразования

Функции преобразования конвертируют типы данных переменных. На рисунке 12.1 представлен обзор преобразований типов данных, описываемых в этой главе.

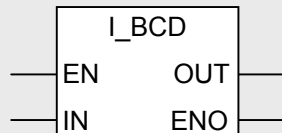
12.1 Обработка функции преобразования

Представление

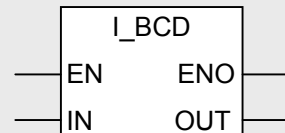
Кроме разрешающего входа (enable input) EN и разрешающего выхода (enable output) ENO блочный элемент функции преобразования оснащен входом IN и выходом OUT. «Заголовок» блочного элемента идентифицирует выполняемую функцию преобразования (например, I_BCD означает преобразование INT в BCD).

Блочный элемент преобразования (в примере: INT в BCD)

Представление LAD:



Представление FBD:



Конвертируемое значение подается на вход IN, результат конвертирования находится на выходе OUT. Тип данных входа и выхода зависит от функции преобразования. В функции преобразования DI_R (DINT в REAL), к примеру, вход имеет тип DINT, а тип выхода – REAL. Используемые переменные должны иметь те же типы данных, что у входа или выхода.

Если вы используете операнды с абсолютными адресами, то размеры операндов должны соответствовать типам данных, так, вы можете использовать операнд размером в слово для типа данных INT.

Функция

Функция преобразования выполняется, если на разрешающем входе присутствует «1» (если ток течет во входе EN). Если во время преобразования возникнет ошибка, то разрешающий выход ENO устанавливается в «0»; в противном случае он устанавливается в «1». Если исполнение функции не разрешено (EN = «0»), преобразование не происходит, и ENO также обнулен (см. блок-схему ниже).

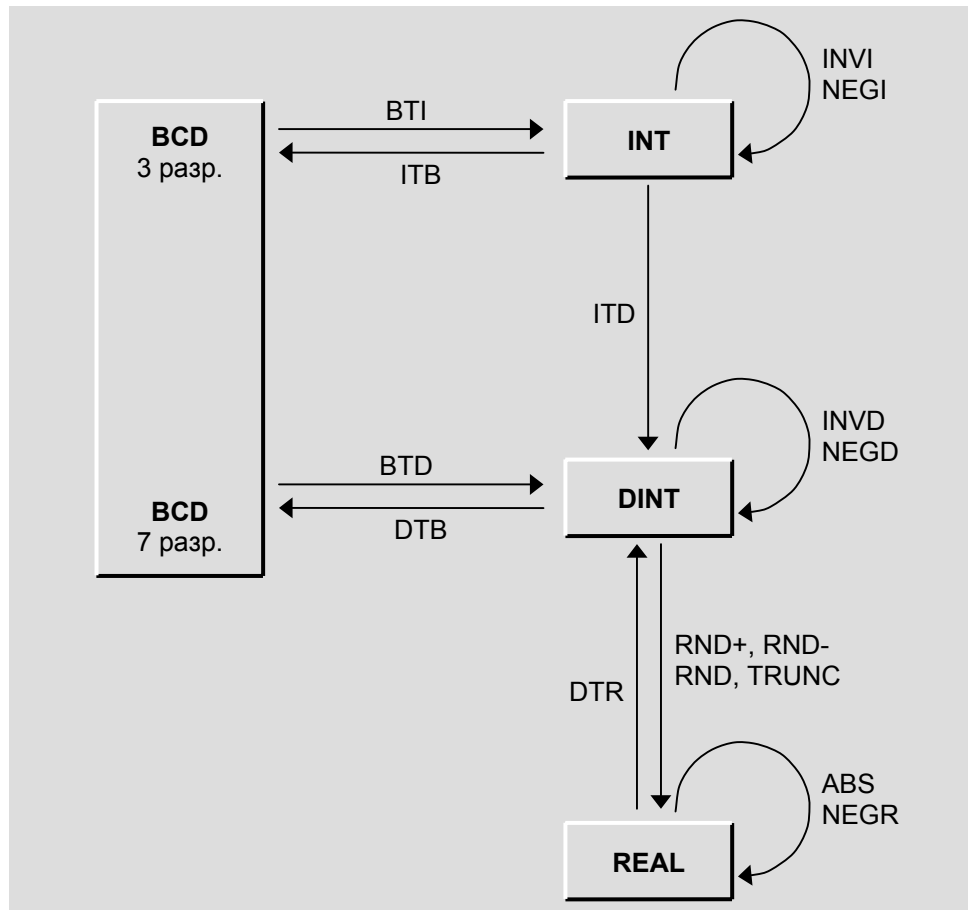


Рисунок 12.1 Обзор функций преобразования

Если главное реле управления (MCR) активировано, то при обработке функции преобразования ($EN = \langle 1 \rangle$) выход **OUT** устанавливается в ноль. MCR на выход **ENO** не воздействует.

ЕСЛИ $EN == \langle 1 \rangle$ или не используется		
ТО		ИНАЧЕ
OUT := Confct (IN)		
ЕСЛИ возникла ошибка		
ТО	ИНАЧЕ	
ENO := "0"	ENO := "1"	ENO := "0"

Confct – функция преобразования

Не все функции преобразования сообщают об ошибке. Ошибка возникает, только если нарушается допустимый диапазон чисел (**I_BCD**, **DI_BCD**) или определено недействительное число **REAL** (**FLOOR**, **CEIL**, **ROUND**, **TRUNC**).

Если входное значение для преобразования **BCD_I** или **BCD_DI** содержит псевдотетраду, то исполнение программы прерывается, и вызывается организационный

блок обработки ошибок ОВ 121 (синхронные ошибки работы программы). В главе 15 «Биты состояния» разъясняется, как функции преобразования устанавливают биты состояния.

На рисунке 12.2 демонстрируется по одному примеру для каждого типа данных. Функции преобразования конвертируют в соответствии с определенными характеристиками, даже если при использовании операндов с абсолютными адресами типы данных не были объявлены.

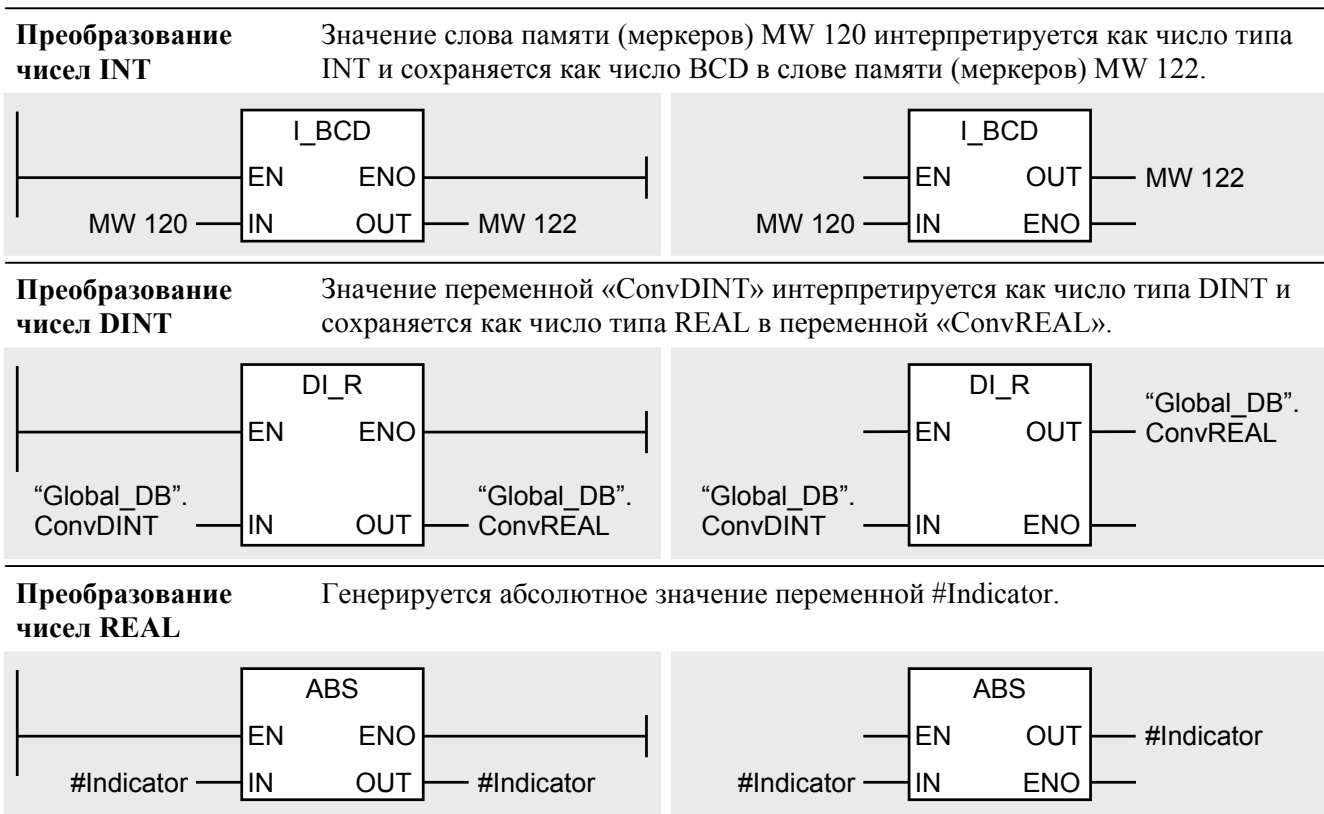


Рисунок 12.2 Примеры функций преобразования

При пошаговом программировании функции преобразования вы можете найти в каталоге программных элементов (Program Element Catalog), вызываемого командой меню View → Catalog (Вид → Каталог) [Ctrl + K] или Insert → Program Elements (Вставка → Программные Элементы), в разделе «Convert» («Преобразование»).

Функция преобразования в цепи (LAD)

Вы можете размещать контакты и соединять их последовательно и параллельно перед входом EN и после выхода ENO.

Сам блочный элемент преобразования может быть вставлен после Т-ветви или находиться на ветви, соединенной непосредственно с левой направляющей (питающей

шиной). Эта ветвь также может иметь контакты перед входом EN и не может быть самой верхней ветвью.

Прямое подключение к левой питающей направляющей позволяет вам соединить блочные элементы преобразования параллельно. Когда блочные элементы подключаются параллельно, требуется катушка для завершения цепи. Если вы не производите проверку на наличие ошибки, назначьте катушке «пустой» операнд, например, бит временных локальных данных.

Вы можете соединить блочные элементы преобразования последовательно. Если выход ENO предыдущего блочного элемента соединен с входом EN следующего, то последний обрабатывается только при условии, если предыдущий элемент сработал без ошибок. Если вы хотите использовать результат из предыдущего блочного элемента в качестве входного значения для следующего блока, то переменные из области временных локальных данных предоставляют удобные промежуточные буферы.

Если вы компонуете несколько блочных элементов преобразования в одной цепи (параллельно левой питающей направляющей и в дальнейшем последовательно), блочные элементы самой верхней ветви обрабатываются слева направо, затем элементы второй ветви слева направо и так далее.

Библиотека «LAD_Book» на дискете, прилагаемой к книге, содержит примеры представления и группировки функций преобразования (FB 112 в программе «Digital Functions», «Числовые функции»).

Функции преобразования в логической схеме (FBD)

EN и ENO не должны быть назначены. Если требуется обработать преобразующий блочный элемент в зависимости от определенных условий, то можно скомпоновать бинарные логические операции перед входом EN. Вы можете соединить выход ENO с бинарными входами других функций. Например, можно соединить блочные элементы преобразования последовательно, таким образом, выход ENO предыдущего блочного элемента будет подключен к входу EN следующего.

Если вы хотите использовать результат вычислений предыдущего блочного элемента в качестве входного значения для последующего элемента, то переменные из области временных локальных данных предоставляют удобные промежуточные буферы.

Библиотека «FBD_Book» на дискете, прилагаемой к книге, содержит примеры представления и группировки функций преобразования (FB 112 в программе «Digital Functions», «Числовые функции»).

12.2 Преобразование чисел типов INT и DINT

В таблице 12.1 показаны функции преобразования для чисел типов INT и DINT. К входам и выходам должны применяться переменные определенных типов данных или абсолютно адресуемые операнды соответствующих размеров (например, операнд размером в слово для типа данных INT).

Таблица 12.1 Преобразование чисел типов INT и DINT

Преобразование типа данных	Блочный элемент	Тип данных параметра	
		IN	OUT
INT в DINT	I_DI	INT	DINT
INT в BCD	I_BCD	INT	WORD
DINT в BCD	DI_BCD	DINT	DWORD
DINT в REAL	DI_R	DINT	REAL

Преобразование из INT в DINT

Функция I_DT интерпретирует значение на входе IN как число типа данных INT и пересылает его в младшее слово выхода OUT. Сигнальное состояние 15-го бита (знак) переносится в биты с 16-го по 31-й старшего слова выхода OUT.

Преобразование из INT в DINT об ошибках не сообщает.

Преобразование из INT в BCD

Функция I_BCD интерпретирует значение на входе IN как число типа данных INT и конвертирует его в 3-разрядное число BCD (двоично-десятичный код) на выходе OUT. Три правых десятичных разряда представляют абсолютное значение десятичного числа. Знак находится в битах с 12-го по 15-й. Если все биты равны «0», то знак положительный; если все биты установлены в «1», то знак отрицательный.

Если число INT слишком большое для преобразования в BCD (> 999), то функция I_BCD устанавливает биты состояния OV и OS. Преобразование не производится.

Преобразование из DINT в BCD

Функция DI_BCD интерпретирует значение на входе IN как число типа данных DINT и преобразует его в 7-разрядное число BCD на выходе OUT. Семь правых десятичных разрядов представляют абсолютное значение десятичного числа. Знак находится в битах с 28-го по 31-й. Если все биты равны «0», то знак положительный; если все биты установлены в «1», то знак отрицательный.

Если число DINT слишком большое для преобразования в число BCD (> 9 999 999), то функция DI_BCD устанавливает биты состояния OV и OS. Преобразование не выполняется.

Преобразование из DINT в REAL

Функция DI_R интерпретирует значение на входе IN как число типа данных DINT и преобразует его в число типа REAL на выходе OUT.

Так как число в формате DINT имеет более высокую точность, чем число в формате REAL, то при конвертировании может произойти округление. Число типа REAL округляется до следующего целого числа (в соответствии с функцией ROUND).

Функция DI_R об ошибке не сообщает.

12.3 Преобразование чисел типа BCD

В таблице 12.2 показаны функции преобразования для чисел типа BCD. К входам и выходам должны применяться переменные определенного типа данных или абсолютно адресуемые операнды соответствующих размеров (например, операнд размером в слово для типа данных INT).

Таблица 12.2 Преобразование чисел BCD

Преобразование типа данных	Блочный элемент	Тип данных параметра	
		IN	OUT
BCD в INT	BCD_I	WORD	INT
BCD в DINT	BCD_DI	DWORD	DINT

Преобразование из BCD в INT

Функция BCD_I интерпретирует значение на входе IN как 3-разрядное BCD-число и преобразует его в число типа INT на выходе OUT. Три правых разряда представляют абсолютное значение десятичного числа. Знак находится в битах с 12-го по 15-й. Если в этих битах нули, то знак положительный; если они заполнены значением «1», то знак отрицательный. При конвертировании во внимание берется только 15-й бит.

Если BCD-число содержит псевдотетраду (численное значение от 10 до 15 или шестнадцатеричное от A до F), то CPU сигнализирует о программной ошибке и вызывает организационный блок OB 121 (синхронная ошибка). Если этот блок недоступен, CPU переходит в режим STOP.

Функция BCD_I биты статуса не устанавливает.

Преобразование из BCD в DINT

Функция BCD_DI интерпретирует значение на входе IN как 7-разрядное BCD-число и преобразует его в число типа DINT на выходе OUT. Семь правых разрядов представляют абсолютное значение десятичного числа. Знак находится в битах с 28-го по 31-й. Если в этих битах «0», то знак положительный; если они заполнены значением «1», то знак отрицательный. При конвертировании во внимание берется только 31-й бит.

Если BCD-число содержит псевдотетраду (численное значение от 10 до 15 или шестнадцатеричное от A до F), то CPU сигнализирует о программной ошибке и вызывает организационный блок OB 121 (синхронная ошибка). Если этот блок недоступен, CPU переходит в режим STOP.

Функция BCD_DI биты статуса не устанавливает.

12.4 Преобразование чисел типа REAL

Есть несколько функций для преобразования числа в формате REAL в число формата DINT (преобразование дробного значения в целое число) (таблица 12.3). Они отличаются по способу округления. К входам и выходам должны быть применены переменные определенного типа данных или абсолютно адресуемые операнды размером в двойное слово.

Таблица 12.3 Преобразование чисел типа REAL в числа типа DINT

Преобразование типа данных с округлением	Блочный элемент	Тип данных параметра	
		IN	OUT
До следующего большего целого числа	CEIL	REAL	DINT
До следующего меньшего целого числа	FLOOR	REAL	DINT
До следующего целого числа	ROUND	REAL	DINT
Без округления	TRUNC	REAL	DINT

Округление до следующего большего целого числа

Функция CEIL интерпретирует значение на входе IN как число в формате REAL и преобразует его в число формата DINT на выходе OUT. CEIL возвращает целое, большее или равное конвертируемому числу.

Если значение на входе IN находится вне диапазона, допустимого для чисел формата DINT, или оно не соответствует числу в формате REAL, то CEIL устанавливает биты состояния OV и OS. Преобразование не выполняется.

Округление до следующего меньшего целого числа

Функция FLOOR интерпретирует значение на входе IN как число в формате REAL и приводит его к числу формата DINT на выходе OUT. FLOOR возвращает целое, меньшее или равное конвертируемому числу.

Если значение на входе IN находится вне диапазона, допустимого для чисел в формате DINT, или оно не соответствует числу в формате REAL, то FLOOR устанавливает биты состояния OV и OS. Преобразование не выполняется.

Округление до следующего меньшего целого числа

Функция ROUND интерпретирует значение на входе IN как число в формате REAL и преобразует его в число формата DINT на выходе OUT. ROUND возвращает следующее целое число. Если результат находится точно между нечетным и четным числом, предпочтение отдается четному числу.

Если значение на входе IN находится вне диапазона, допустимого для чисел в формате DINT, или оно не соответствует числу в формате REAL, то ROUND устанавливает биты состояния OV и OS. Преобразование не выполняется.

Без округления

Функция TRUNC интерпретирует значение на входе IN как число в формате REAL и преобразует его в число формата DINT на выходе OUT. TRUNC возвращает целую составляющую преобразуемого числа; дробная составляющая отсекается.

Если значение на входе IN находится вне диапазона, допустимого для чисел в формате DINT, или оно не соответствует числу в формате REAL, то TRUNC устанавливает биты состояния OV и OS. Преобразование не выполняется.

Обобщение преобразований из типа REAL к типу DINT

Таблица 12.4 демонстрирует различные результаты функций для преобразования из REAL в DINT. В качестве примера выбран диапазон от -1 до +1.

Таблица 12.4 Режимы округления для преобразования чисел типа REAL

Значение на входе		Результат			
REAL	DW#16#	ROUND	CEIL	FLOOR	TRUNC
1.0000001	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

12.5 Различные функции преобразования

Другие имеющиеся функции преобразования – это функция дополнения до единицы (обратный код), отрицание и получение абсолютного значения числа типа REAL (таблица 12.5). К входам и выходам должны применяться переменные определенного типа данных или абсолютно адресуемые операнды соответствующего размера (например, операнд размером в двойное слово для типа данных DINT).

Таблица 12.5 Различные функции преобразования

Преобразование	Блочный элемент	Тип данных параметра	
		IN	OUT
Дополнение до единицы типа INT	INV_I	INT	INT
Дополнение до единицы типа DINT	INV_DI	DINT	DINT
Отрицание числа INT	NEG_I	INT	INT
Отрицание числа DINT	NEG_DI	DINT	DINT
Отрицание числа REAL	NEG_R	REAL	REAL
Абсолютное значение числа REAL	ABS	REAL	REAL

Дополнение до единицы типа INT

Функция INV_I производит операцию отрицания значения на входе IN побитно (бит за битом) и записывает его в выход OUT. INV_I заменяет нули единицами и наоборот. Функция INV_I об ошибке не сигнализирует.

Дополнение до единицы типа DINT

Функция INV_DI производит операцию отрицания значения на входе IN побитно (бит за битом) и записывает его в выход OUT. INV_DI заменяет нули единицами и наоборот. Функция INV_DI об ошибке не сообщает.

Отрицание типа INT

Функция NEG_I интерпретирует значение на входе IN как число типа INT, меняет знак путем генерирования дополнения до двух (дополнительного кода) и записывает преобразованное значение на выход OUT. NEG_I идентична умножению на -1 . Функция NEG_I устанавливает биты состояния CC0, CC1, OV и OS.

Отрицание типа DINT

Функция NEG_DI интерпретирует значение на входе IN как число типа DINT, меняет знак путем генерирования дополнения до двух (дополнительного кода) и записывает конвертированное значение на выход OUT. NEG_DI идентична умножению на -1 . Функция NEG_DI устанавливает биты состояния CC0, CC1, OV и OS.

Отрицание типа REAL

Функция NEG_R интерпретирует значение на входе IN как число типа REAL, умножает его на -1 и записывает на выход OUT. NEG_R меняет знак мантиссы даже недействительного числа REAL. NEG_R об ошибке не сообщает.

Получение абсолютного значения типа REAL

Функция ABS интерпретирует значение на входе IN как число типа REAL, генерирует абсолютное значение этого числа и записывает его на выход OUT. ABS устанавливает в «0» знак мантиссы даже недействительного числа REAL. ABS об ошибках не сигнализирует.

Содержание главы 13

<u>13</u>	<u>Функции сдвига</u>	4
13.1	Обработка функции сдвига	4
13.2	Сдвиг	8
13.3	Циклический сдвиг	10

13 Функции сдвига

Функции сдвига смещают содержимое переменной побитно вправо или влево. Выдвинутые биты либо теряются, либо используются для заполнения переменной с другой стороны. Таблица 13.1 предоставляет обзор функций сдвига.

Таблица 13.1 Обзор функций сдвига

Функции сдвига	Переменная размером в слово	Переменная размером в двойное слово
Сдвиг влево	SHL_W	SHL_DW
Сдвиг вправо	SHR_W	SHR_DW
Сдвиг со знаком	SHR_I	SHR_DI
Циклический сдвиг влево	-	ROL_DW
Циклический сдвиг вправо	-	ROR_DW

13.1 Обработка функции сдвига

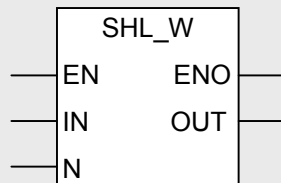
Представление

Кроме разрешающего входа (enable input) EN и разрешающего выхода (enable output) ENO блочный элемент функции сдвига имеет вход IN, вход N и выход OUT. «Заголовок» в блочном элементе идентифицирует выполняемую функцию сдвига (например, SHL_W означает сдвиг влево переменной размером в слово).

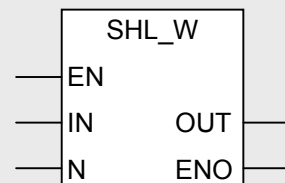
Блочный элемент сдвига

(в примере: сдвиг влево переменной размером в слово)

Представление LAD:



Представление FBD:



Значение, предназначенное для сдвига, подается на вход IN. Количество разрядов для сдвига задается на входе N. Результат выдается на выход OUT. Вход и выход в зависимости от функции сдвига имеют различные типы данных. Например, вход и выход принадлежат к типу DWORD для функции сдвига SHR_DW (сдвиг вправо переменной размером в двойное слово). Применяемые переменные должны быть того же типа данных, что и вход или выход. Если вы используете операнды с абсолютными адресами, то размеры этих операндов должны соответствовать типу данных; к примеру, для типа данных INT можно использовать операнд размером в слово. Тип данных входа N – WORD для каждой функции сдвига.

Описание битов в каждом формате данных содержится в параграфе 3.5.4 «Простые типы данных».

Функция

Функция сдвига выполняется, если на разрешающем входе присутствует «1», или когда «ток» протекает через вход EN. Тогда ENO установлен в «1». Если исполнение функции не разрешено (EN = «0»), то сдвиг не происходит, ENO сбрасывается в «0».

ЕСЛИ EN == "1" или не используется	
ТО	ИНАЧЕ
OUT := Sfct (IN, N)	
ENO := "1"	ENO := "0"

Sfct – функция сдвига

Когда главное реле управления (MCR) активировано, при обработке функции сдвига (EN = «1») выход OUT установлен в ноль. MCR на выход ENO не воздействует.

В главе 15 «Биты состояния» объясняется, как функции сдвига устанавливают биты состояния.

Примеры

На рисунке 13.1 показано по одному примеру для различных функций сдвига.

При пошаговом программировании функции сдвига вы можете найти в каталоге программных элементов (Program Element Catalog), вызываемого командой меню View → Catalog (Вид → Каталог) [Ctrl + K] или Insert → Program Elements (Вставка → Программные элементы), в разделе «Shift» («Сдвиг»).

Функция сдвига в цепи (LAD)

Контакты перед входом EN и после выхода ENO могут быть соединены последовательно и параллельно.

Сам блочный элемент функции сдвига может быть помещен после Т-ветви или вставлен в ветвь, соединенную непосредственно с левой направляющей (шиной питания). В этой ветви также могут быть контакты до входа IN, и она не может быть самой верхней ветвью.

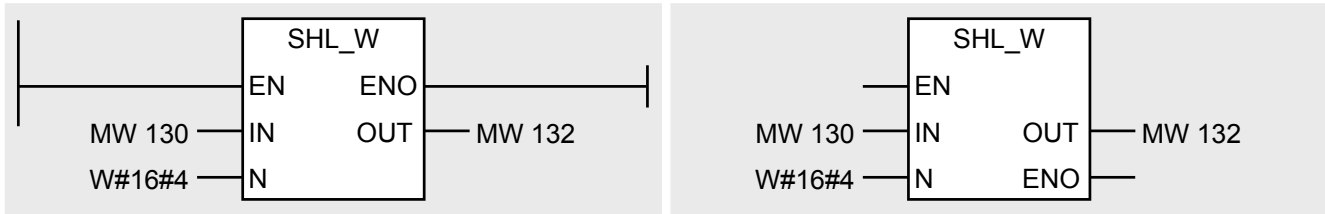
Прямое подключение к левой питающей направляющей позволяет вам соединить блочные элементы сдвига параллельно. Когда блочные элементы подключаются параллельно, требуется катушка для завершения цепи. Если вы не производите про-

верку на наличие ошибки, назначьте катушке «пустой» операнд, например, бит временных локальных данных.

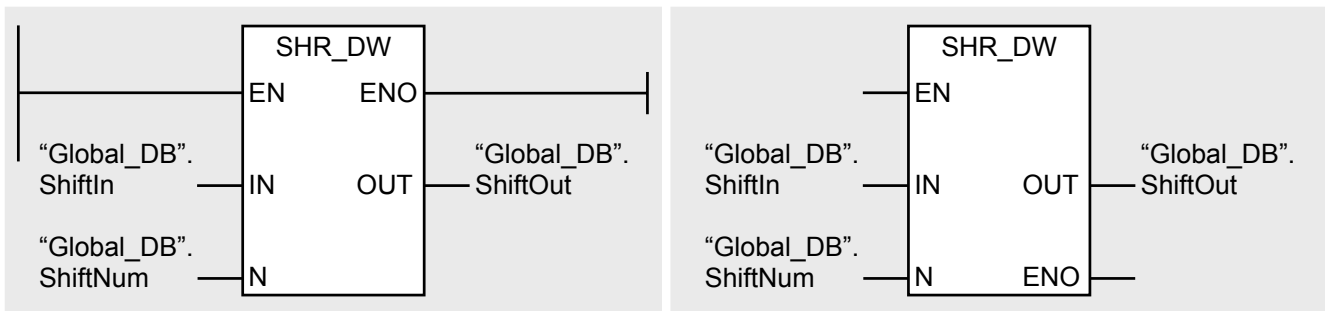
Вы можете соединить блочные элементы сдвига последовательно. Если выход ENO предыдущего блочного элемента соединен с входом EN следующего, то последний обрабатывается всегда. Если вы хотите использовать результат из предыдущего блочного элемента в качестве входного значения для следующего блока, то переменные из области временных локальных данных предоставляют удобные промежуточные буферы.

Если вы компонуете несколько блочных элементов сдвига в одной цепи (параллельно левой питающей направляющей и в дальнейшем последовательно), то сначала обрабатываются блочные элементы самой верхней ветви слева направо, затем элементы второй ветви слева направо и так далее.

Сдвиг переменных размером в слово Значение в слове памяти MW 130 сдвигается на 4 позиции влево и сохраняется в слове памяти MW 132.



Сдвиг переменных (двойное слово) Значение переменной «ShiftIn» сдвигается вправо на количество позиций, задаваемое в «ShiftNum», полученное значение записывается в переменную «ShiftOut».



Сдвиг со знаком Переменная #Act_value сдвигается на 2 позиции вправо со знаком, результат затем пересылается в переменную #Indicator.

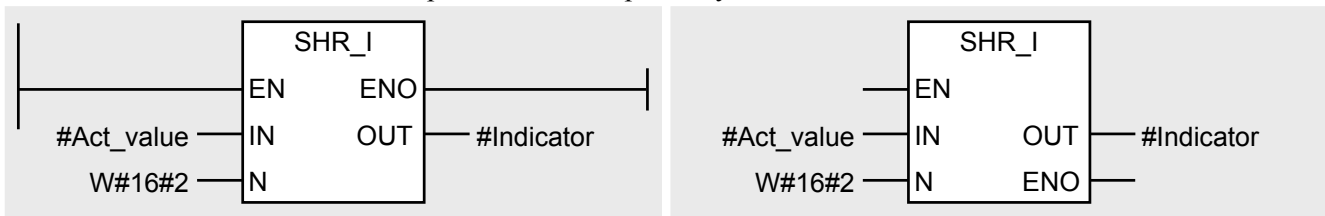


Рисунок 13.1 Примеры функций сдвига

Библиотека «LAD_Book» на дискете, прилагаемой к книге, содержит примеры представления и группировки функций сдвига (FB 113 в программе «Digital Functions», «Числовые функции»).

Функция сдвига в цепи (FBD)

EN и ENO могут быть не назначены. Если требуется обработать блочный элемент сдвига в зависимости от определенных условий, то можно скомпоновать бинарные логические операции перед входом EN. Вы можете соединить выход ENO с бинарными входами других функций. Например, можно соединить блочные элементы сдвига последовательно, таким образом, выход ENO предыдущего блочного элемента будет подключен к входу EN следующего. Если вы хотите использовать результат вычислений предыдущего блочного элемента в качестве входного значения для последующего элемента, то переменные из области временных локальных данных предоставляют удобные промежуточные буферы.

Библиотека «FBD_Book» на дискете, прилагаемой к книге, содержит примеры представления и группировки функций сдвига (FB 113 в программе «Digital Functions», «Числовые функции»).

13.2 Сдвиг

Сдвиг влево переменной размером в слово

Функция сдвига SHL_W сдвигает содержимое переменной типа WORD на входе IN побитно влево на количество позиций, определяемое числом сдвига на входе N. Битовые позиции, освобожденные сдвигом, заполняются нулями. Переменная типа WORD на выходе OUT содержит результат.

Число сдвига на входе N определяет число битовых разрядов, на которое содержимое сдвигается. Оно может быть константой или переменной. Если число сдвига равно 0, то функция не выполняется; если больше 15, то выходная переменная после выполнения функции SHL_W содержит ноль.

Сдвиг влево переменной размером в двойное слово

Функция сдвига SHL_DW сдвигает содержимое переменной типа DWORD на входе IN побитно влево на количество разрядов, определяемое числом сдвига на входе N. Битовые позиции, освобожденные сдвигом, заполняются нулями. Переменная типа DWORD на выходе OUT содержит результат.

Число сдвига на входе N определяет число битовых позиций, на которое содержимое сдвигается. Оно может быть константой или переменной. Если число сдвига равно 0, то функция не выполняется; если больше 31, то выходная переменная после выполнения функции SHL_DW содержит ноль.

Сдвиг вправо переменной размером в слово

Функция сдвига SHR_W сдвигает содержимое переменной типа WORD на входе IN побитно вправо на количество позиций, определяемое числом сдвига на входе N. Битовые позиции, освобожденные сдвигом, заполняются нулями. Переменная типа WORD на выходе OUT содержит результат.

Число сдвига на входе N определяет число битовых разрядов, на которое содержимое сдвигается. Оно может быть константой или переменной. Если число сдвига равно 0, то функция не выполняется; если больше 15, то выходная переменная после выполнения функции SHR_W содержит ноль.

Сдвиг вправо переменной размером в двойное слово

Функция сдвига SHR_DW сдвигает содержимое переменной типа DWORD на входе IN побитно вправо на количество разрядов, определяемое числом сдвига на входе N. Битовые позиции, освобожденные сдвигом, заполняются нулями. Переменная типа DWORD на выходе OUT содержит результат.

Число сдвига на входе N определяет число битовых позиций, на которое содержимое сдвигается. Оно может быть константой или переменной. Если число сдвига равно 0, то функция не выполняется; если больше 31, то выходная переменная после выполнения функции SHR_DW содержит ноль.

Сдвиг переменной размером в слово со знаком

Функция SHR_I сдвигает содержимое переменной типа INT на входе IN побитно вправо на количество позиций, определяемое числом сдвига на входе N. Битовые разряды, освобожденные при сдвиге, заполняются сигнальным состоянием бита 15 (который является знаком числа INT), то есть значением «0», если число положительное, и значением «1», если число отрицательное. На выходе OUT переменная типа INT содержит результат.

Число сдвига на входе N определяет число битовых позиций, на которое содержимое сдвигается. Оно может быть константой или переменной. Если число сдвига равно 0, то функция не выполняется; если больше 15, то выходная переменная после выполнения функции SHR_I содержит ноль.

Сдвиг вправо числа в формате данных INT эквивалентен делению на степень числа 2. Показатель степени является числом сдвига. Результат такого деления соответствует целому числу, округленному в меньшую сторону.

Сдвиг переменной размером в двойное слово со знаком

Функция SHR_DI сдвигает содержимое переменной типа DINT на входе IN побитно вправо на количество позиций, определяемое числом сдвига на входе N. Битовые разряды, освобожденные при сдвиге, заполняются сигнальным состоянием бита 31 (который является знаком числа DINT), то есть значением «0», если число положительное, и значением «1», если число отрицательное. На выходе OUT в переменной типа DINT содержится результат.

Число сдвига на входе N определяет число битовых разрядов, на которое содержимое сдвигается. Оно может быть константой или переменной. Если число сдвига равно 0, то функция не выполняется; если больше 31, то выходная переменная после выполнения функции SHR_DI содержит ноль.

Сдвиг вправо числа в формате данных DINT эквивалентен делению на степень числа 2. Показатель степени является числом сдвига. Результат такого деления соответствует целому числу, округленному в меньшую сторону.

13.3 Циклический сдвиг

Циклический сдвиг переменной (двойное слово) влево

Функция сдвига ROL_DW сдвигает влево содержимое переменной типа DWORD на входе IN побитно на количество позиций, определяемое числом сдвига на входе N. Высвобождаемые при сдвиге битовые позиции заполняются вытесненными битовыми разрядами. Результат содержится в переменной типа DWORD на выходе OUT.

Число сдвига на входе N определяет количество битовых позиций, на которое должно быть сдвинуто содержимое. Это может быть константа или переменная. Если число сдвига равно 0, то функция не выполняется. Если оно равно 32, то содержимое входной переменной сохраняется, и устанавливаются биты состояния. Если число сдвига равно 33, осуществляется сдвиг одного разряда. Если оно равно 34, сдвигается два разряда и так далее (сдвиг выполняется по модулю 32).

Циклический сдвиг переменной (двойное слово) вправо

Функция сдвига ROL_DW сдвигает вправо содержимое переменной типа DWORD на входе IN побитно на количество позиций, определяемое числом сдвига на входе N. Высвобождаемые при сдвиге битовые позиции заполняются вытесненными битовыми разрядами. Результат содержится в переменной типа DWORD на выходе OUT.

Число сдвига на входе N определяет количество битовых позиций, на которое должно быть сдвинуто содержимое. Это может быть константа или переменная. Если число сдвига равно 0, то функция не выполняется. Если оно равно 32, то содержимое входной переменной сохраняется, и устанавливаются биты состояния. Если число сдвига равно 33, осуществляется сдвиг одного разряда. Если число сдвига 34, сдвигается два разряда и так далее (сдвиг выполняется по модулю 32).

Содержание главы 14

<u>14</u>	<u>Побитовые логические операции</u>	4
<u>14.1</u>	<u>Выполнение побитовой логической операции</u>	4
<u>14.2</u>	<u>Описание побитовых логических операций</u>	8

14 Побитовые логические операции

Побитовые логические операции (числовая логика) комбинируют побитно значения двух переменных в соответствии с операциями AND, OR и исключающего OR. Логическая операция может применяться к словам или двойным словам. Доступные логические операции со словами приведены в таблице 14.1.

Таблица 14.1 Обзор логических операций со словами

Логическая операция	С переменной размером в слово	С переменной размером в двойное слово
AND (И)	WAND_W	WAND_DW
OR (ИЛИ)	WOR_W	WOR_DW
Исключающее OR (ИЛИ)	WXOR_W	WXOR_DW

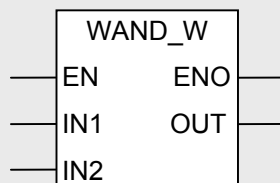
14.1 Выполнение побитовой логической операции

Представление

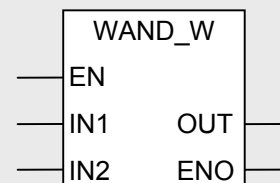
Кроме разрешающего входа (enable input) EN и разрешающего выхода (enable output) ENO блочный элемент побитовой логической операции оснащен двумя входами IN1 и IN2 и одним выходом OUT. «Заголовок» блочного элемента идентифицирует выполняемую логическую операцию (например, WAND_W означает объединение слов по AND).

Блочный элемент побитовой логической операции (в примере: объединение по AND)

Представление LAD:



Представление FBD:



Комбинируемые значения подаются на входы IN1 и IN2, результат операции находится на выходе OUT. Входы и выход имеют различные типы данных в зависимости от операции: WORD для операции со словами (16 бит) и DWORD для операций с двойными словами (32 бита). Применяемые переменные должны быть того же типа данных, что и входы или выход.

Описание битов различных форматов данных содержится в параграфе 3.5.4 «Простые типы данных».

Функция

Побитовая логическая операция выполняется, когда на разрешающем входе присутствует «1» (когда через вход EN протекает ток). Если выполнение операции не допускается (EN сброшен в «0»), то операция не производится, и ENO также имеет значение «0».

ЕСЛИ EN == "1" или не используется	
ТО	ИНАЧЕ
OUT := IN1 Wlog IN2	ENO := "0"
ENO := "1"	

Wlog – побитовая логическая операция

Если главное реле управления (MCR) активировано, то при исполнении побитовой логической операции (EN = «1») выход OUT установлен в ноль. MCR на выход ENO не воздействует.

Побитовая логическая операция генерирует результат побитно. Бит 0 входа IN1 комбинируется с битом 0 входа IN2, и результат сохраняется в бите 0 выхода OUT. То же самое происходит с битами 1, 2 и так далее до бита 15 или 31 соответственно. В таблице 14.2 показано формирование результата для отдельного бита.

В главе 15 «Биты состояния» объясняется, как побитные логические операции управляют битами состояния.

Таблица 14.2 Формирование результата побитовой логической операций

Содержимое входа IN1	0	0	1	1
Содержимое входа IN2	0	1	0	1
Результат операции AND	0	0	0	1
Результат операции OR	0	1	1	1
Результат операции исключающего OR	0	1	1	0

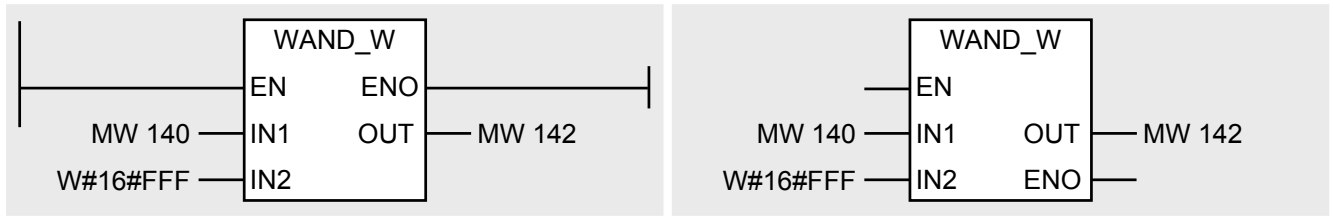
Примеры

Рисунок 14.1 показывает по одному примеру для каждой побитовой логической операции.

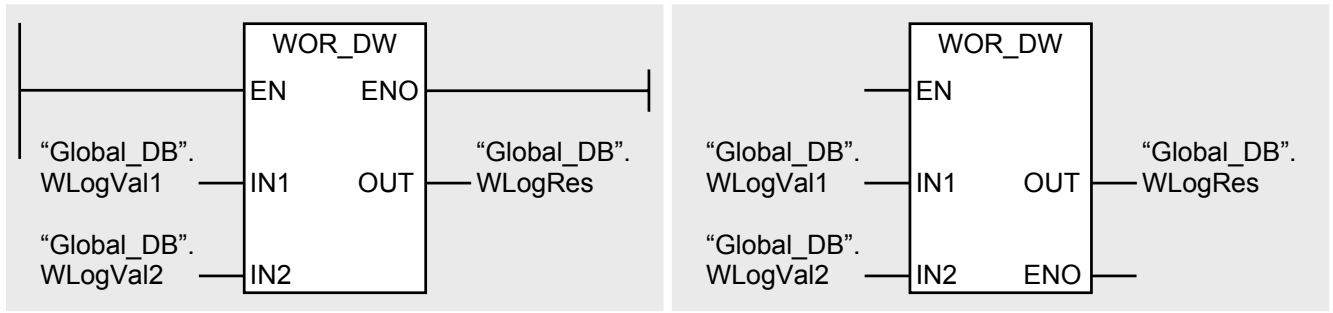
При пошаговом программировании побитовые логические операции вы можете найти в каталоге программных элементов (Program Element Catalog), вызываемого командой меню View → Catalog (Вид → Каталог) [Ctrl + K] или Insert → Program Elements (Вставка → Программные элементы), в разделе «Word Logic» («Побитовая логика»).

Операция AND

4 старших бита в слове памяти MW 140 устанавливаются в «0», и результат сохраняется в слове памяти MW 142.

**Операция OR**

Переменные «WLogVal1» и «WLogVal2» побитно комбинируются в соответствии с операцией OR, и результат записывается в «WLogRes».

**Операция Исключающее OR**

Значение, генерируемое объединением по исключающему OR двух переменных #Input и #Mask, сохраняется в переменной #Store.

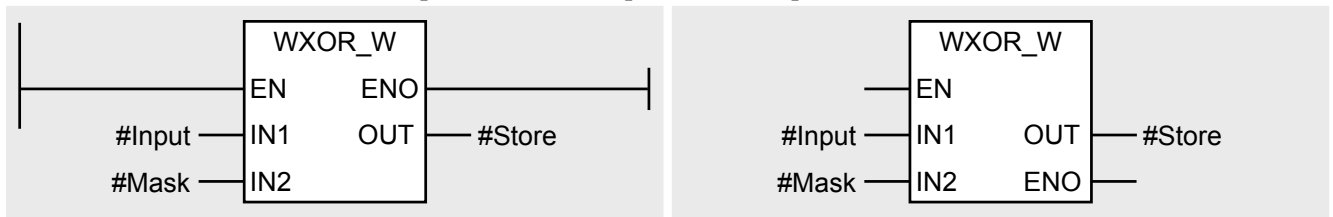


Рисунок 14.1 Примеры побитовых логических операций

Побитовые логические операции в цепи (LAD)

Вы можете соединить контакты перед входом EN и после выхода ENO последовательно и параллельно.

Сам блочный элемент побитовой логической операции может быть помещен после Т-ветви или вставлен в ветвь, соединенную непосредственно с левой направляющей (шиной питания). В этой ветви также могут быть контакты перед входом IN, и она не может быть самой верхней ветвью.

Прямое подключение к левой питающей направляющей позволяет вам соединить блочные элементы побитовой логики параллельно. Когда блочные элементы подключаются параллельно, требуется катушка для завершения цепи. Если вы не производите проверку на наличие ошибки, назначьте катушке «пустой» операнд, например, бит временных локальных данных.

Можно соединить блочные элементы побитовой логической операции последовательно. Если выход ENO предыдущего блочного элемента соединен с входом EN следующего, то последний обработается всегда. Если вы хотите использовать ре-

зультат из предыдущего блочного элемента в качестве входного значения для следующего блока, то переменные из области временных локальных данных предоставляют удобные промежуточные буферы.

Если вы компонуете несколько блочных элементов побитовой логики в одной цепи (параллельно левой питающей направляющей и в дальнейшем последовательно), то сначала обрабатываются блочные элементы самой верхней ветви слева направо, затем элементы второй ветви слева направо и так далее.

Библиотека «LAD_Book» на дискете, прилагаемой к книге, содержит примеры представления и группировки операций побитовой логики (FB 114 в программе «Digital Functions», «Числовые функции»).

Побитовые операции в логической схеме (FBD)

EN и ENO могут быть не назначены. Если требуется обработать блочный элемент побитовой логической операции в зависимости от определенных условий, то можно скомпоновать бинарные логические операции перед входом EN. Вы можете соединить выход ENO с бинарными входами других функций. Например, можно соединить блочные элементы побитовой логики последовательно, таким образом, выход ENO предыдущего блочного элемента будет подключен к входу EN следующего. Если вы хотите использовать результат вычислений предыдущего блочного элемента в качестве входного значения для последующего элемента, то переменные из области временных локальных данных предоставляют удобные промежуточные буферы.

Библиотека «FBD_Book» на дискете, прилагаемой к книге, содержит примеры представления и группировки побитовых логических операций (FB 114 в программе «Digital Functions», «Числовые функции»).

14.2 Описание побитовых логических операций

Операция AND

AND комбинирует отдельные биты значения на входе IN1 с соответствующими битами значения на входе IN2 в соответствии с правилами объединения по AND. Бит в результирующем слове OUT будет иметь значение «1», если соответствующие биты в обоих комбинируемых значениях равны «1».

Так как биты, равные «0» на входе IN2, также устанавливают соответствующие биты результата в «0» в независимости от значения этих битов на входе IN1, мы называем эти биты «маскированными». Маскирование – основное применение (числовой) операции AND.

Операция OR

OR комбинирует отдельные биты значения на входе IN1 с соответствующими битами значения на входе IN2 в соответствии с правилами объединения по OR. Бит в результирующем слове OUT будет иметь значение «0», если соответствующие биты в обоих комбинируемых значениях равны «0».

Так как биты, равные «1» на входе IN2, также устанавливают соответствующие биты результата в «1» независимо от значения этих битов на входе IN1, мы называем эти биты «маскированными». Маскирование – основное применение (числовой) операции OR.

Операция Исключающее OR

Исключающее OR комбинирует отдельные биты значения на входе IN1 с соответствующими битами значения на входе IN2 в соответствии с правилами объединения по исключающему OR. Бит в результирующем слове OUT будет иметь значение «1», если соответствующий бит только одного из двух комбинируемых значений равен «1». Если бит на входе IN2 равен «1», то соответствующий бит результата будет иметь обратное состояние по отношению к биту, расположенному в той же позиции на входе IN1.

В результате равны «1» будут только биты IN1 и IN2, имеющие противоположные сигнальные состояния до числовой операции исключающего OR. Основное использование (числовой) операции исключающего OR – обнаружение битов с противоположными сигнальными состояниями или «инвертирование» сигнальных состояний отдельных битов.

Управление программным потоком

LAD и FBD предоставляют вам инструментарий для управления программным потоком. Вы можете прервать линейное исполнение программы внутри блока или структурировать программу с использованием программируемых вызовов блоков. Можно воздействовать на выполнение программы в зависимости от значений, полученных во время исполнения, параметров процесса или в соответствии с состоянием управляемой системы (предприятия).

Биты состояния предоставляют информацию о результате арифметической или математической функции и об ошибках (таких, как выход за границы допустимого диапазона во время вычисления). Вы можете использовать совместно (объединить) сигнальные состояния битов состояния непосредственно в вашей программе с помощью контактов.

Функции перехода вы можете использовать для разветвления как безусловного, так и в зависимости от RLO (результата логической операции).

Еще один метод воздействия на исполнение программы предоставляет **главное реле управления** (Master Control Relay, MCR). Изначально разработанный для управления релейными контакторами этот метод предлагается языками LAD и FBD в программной версии.

Для структурирования программы вы можете использовать **функции блоков**. Функции и функциональные блоки можно использовать снова и снова, определяя **параметры блоков**.

Глава 19 «Параметры блоков» содержит примеры, показанные в главе 5 «Функции для работы с памятью» и главе 8 «Счетчики». В этот раз они запрограммированы как функциональные блоки с параметрами блоков. Эти функциональные блоки позже вызываются также в примере «Feed» («Устройство подачи») как локальные экземпляры.

15 Биты состояния

Биты состояния RLO, BR, CC0, CC1 и переполнение (overflow); установка и оценка битов состояния; использование бинарного результата; EN/ENO

16 Функции перехода

Безусловный переход; переход в зависимости от RLO

17 Главное реле управления

MCR-зависимость; диапазон MCR; зона MCR

18 Функции блоков

Типы блоков, вызов блока, конец блока; статические локальные данные (локальные данные состояния); регистр блока данных, использование операндов данных; управление блоками данных

19 Параметры блоков

Объявление параметров; формальные параметры, фактические параметры; передача параметров в вызываемые боки; примеры: транспортер конвейера, счетчик деталей и подача

Содержание главы 15

<u>15</u>	<u>Биты состояния</u>	4
<u>15.1</u>	<u>Описание битов состояния</u>	4
<u>15.2</u>	<u>Установка битов состояния</u>	7
<u>15.3</u>	<u>Считывание битов состояния</u>	11
<u>15.4</u>	<u>Использование бинарного результата</u>	13
<u>15.4.1</u>	<u>Установка бинарного результата BR</u>	13
<u>15.4.2</u>	<u>Главная цепь, механизм EN/ENO</u>	14
<u>15.4.3</u>	<u>ENO в случае блоков, написанных пользователем</u>	15

15 Биты состояния

Биты состояния представляют собой двоичные «флаги» (кодовые биты состояния). CPU использует их для управления операциями бинарной логики и устанавливает их во время цифровой обработки. Вы можете считать эти биты состояния или воздействовать на определенные биты. Биты состояния объединены в слово, слово состояния. Однако, вы не можете получить доступ к этому слову состояния посредством LAD или FBD.

15.1 Описание битов состояния

Таблица 15.1 содержит доступные биты состояния. CPU использует бинарные флаги для управления бинарными функциями; цифровые флаги индицируют в основном результаты арифметических и математических функций.

Таблица 15.1 Биты состояния

Бинарные флаги	
/FC	Первичный опрос / First Check
RLO	Результат логической операции / Result of logic operation
STA	Состояние / Status
OR	Бит состояния OR / Status bit OR
BR	Бинарный результат / Binary result
Цифровые флаги	
OS	Переполнение с запоминанием / Stored overflow
OV	Переполнение / Overflow
CC0	Кодовый бит условия (состояния) 0 / Condition code (status) bit 0
CC1	Кодовый бит условия (состояния) 1 / Condition code (status) bit 1

Первичный опрос (First check)

Бит состояния /FC управляет бинарной логикой внутри логической системы управления. Шаг битовой логики всегда начинается с /FC = «0» и инструкции бинарной проверки, первичного опроса. Первичный опрос устанавливает /FC = «1».

Первичный опрос соответствует в LAD первому контакту в сети и в FBD входу первой двоичной функции.

Шаг битовой логики завершается присваиванием бинарного значения (например, единичной катушки или назначения), условным переходом или сменой блока. Эти элементы устанавливают /FC = «0».

Результат логической операции (RLO)

Бит состояния RLO – это промежуточный буфер в операциях бинарной логики. В ходе начальной проверки CPU пересылает результат считывания (проверки) в RLO, комбинирует результат считывания с хранимым RLO в каждое последующее считывание и сохраняет результат, в свою очередь, в RLO.

Вы можете сохранить RLO с помощью катушки / блочного элемента SAVE в бинарном результате BR. При помощи RLO управляются функции для работы с памятью, таймеры и счетчики, а также выполняются некоторые функции перехода. RLO соответствует в LAD наличию тока в цепи (RLO = «1» – то же самое, что и «протекание тока»).

Состояние

Бит состояния STA соответствует сигнальному состоянию считанного бинарного операнда. В случае функций для работы с памятью значение STA то же, что и записанное значение, или (если операции записи не было произведено, например, если RLO = «0» или MCR активировано) STA соответствует значению адресованного (и не модифицированного) бинарного операнда.

В случае оценок фронта FP или FN значение RLO перед оценкой фронта записывается в STA. Все остальные бинарные функции устанавливают STA = «1».

Бит состояния STA не влияет на обработку функций LAD или FBD.

Бит состояния OR

Бит состояния OR хранит результат выполненной последовательной схемы или выполненного условия AND и показывает следующим обрабатываемым параллельным схемам или функции OR, что результат уже определен. Все другие бинарные функции сбрасывают бит состояния OR.

Переполнение

Бит состояния OV индицирует выход из допустимого диапазона или использование недействительных чисел REAL. На бит состояния OV влияют следующие функции: арифметические, математические функции, некоторые функции преобразования, функции сравнения типа REAL.

Бит состояния OV можно считать непосредственно.

Переполнение с запоминанием

Бит состояния OS хранит установленный бит состояния OV. Всякий раз, когда CPU устанавливает бит состояния OV, он также устанавливает бит состояния OS. Однако, если следующая корректно выполненная операция сбрасывает OV, OS остается установленным. Это дает возможность обнаружить выход за границы диапазона или операцию с недействительным числом REAL, даже позже по ходу программы.

Бит состояния OS доступен для прямого чтения. Смена блока сбрасывает бит состояния OS.

Биты состояния CC0 и CC1 (кодовые биты условия)

Биты состояния CC0 и CC1 предоставляют информацию о результате функции сравнения, арифметической или математической функции, побитовые логические операции или о бите, выдвинутом функцией сдвига.

Вы можете непосредственно считывать комбинации CC0 и CC1 (см. ниже).

Бинарный результат

LAD и FBD используют бит состояния BR для выполнения механизма EN/ENO для блочных элементов. Вы также можете самостоятельно устанавливать, сбрасывать или считывать бит состояния BR.

15.2 Установка битов состояния

Цифровые функции воздействуют на биты состояния CC0, CC1, OV и OS, как показано в таблице 15.2. Вы можете считать эти биты состояния сразу после блочного элемента функции.

Таблица 15.1 Установка битов состояния

Вычисление с типом INT				
Результат:	CC0	CC1	OV	OS
< -32 768 (ADD_I, SUB_I)	0	1	1	1
< -32 768 (MUL_I)	1	0	1	1
от -32 768 до -1	1	0	0	-
0	0	0	0	-
от +1 до +32 767	0	1	0	-
> +32 767 (ADD_I, SUB_I)	1	0	1	1
> +32 767 (MUL_I)	0	1	1	1
32 768 (DIV_I)	0	1	1	1
(-) 65 536	0	0	1	1
Деление на ноль	1	1	1	1

Вычисление с типом DINT				
Результат:	CC0	CC1	OV	OS
< -2 147 483 648 (ADD_DI, SUB_DI)	0	1	1	1
< -2 147 483 648 (MUL_DI)	1	0	1	1
от -2 147 483 648 до -1	1	0	0	-
0	0	0	0	-
от +1 до +2 147 483 647	0	1	0	-
> +2 147 483 647 (ADD_DI, SUB_DI)	1	0	1	1
> +2 147 483 647 (MUL_DI)	0	1	1	1
2 147 483 648 (DIV_DI)	0	1	1	1
(-) 4 294 967 296	0	0	1	1
Деление на ноль (DIV_DI, MOD_DI)	1	1	1	1

Сравнение				
Результат:	CC0	CC1	OV	OS
Равно	0	0	0	-
Больше	0	1	0	-
Меньше	1	0	0	-
Недействительное число REAL	1	1	1	1

Таблица 15.1 (Продолжение)

Вычисление с типом REAL				
Результат:	CC0	CC1	OV	OS
+ нормализованное	0	1	0	-
± ненормализованное	0	0	1	1
± ноль	0	0	0	-
– нормализованное	1	0	0	-
+ бесконечность (деление на ноль)	0	1	1	1
– бесконечность (деление на ноль)	1	0	1	1
± действительное число REAL	1	1	1	1

Преобразование NEG_I				
Результат:	CC0	CC1	OV	OS
от +1 до +32 767	0	1	0	-
0	0	0	0	-
от –1 до –32 767	1	0	0	-
(–) 32 768	1	0	1	1

Преобразование NEG_D				
Результат:	CC0	CC1	OV	OS
от +1 до +2 147 483 647	0	1	0	-
0	0	0	0	-
от –1 до –2 147 483 647	1	0	0	-
(–) 2 147 483 648	1	0	1	1

Функция сдвига				
Выдвинутый бит:	CC0	CC1	OV	OS
«0»	0	0	0	-
«1»	0	1	0	-
с числом сдвига 0	-	-	-	-

Побитовые логические операции				
Результат:	CC0	CC1	OV	OS
ноль	0	0	0	-
не ноль	0	1	0	-

Биты состояния в вычислениях с типами INT и DINT

Арифметические функции с форматами данных INT и DINT устанавливают все цифровые биты состояния. Нулевой результат устанавливает CC0 и CC1 в «0». Биты CC0 = «0» и CC1 = «1» указывают на положительный результат, CC0 = «1» и CC1 = «0» сообщают об отрицательном результате. Выход из диапазона допустимых значений устанавливает OV и OS (обратите внимание на другое значение CC0 и CC1 в случае переполнения). Все цифровые биты состояния, установленные в «1», свидетельствуют о делении на ноль.

Биты состояния в вычислениях с типом REAL

Арифметические функции с форматом данных REAL и математические функции устанавливают все цифровые биты состояния. Нулевой результат устанавливает CC0 и CC1 в «0». CC0 = «0» и CC1 = «1» указывают на положительный результат, CC0 = «1» и CC1 = «0» сообщают об отрицательном результате. Выход за границы допустимого диапазона устанавливает OV и OS (обратите внимание на другое значение CC0 и CC1 в случае переполнения). Недействительное число REAL указывается в виде установленных в «1» всех цифровых битов.

Число типа REAL называется «ненормализованным», если оно представляется с уменьшенной точностью. В таком случае экспонента равна нулю; абсолютное значение ненормализованного числа REAL меньше $1.175\,494 \times 10^{-38}$. CPU S7-300 рассматривают ненормализованные числа REAL, как нуль (см. также параграф 3.5.4 «Простые типы данных»).

Биты состояния для функций преобразования

Из функций преобразования на все цифровые биты состояния влияет дополнение до двух (дополнительный код). Кроме того, следующие функции преобразования устанавливают биты состояния OV и OS в случае ошибки (выход за границы допустимого диапазона или недействительное число REAL):

- I_BCD и DI_BCD:
Преобразование INT и DINT в BCD;
- CEIL, FLOOR, ROUND, TRUNC:
Преобразование REAL в DINT.

Биты состояния для функций сравнения

Функции сравнения устанавливают биты состояния CC0 и CC1. Флаги устанавливаются независимо от выполняемой функции сравнения.

Биты состояния для функций сдвига

В случае функций сдвига сигнальное состояние последнего выдвигаемого бита пересылается в бит состояния *CC1*. *CC0* и *OV* сбрасываются.

Биты состояния для побитовых логических операций

Если результат побитовой логической операции нулевой (все биты обнулены), то *CC1* сбрасывается; если хотя бы один бит результата равен «1», то *CC1* устанавливается. *CC0* и *OV* сбрасываются.

15.3 Считывание битов состояния

LAD: Для считывания цифровых битов состояния и бинарного результата вы можете использовать нормально разомкнутый (NO) и нормально замкнутый (NC) контакт. Рисунок 15.1 показывает чтение с помощью нормально разомкнутого контакта. Считывание с применением нормально замкнутого контакта возвращает инвертированный результат считывания. Вы можете оперировать NO- и NC-контактами для оценки битов состояния точно так же, как и «нормальными» контактами. Библиотека «LAD_Book» на прилагаемой к книге дискете содержит примеры оценки битов состояния (FB 115 в программе «Program Flow Control», «Управление программным потоком»).

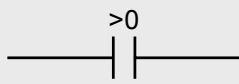
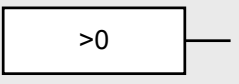
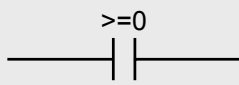
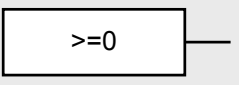
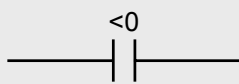
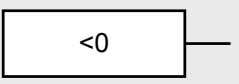
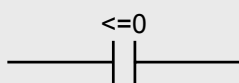
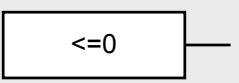

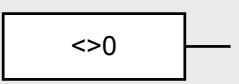
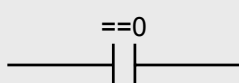
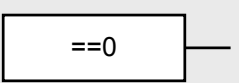

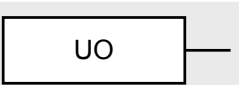





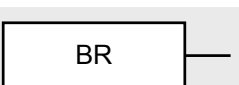
Представление LAD	Представление FBD	Ток течет или считывание успешно, когда
		Результат больше нуля [(CC0 = 0) & (CC1 = 1)]
		Результат больше или равен нулю [(CC0 = 0)]
		Результат меньше нуля [(CC0 = 1) & (CC1 = 0)]
		Результат меньше или равен нулю [(CC1 = 0)]
		Результат не равен нулю [(CC0 = 0) & (CC1 = 1) v (CC0 = 1) & (CC1 = 0)]
		Результат равен нулю [(CC0 = 0) & (CC1 = 0)]
		Результат недействителен (неупорядоченный) [(CC0 = 1) & (CC1 = 1)]
		Переполнение диапазона чисел [OV = 1]
		Сохраненное переполнение [OS = 1]
		Бинарный результат [BR = 1]

Рисунок 15.1 Чтение битов состояния

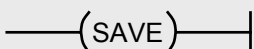
FBD: Возможно прямое или инвертированное считывание цифровых битов состояния и бинарного результата. Это подтверждается прямым чтением с ожиданием сигнального состояния «1». Считывание с ожиданием сигнального состояния «0» возвращает инвертированный результат проверки. Операции чтения для оценки битов состояния могут проводиться так же, как и «нормальные» проверки бинарных операндов. Библиотека «FBD_Book» на поставляемой с книгой дискете содержит примеры оценки битов состояния (FB 115 в программе «Program Flow Control», «Управление программным потоком»).


При пошаговом программировании вы можете найти эти элементы проверок в каталоге программных элементов (Program Element Catalog), вызываемого командой меню View → Catalog (Вид → Каталог) [Ctrl + K], или использовать пункт меню Insert → Program Elements (Вставка → Программные элементы), раздел «Status Bits» («Биты состояния»).

15.4 Использование бинарного результата

15.4.1 Установка бинарного результата BR

Присваивание бинарного результата

Представление LAD: 

Представление FBD: 

Катушка SAVE в цепи (LAD)

Вы можете сохранить RLO в бинарном результате с помощью катушки SAVE (Сохранить). Если в катушке SAVE течет электрический ток, то BR устанавливается, иначе он сбрасывается. Программируется катушка SAVE таким же образом, как и «единичная» катушка без бинарных операндов.

Обратите внимание на то, что катушка SAVE не завершает логическую операцию (бит состояния /FC не устанавливается в «0»). Это означает, что логическая операция, предшествующая катушке SAVE, также является предшествующей логической операцией для следующей сети.

Катушка SAVE не может быть запрограммирована совместно с Т-ветвью.

Блочный элемент SAVE в логической цепи (FBD)

При помощи блочного элемента SAVE вы можете сохранить RLO в бинарном результате. Если перед блочным элементом SAVE результат RLO равен «1», то BR устанавливается; в противном случае BR обнуляется. Блочный элемент SAVE программируется так же, как и блочный элемент присваивания без бинарного операнда.

Обратите внимание на то, что блочный элемент SAVE не завершает логическую операцию (бит состояния /FC не устанавливается в «0»). Следующая сеть, таким образом, начинается с «открытой» логической операции.

Блочный элемент SAVE после Т-ветви не допускается.

Управление бинарным результатом

LAD и FBD воздействуют на бинарный результат, чтобы управлять выходом ENO (рисунок 15.2). Если разрешающий выход ENO назначен, то его сигнальное состояние то же, что и у BR. В определенных случаях («BR соответствует функции») выполняемая функция LAD или FBD устанавливает бинарный результат в следующем виде:

- BR := «1»
для MOVE, функций сдвига и логических операций со словами;
- BR := OV
для арифметических и математических функций;
- BR := OV или «1»
для функций преобразования;
- BR := BR вызванного блока в случае вызовов блоков.

ENO подключен ?					
ДА			НЕТ		
EN подключен ?			EN подключен ?		
ДА		НЕТ	ДА		НЕТ
EN == «1» ?			EN == «1» ?		
ДА	НЕТ		ДА	НЕТ	
BR соответствует функции	BR := «0»	BR соответствует функции	BR := «1»	BR := «0»	на BR влияние не оказывается

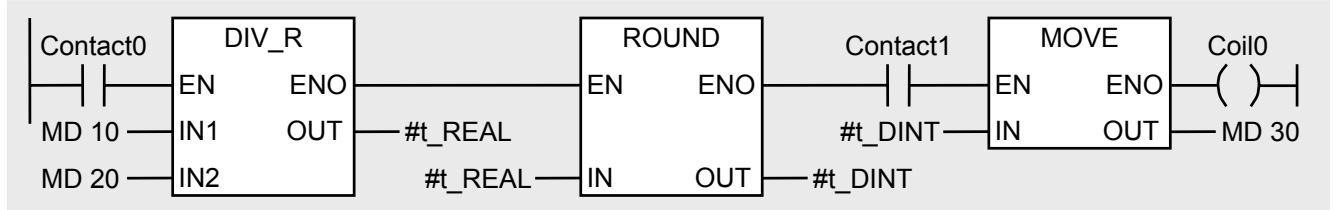
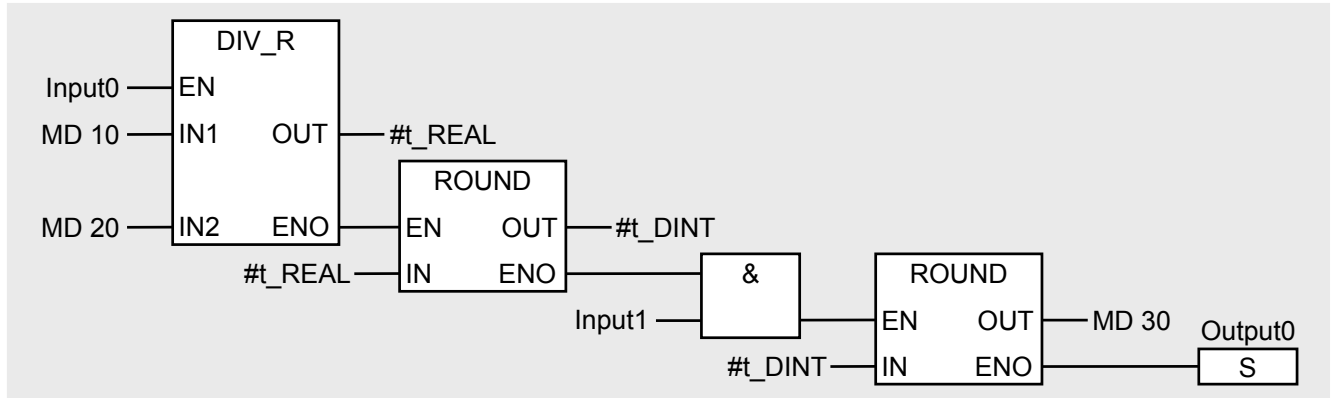
Рисунок 15.2 Общая схема установки бинарного результата

15.4.2 Главная цепь, механизм EN/ENO

В языках программирования LAD и FBD многие блочные элементы имеют разрешающий вход EN и разрешающий выход ENO. Если на разрешающем входе присутствует «1», то функция блочного элемента обрабатывается. Когда блочный элемент обработан корректно, разрешающий выход также имеет сигнальное состояние «1». Если во время обработки блочного элемента возникает ошибка (к примеру, переполнение при выполнении арифметической функции), то ENO устанавливается в «0». Если EN имеет сигнальное состояние «0», то ENO также установлен в «0».

Эти характеристики EN и ENO могут быть использованы для соединения в цепи нескольких блочных элементов с подключением разрешающего выхода к разрешающему входу следующего элемента (рисунок 15.3). Это означает, например, что вся цепь может быть «выключена» (блочные элементы не обрабатываются, если вход I 1.0 в примере имеет нулевое сигнальное состояние) или оставшаяся часть цепи не будет обработана, если один блочный элемент выдаст ошибку.

Вход EN и выход ENO не являются параметрами блока, но представляют собой последовательности операторов, которые программный редактор сам генерирует до и после всех блочных элементов (также в случае функций и функциональных блоков). Программный редактор использует бинарный результат для сохранения сигнального состояния на EN, пока блок обрабатывается, или для проверки флага ошибки блочного элемента.

Пример главной цепи (LAD)**Пример последовательного соединения блочных элементов (FBD)****Рисунок 15.3** Пример последовательного соединения EN и ENO**15.4.3 ENO в случае блоков, написанных пользователем**

Редактор программ обеспечивает вызов ваших собственных блоков с использованием разрешающего входа EN и разрешающего выхода ENO. Вы можете использовать разрешающий вход EN для условного вызова блока. Разрешающий выход ENO может использоваться, например, для сообщения о групповой ошибке (сигнальное состояние «1», если блок обработан правильно; «0», если при обработке блока возникла ошибка). Все системные блоки также сигнализируют о групповых ошибках через BR.

Вы можете управлять выходом ENO с помощью бинарного результата BR. Выход ENO имеет то же сигнальное состояние, что и BR по выходу из блока.

Например, при запуске блока BR может быть установлен в «1». Если затем во время обработки блока возникает ошибка (к примеру, если результат выходит за границы фиксированного диапазона, и дальнейшая обработка не допускается), установите с помощью катушки / блочного элемента SAVE бинарный результат в «0» и перейдите в конец блока, где будет произведен выход из него (в случае ошибки условие должно выдавать сигнальное состояние «0»). Заметьте, что катушка / блочный элемент RET устанавливает BR в «1», если вы выходите из блока посредством этой катушки / блочного элемента.

Содержание главы 16


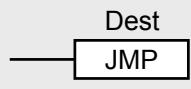
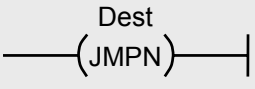
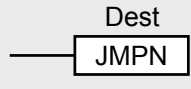


<u>16</u>	<u>Функции перехода</u>	4
16.1	Обработка функции перехода	4
16.2	Безусловный переход	6
16.3	Переход, если RLO = «1».....	7
16.4	Переход, если RLO = «0».....	8

16 Функции перехода

Функции перехода вы можете использовать для прерывания линейного потока программы и продолжения с другой точки блока. Это программное ветвление может быть выполнено безусловно или в зависимости от RLO (результата логической операции).

16.1 Обработка функции перехода

Представление

Представление LAD	Представление FBD
Переход, если RLO = «1» 	Переход, если RLO = «1» 
Переход, если RLO = «0» 	Переход, если RLO = «0» 
Место назначения, метка перехода 	Место назначения, метка перехода 

Функция перехода состоит из операции перехода в форме катушки (LAD) или блочного элемента (FBD) и метки перехода, обозначающей место в программе, с которого после перехода продолжится обработка. Метка перехода указывается над операцией перехода.

Переходы нельзя программировать совместно с T-ветвью.

Метка перехода может включать в себя до 4 символов, которые могут быть буквами, цифрами и знаками подчеркивания. Начинается она с буквы. Метка перехода в блочном элементе обозначает сеть, которая будет обрабатываться после окончания операции перехода. Блочный элемент с меткой перехода должен располагаться в начале сети (в каталоге программных элементов – «LABEL», «Метка»).

Для пошагового программирования функции перехода можно найти в каталоге программных элементов (Program Elements Catalog), вызвав его командой View → Catalog (Вид → Каталог) [Ctrl – K], или выбрав команду Insert → Program Elements (Вставка → Программные элементы), раздел «Logic Control / Jump» («Логическое управление / Переход»).

Функция

Переход выполняется либо всегда (абсолютный или безусловный переход), либо в зависимости от результата логической операции (RLO) (условный переход). В случае перехода, зависящего от RLO, вы можете решить, при RLO, равным «1» или «0», этот переход будет выполнен.

Переходы можно выполнять как вперед (в направлении обработки программы или увеличения сетевых номеров), так и назад. Переход может осуществляться только внутри блока, то есть место назначения перехода должно быть в том же блоке, что и функция перехода. Если вы используете главное реле управления (Master Control Relay, MCR), метка перехода должна находиться в той же MCR-зоне или в MCR-области, что и функция перехода.

Место назначения перехода должно быть обозначено однозначно, то есть вы должны назначить метку перехода в блоке только один раз. Перейти к месту назначения можно из нескольких точек программы.

Программный редактор сохраняет имена меток переходов в неисполняемом разделе соответствующих блоков в среде данных устройства программирования. В пользовательской памяти CPU (в скомпилированном блоке) хранятся только размеры переходов (смещение перехода). Когда в онлайн-режиме производятся изменения программы в блоке в CPU, эти модификации всегда должны быть проведены в среде данных устройства программирования с тем, чтобы сохранить исходные имена меток. Если это обновление не осуществляется, или если блоки передаются из CPU в программатор, то неисполняемые разделы блоков будут перезаписаны или удалены. Тогда на дисплее или при печати редактор генерирует для меток переходов символы замены (M001, M002 и т.д.).

16.2 Безусловный переход

Абсолютным переходом, который всегда выполняется, является функция перехода JMP, чья катушка соединена с левой направляющей (питающей шиной) (LAD), или ее блочный элемент не имеет предшествующей логической операции (FBD). Эта функция перехода выполняется всегда, когда встречается в программе. CPU прерывает линейный поток программы и продолжает обработку в сети, обозначенной меткой перехода. Пример (рисунки 16.1 и 16.2): В сети 3 (Network 3) представлен безусловный переход к метке перехода M2. После обработки этой сети CPU продолжает выполнение программы с метки перехода M2 в сети 5 (Network 5). Чтобы обработать сеть 4 (Network 4), требуется переход из другой точки программы.

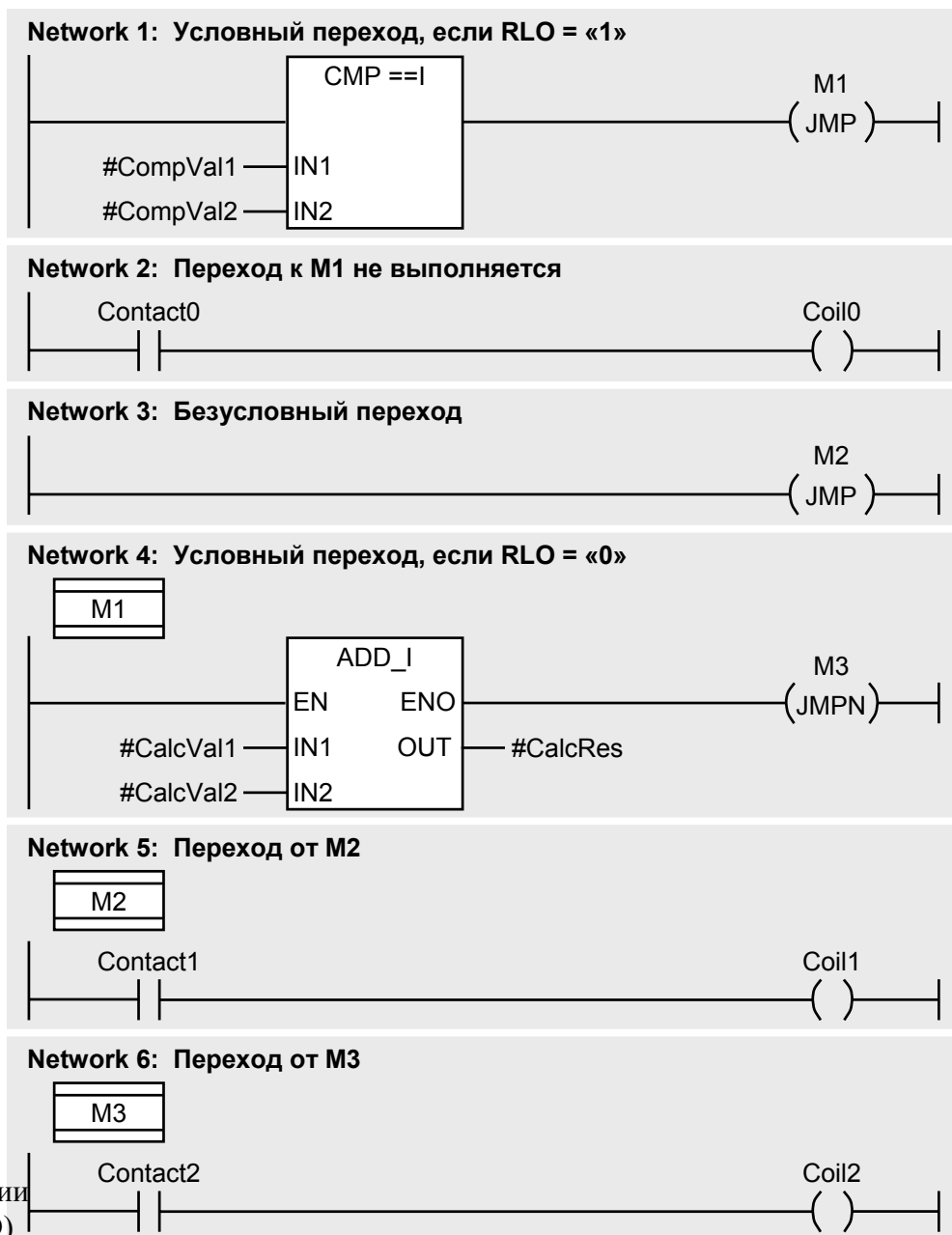


Рисунок 16.1
Пример функции
перехода (LAD)

16.3 Переход, если RLO = «1»

Условный переход при RLO = «1» выполняет функция JMP, чья катушка не имеет прямого соединения с левой направляющей (шиной питания) (LAD), или блочному элементу которой предшествует логическая операция (FBD). Логическая операция, предшествующая этой катушке, может быть выполнена в любом случае. Если RLO = «1» (в случае выполнения предшествующей логической операции), то CPU прерывает линейный поток программы и продолжает обработку в сети, обозначенной меткой перехода. Если предшествующая логическая операция не выполнена, то CPU продолжает выполнять программу в следующей сети.

Пример (рисунки 16.1 и 16.2): Если условие сравнения в сети 1 (Network 1) выполнено, то программа продолжит работу в сети 4 (Network 4). Если условие сравнения не удовлетворено, то обрабатывается следующая сеть, сеть 2 (Network 2).

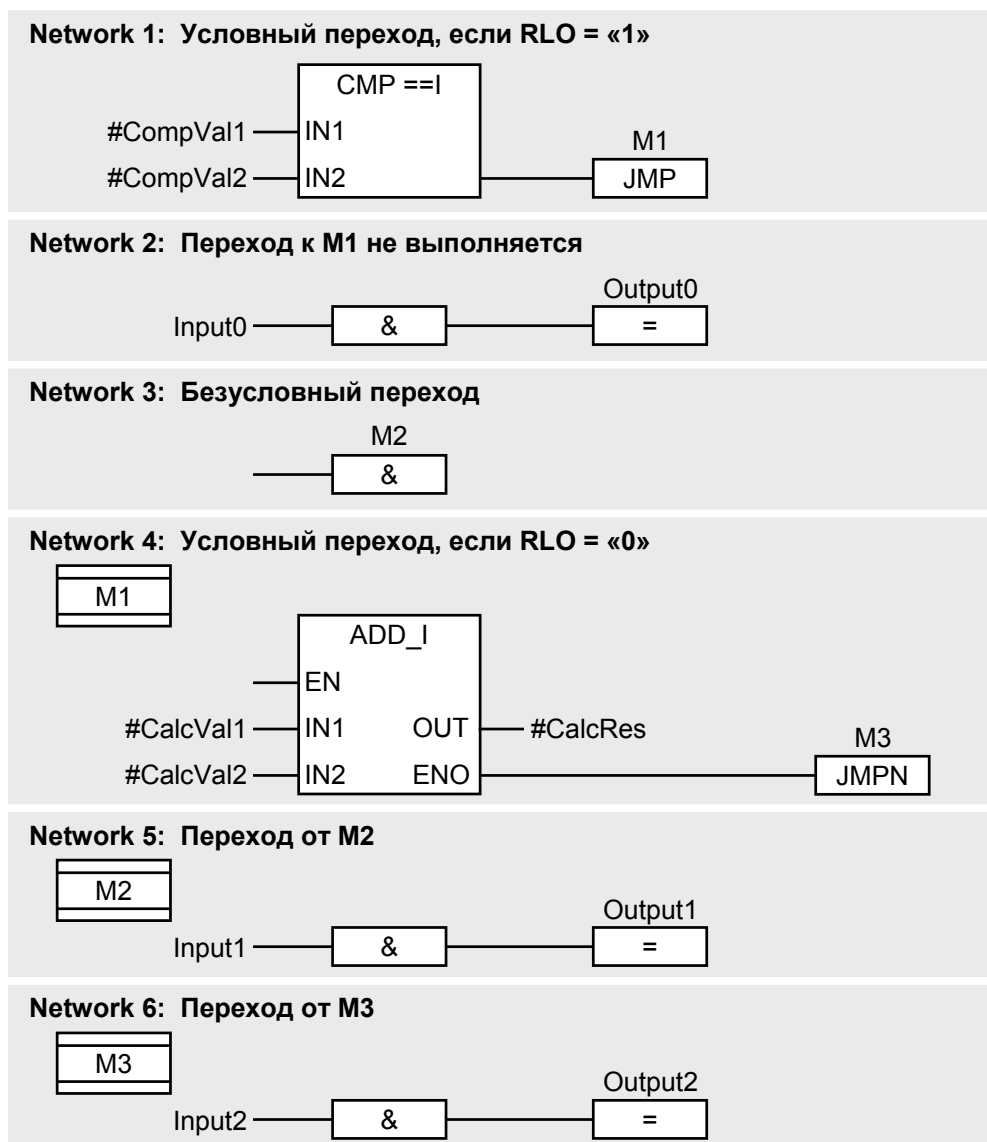


Рисунок 16.2 Пример функций перехода (FBD)

16.4 Переход, если RLO = «0»

Условный переход при RLO = «0» выполняет функция JMPN, чья катушка не соединена напрямую с левой направляющей (шиной питания) (LAD), или блочному элементу которой предшествует логическая операция (FBD). Логическая операция, предшествующая этой катушке, может быть выполнена в любом случае. Если RLO = «0» (в случае неудовлетворения предшествующей логической операции), то CPU прерывает линейный поток программы и продолжает обработку в сети, обозначенной меткой перехода. Если предшествующая логическая операция выполнена, то CPU продолжает выполнять программу в следующей сети.

Пример (рисунки 16.1 и 16.2): Если сумматор в сети 4 (Network 4) сигнализирует об ошибке, то выполняется переход к сети 6 (Network 6) (метка перехода M3). Если ошибка не возникает, то обрабатывается следующая сеть, сеть 5 (Network 5).

Содержание главы 17

<u>17</u>	<u>Главное реле управления</u>	4
<u>17.1</u>	<u>MCR-зависимость</u>	5
<u>17.2</u>	<u>MCR-область</u>	6
<u>17.3</u>	<u>MCR-зона</u>	7
<u>17.4</u>	<u>Установка и сброс I/O-битов</u>	10

17 Главное реле управления

В системах управления контактами главное реле управления (Master Control Relay, MCR) активирует или деактивирует секцию системы управления, которая состоит из одной или более цепей.

В деактивированной цепи

- отключаются все несохраняющие контакторы и
- сохраняют состояния сохраняющие контакторы.

Состояния контакторов изменить нельзя до активации MCR.

Из-за этого свойства MCR чаще используется в языке программирования LAD, хотя язык программирования FBD также предоставляет функции, требующиеся для MCR.

Представление LAD	Представление FBD
Активация MCR-области 	Активация MCR-области 
Открытие MCR-зоны 	Открытие MCR-зоны 
Закрытие MCR-зоны 	Закрытие MCR-зоны 
Деактивация MCR-области 	Деактивация MCR-области 

Обратите внимание, что отключение (снятие напряжения) с использованием «программного» главного реле управления не является заменой аварийного отключения (EMERGENCY OFF) или средств безопасности! Используйте выключение главного реле управления точно так же, как и выключение с помощью функции для работы с памятью!

Используя MCRA и MCRD, вы можете обозначить область в вашей программе, в которой будет действовать MCR-зависимость. В этой области с помощью MCR< и MCR> определяются одна или более зон, в которых MCR-зависимость может быть активирована или отключена. Можно построить вложения MCR-зон. Результат логической операции (RLO), непосредственно предшествующий MCR-зоне, включает или деактивирует MCR-зависимость в этой зоне.

17.1 MCR-зависимость

MCR-зависимость воздействует на катушки и блочные элементы. Если MCR-зависимость активирована (соответствует заблокированному главному реле управления), то

- одиночная катушка или блочный элемент присваивания и коннектор устанавливают бинарный операнд в сигнальное состояние «0» (вслед за коннектором значение RLO становится «0», то есть ток больше не течет);
- катушка или блочный элемент установки и сброса больше не оказывают влияния на сигнальное состояние бинарного операнда («замораживают» его);
- блочный элемент SR и RS больше не воздействуют на сигнальное состояние бинарного операнда («замораживают» его);
- операция пересылки записывает ноль в числовой операнд (таким образом, каждый выход блочного элемента функции числового типа данных записывает ноль в операнд или переменную).

Кроме прямого влияния на операнд RLO сбрасывается в «0» (ток не протекает) после коннектора и T-ветви.

Некоторые функции LAD и FBD используют операторы пересылки (незаметно для пользователя). Так как оператор пересылки записывает нулевое значение, если MCR-зависимость включена, выполнение соответствующей функции не гарантировано.

Чтобы избежать перехода CPU в режим STOP или произвольного поведения во время работы, вы должны исключить из MCR-зависимости следующие разделы программы:

- вызовы блоков с параметрами;
- обращения к параметрам блоков, которые имеют параметрические типы (например, BLOCK_DB);
- обращения к параметрам блоков, являющихся компонентами или элементами сложных типов данных или UDT (пользовательских типов).

Вы активируете MCR-зависимость в зоне, если RLO равен «0» непосредственно перед открытием зоны (аналогично отключению главного реле управления). Если вы открываете MCR-зону с RLO, равным «1», (главное реле управления активировано), то обработка в этой MCR-зоне происходит без MCR-зависимости. MCR-зависимость работает только внутри MCR-зоны. Для пошагового программирования функции MCR можно найти в каталоге программных элементов (Program Elements Catalog), вызвав его командой View → Catalog (Вид → Каталог) [Ctrl – K], или выбрав команду Insert → Program Elements (Вставка → Программные элементы), раздел «Program Control» («Программное управление»).

17.2 MCR-область

Для того, чтобы воспользоваться возможностями главного реле управления, определите MCR-область (MCR area) с помощью MCRA (начало) и MCRD (конец MCR-области). MCR-зависимость активна в MCR-области (но еще не включена).

Катушка / блочный элемент MCRA и катушка / блочный элемент MCRD всегда стоят одни в отдельных сетях (других элементов нет).

Если вы внутри MCR-области вызываете блок, MCR-зависимость в вызванном блоке деактивируется (рисунок 17.1). MCR-область начинается только с катушки / блочного элемента MCRA. Когда происходит выход из блока, MCR-зависимость устанавливается, как это было до вызова блока, несмотря на MCR-зависимость, с которой был осуществлен выход из вызванного блока.

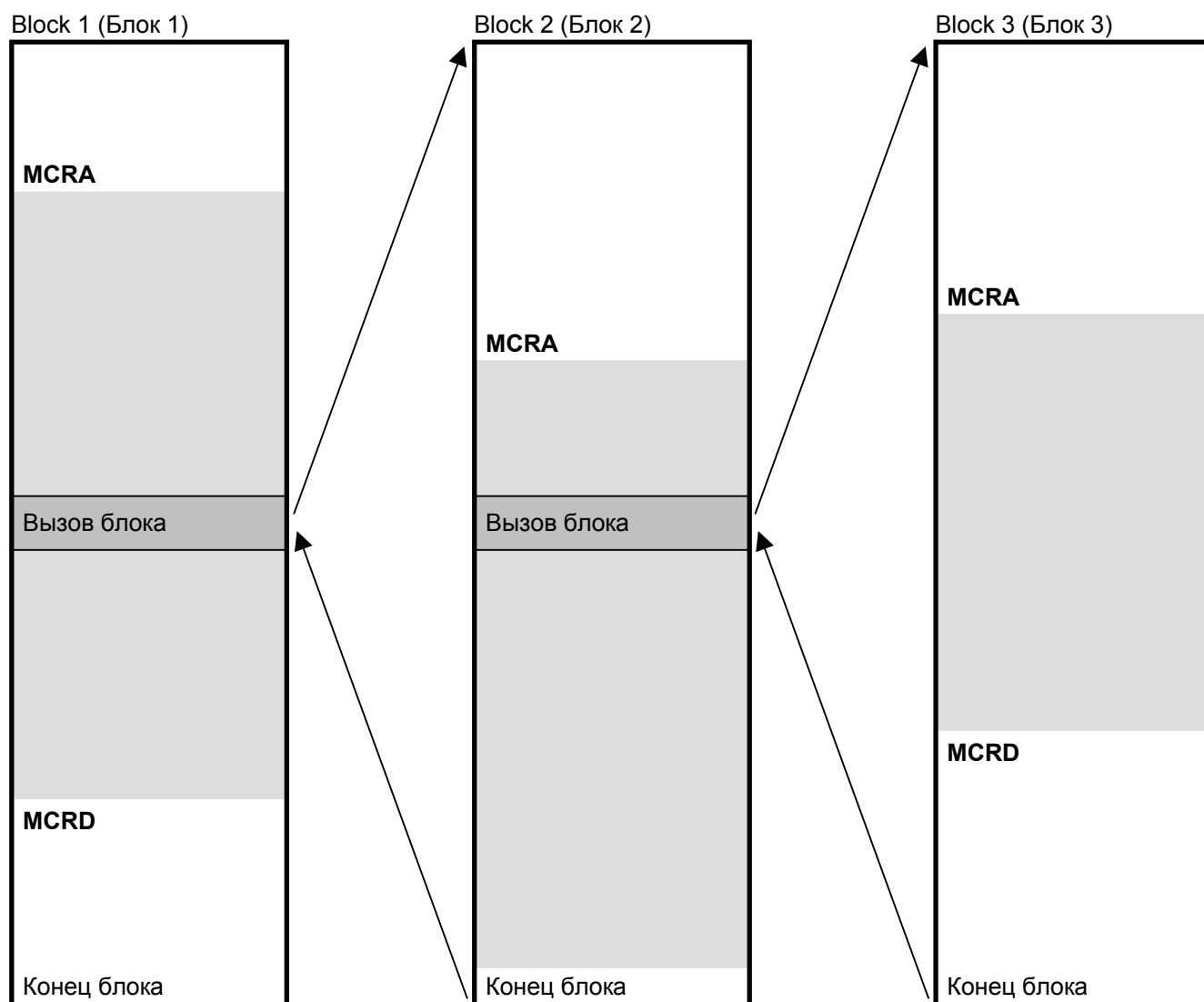


Рисунок 17.1 MCR-область в случае смены блоков

17.3 MCR-зона

LAD: MCR-зону (MCR zone) вы можете определить с помощью катушки MCR< (начало) и катушки MCR> (конец MCR-зоны). Для катушки MCR< требуется предшествующая логическая операция; катушка MCR> подключается непосредственно к левой направляющей (шине питания). Обе катушки завершают цепь. Внутри этой хоны вы можете управлять MCR-зависимостью при помощи RLO, предшествующего катушке MCR<. Если в катушке течет ток, MCR-зависимость отключена («нормальная» обработка); если ток в катушке не течет, MCR-зависимость активирована.

FBD: MCR-зона определяется при помощи блочного элемента MCR< в начале и блочного элемента MCR> в конце MCR-зоны. Блочному элементу MCR< требуется предшествующая логическая операция; блочный элемент MCR> в сети ставится один. В этой зоне вы можете управлять MCR-зависимостью, используя предшествующий блочному элементу MCR< результат RLO. Если он равен «1», MCR-зависимость деактивирована; если он равен «0», MCR-зависимость включена.

MCR-зону вы можете открыть внутри другой MCR-зоны. Глубина вложения для MCR-зон равна 8, то есть можно открыть до восьми зон перед тем, как их закрыть.

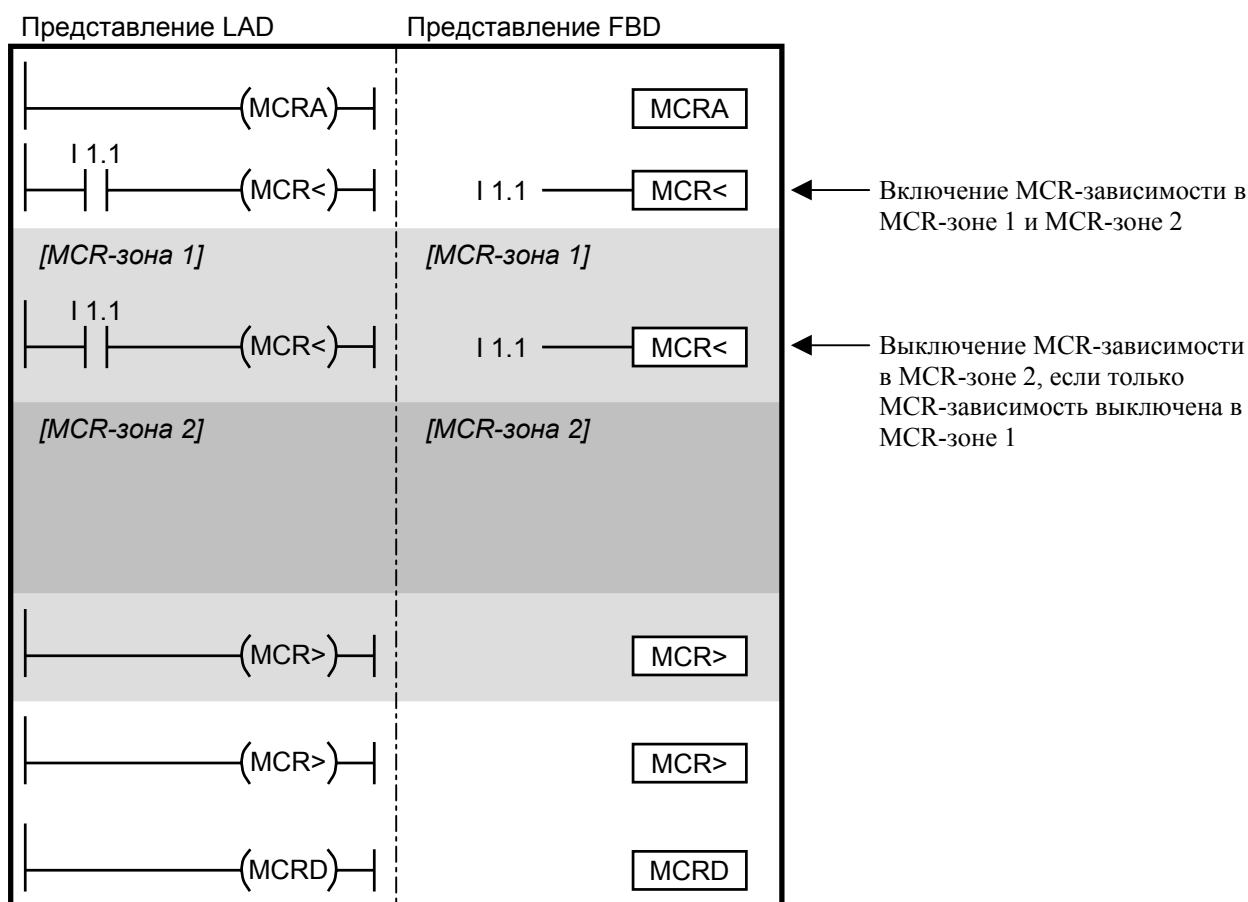


Рисунок 17.2 MCR-зависимость во вложенных MCR-зонах

Управление MCR-зависимостью вложенной MCR-зоны осуществляется с помощью RLO в момент ее открытия. Однако, если MCR-зависимость разрешена во «внешней» зоне, вы не сможете отключить MCR-зависимость в «подчиненной» MCR-зоне. Главное реле управления первой MCR-зоны управляет MCR-зависимостью во всех вложенных (рисунок 17.2).

Вызов блока внутри MCR-зоны не меняет ее глубину вложения. Программа вызванного блока все еще находится в MCR-зоне, которая была открыта, когда блок вызывался (и управляется отсюда). Тем не менее, вы должны повторно активировать MCR-зависимость в вызванном блоке путем открытия MCR-области.

На рисунке 17.3 меркеры M 10.0 и M 11.0 управляют MCR-зависимостями. При помощи меркера M 10.0 вы можете активировать MCR-зависимость в обеих зонах (значением «0») вне зависимости от сигнального состояния меркера M 11.0. Если MCR-зависимость для зоны 1 отключена с помощью M 10.0 = «1», можно управлять MCR-зависимостью зоны 2 меркером M 11.0 (таблица 17.1).

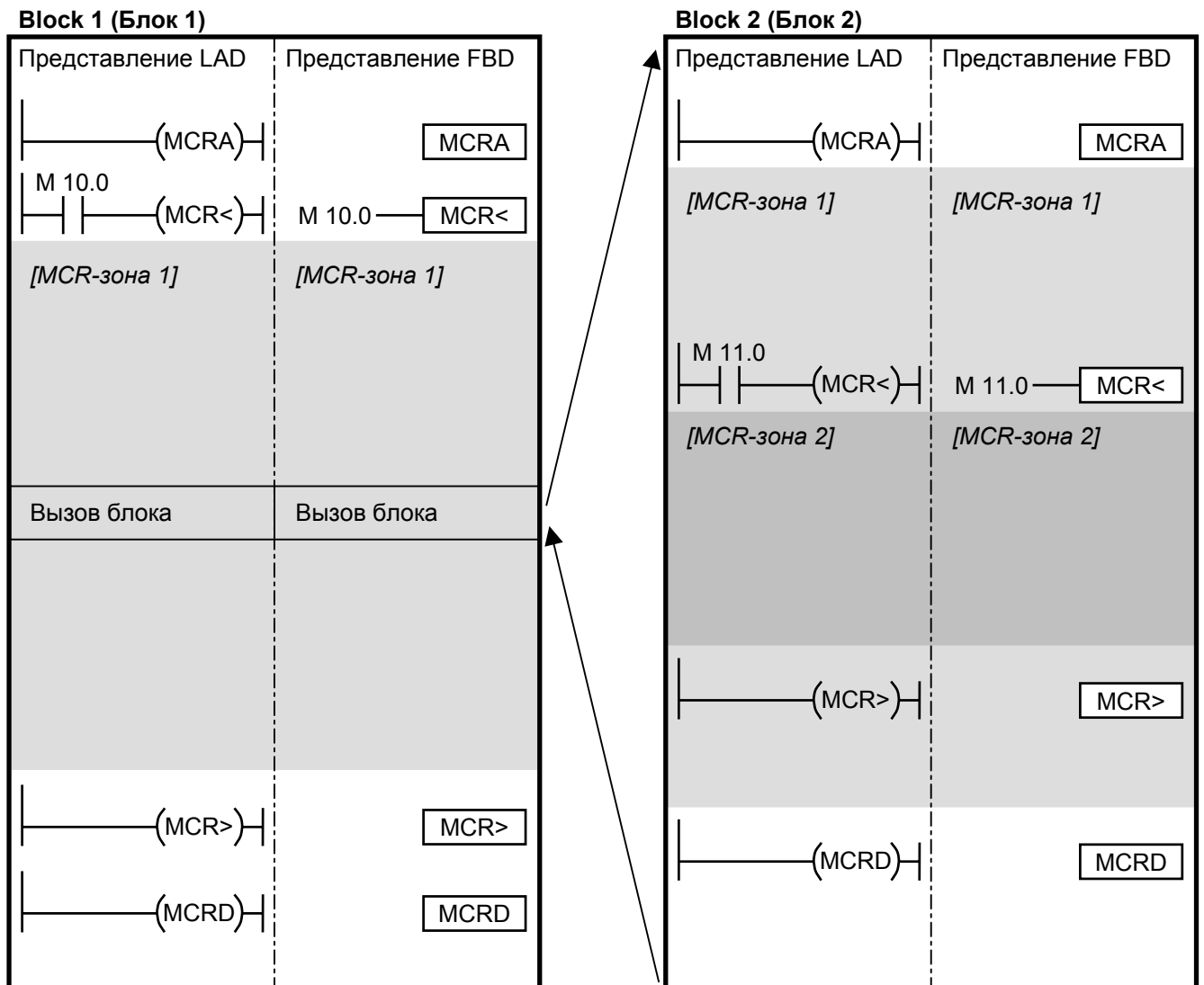


Рисунок 17.3 MCR-зоны в случае смены блоков

Таблица 17.1 MCR-зависимость в случае вложенных MCR-зон (пример)

М 10.0	М 11.0	Зона 1	Зона 2
«1»	«1»	MCR-зависимости нет	
«1»	«0»	MCR-зависимости нет	MCR-зависимость активирована
«0»	«1» или «0»	MCR-зависимость активирована	

17.4 Установка и сброс I/O-битов

Несмотря на активированную MCR-зависимость, вы можете установить или сбросить биты I/O-области (области ввода/вывода или входов/выходов), используя системные функции. Для этого необходимо, чтобы управляемые биты находились в выходной таблице образа процесса, или выходная таблица образа процесса была определена для управляемой I/O-области.

Системная функция **SFC 79 SET** предназначена для установки, а **SFC 80 RSET** – для сброса I/O-битов (таблица 17.2). Вы можете вызвать эти системные функции в MCR-зоне. Системные функции действуют при условии активной MCR-зависимости; если MCR-зависимость деактивирована, вызов этих SFC результатов не даст.

Установка и сброс I/O-битов также одновременно обновляет таблицу выходов образа процесса. Воздействие на входы/выходы (I/O) осуществляется побайтно (байт за байтом). Биты, не выбранные с помощью SFC (в первом и последнем байте) сохраняют сигнальные состояния, соответствующие текущим значениям в образе процесса.

Библиотеки «LAD_Book» и «FBD_Book» на дискете, прилагаемой к книге, содержат примеры с главным реле управления и системными функциями SFC 79 и SFC 80 (FB 117 в программе «Program Flow Control», «Управление программным потоком»).

Таблица 17.2 Параметры SFC для управления I/O-битами

SFC	Параметр	Объявление	Тип данных	Назначение, описание
79	N	INPUT	INT	Количество устанавливаемых битов
	RET_VAL	OUTPUT	INT	Информация об ошибке
	SA	OUTPUT	POINTER	Указатель на первый устанавливаемый бит
80	N	INPUT	INT	Количество сбрасываемых битов
	RET_VAL	OUTPUT	INT	Информация об ошибке
	SA	OUTPUT	POINTER	Указатель на первый сбрасываемый бит

Содержание главы 18

18	<u>Функции для работы с блоками</u>	4
18.1	<u>Функции для работы с кодовыми блоками</u>	4
18.1.1	<u>Вызовы блоков: общая информация</u>	5
18.1.2	<u>Блочный элемент вызова</u>	6
18.1.3	<u>Катушка / блочный элемент CALL</u>	9
18.1.4	<u>Функция завершения блока</u>	10
18.1.5	<u>Временные локальные данные</u>	11
18.1.6	<u>Статические локальные данные</u>	14
18.2	<u>Функции блоков для блоков данных</u>	19
18.2.1	<u>Два регистра блоков данных</u>	19
18.2.2	<u>Доступ к операндам данных</u>	20
18.2.3	<u>Открытие блока данных</u>	22
18.2.4	<u>Особые замечания по адресации данных</u>	24
18.3	<u>Системные функции для блоков данных</u>	26
18.3.1	<u>Создание блока данных</u>	26
18.3.2	<u>Удаление блока данных</u>	26
18.3.3	<u>Тестирование блоков данных</u>	27

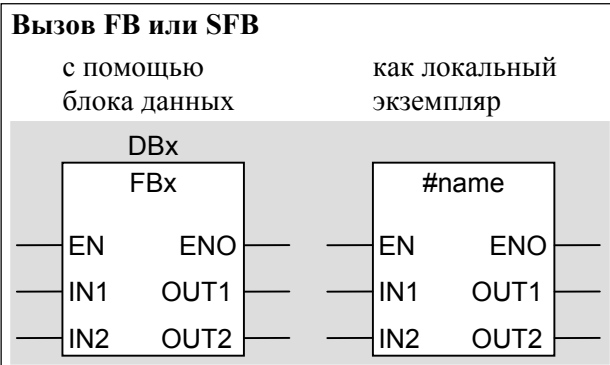
18 Функции для работы с блоками

В этой главе рассказывается о том, как вызывать и завершать кодовые блоки и как работать с операндами из блоков данных. Затем в следующей главе вы познакомитесь с использованием параметров блоков.

18.1 Функции для работы с кодовыми блоками

Функции для работы с кодовыми блоками (code blocks) включают в себя инструкции для вызова и завершения блоков (рисунок 18.1). Кодовые блоки вызываются с помощью блочного элемента вызова (call box). Если функции или системные функции не имеют параметров блока, они также могут быть вызваны с использованием катушки или блочного элемента CALL. В обоих случаях допустима предшествующая логическая операция, разрешающая условный вызов (вызов, зависящий от условий). Функция завершения блока RET всегда зависит от RLO (результата предшествующей логической операции).

Представление LAD



Представление FBD

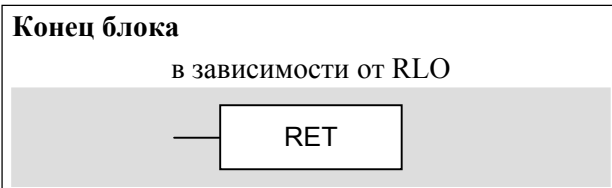
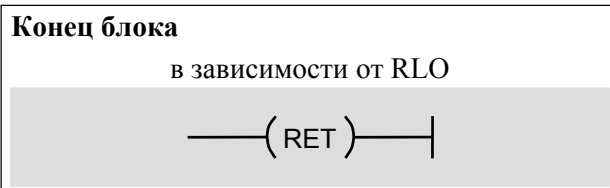
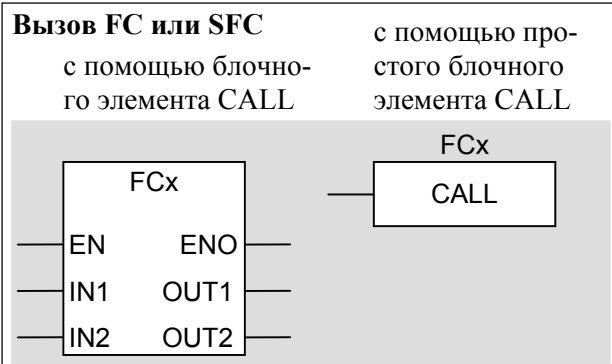
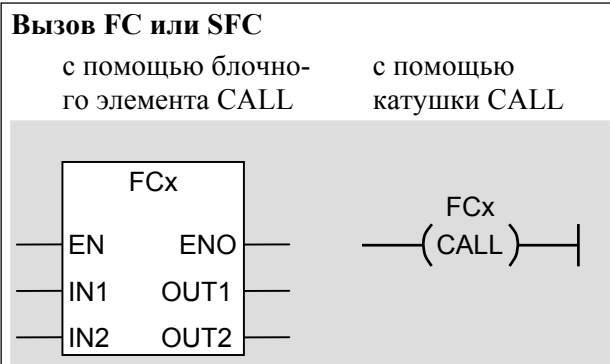
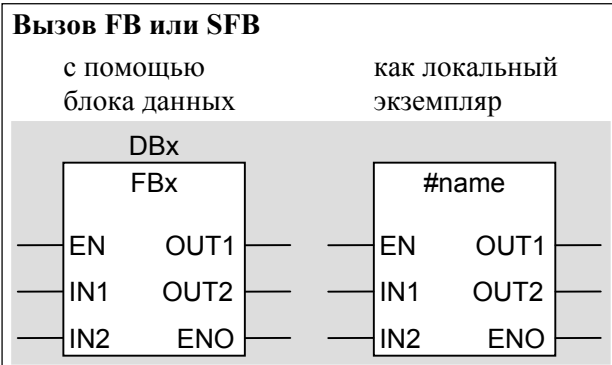


Рисунок 18.1 Функции для работы с кодовыми блоками

Кроме смены блока блочный элемент вызова также содержит пересылку параметров блока. Когда вызываются функциональные блоки, он также открывает экземпляр блока данных. Катушка / блочный элемент CALL есть ничто иное, как переход к другому блоку, и имеет значение (и допустим) только в случае функций и системных функций.

После завершения блока, вслед за функцией вызова CPU продолжает выполнение программы в блоке, который осуществил вызов (вызывающий блок). Если организационный блок завершен, то CPU передает управление операционной системе.

В случае пошагового программирования вы сможете найти катушки / блочные элементы CALL и RET в каталоге программных элементов (Program Elements Catalog), вызвав его командой меню View → Catalog (Вид → Каталог) [Ctrl – K], а также применив команду Insert → Program Elements (Вставка → Программные элементы), раздел «Program Control» («Программное управление»). Вы можете вставлять в программу вызовы блоков с использованием блочных элементов вызова при выборе блоков из «FC/FB/SFC/SFB blocks» («Блоки FC/FB/SFC/SFB»), «Multiple Instances» («Мультиэкземпляры») или «Libraries» («Библиотеки»).

18.1.1 Вызовы блоков: общая информация

Если кодовый блок предназначен для обработки, он должен быть «вызван». На рисунке 18.2 представлен пример вызова функции FC 10 в организационном блоке OB1.

Вызов блока состоит из блочного элемента вызова, который содержит адрес вызываемого блока (здесь: FC 10), разрешающий вход (enable input) EN, разрешающий выход (enable output) ENO и любые параметры блока. После обработки функции вызова CPU продолжает выполнение программы в вызванном блоке. Блок обрабатывается до конца или до встречи функции завершения блока. Затем CPU возвращается к вызывающему блоку (здесь: OB 1) и продолжает обработку этого блока после блочного элемента вызова.

Информация, необходимая CPU для нахождения обратного пути в вызывающий блок, сохраняется в стеке блоков (В-стек). С каждым новым вызовом блока создается новый элемент стека, включающий в себя адрес возврата, содержимое регистра блока данных и адрес локального стека данных вызываемого блока.

Если CPU в результате ошибки переходит в состояние STOP, то вы можете, используя программирующее устройство, увидеть по содержимому В-стека, какой блок обрабатывался до возникновения ошибки.

Вы можете переслать в вызываемый блок и из него данные для обработки. Эти данные передаются через параметры блока. Посредством блочного элемента вызова можно также вызывать блоки без параметров.

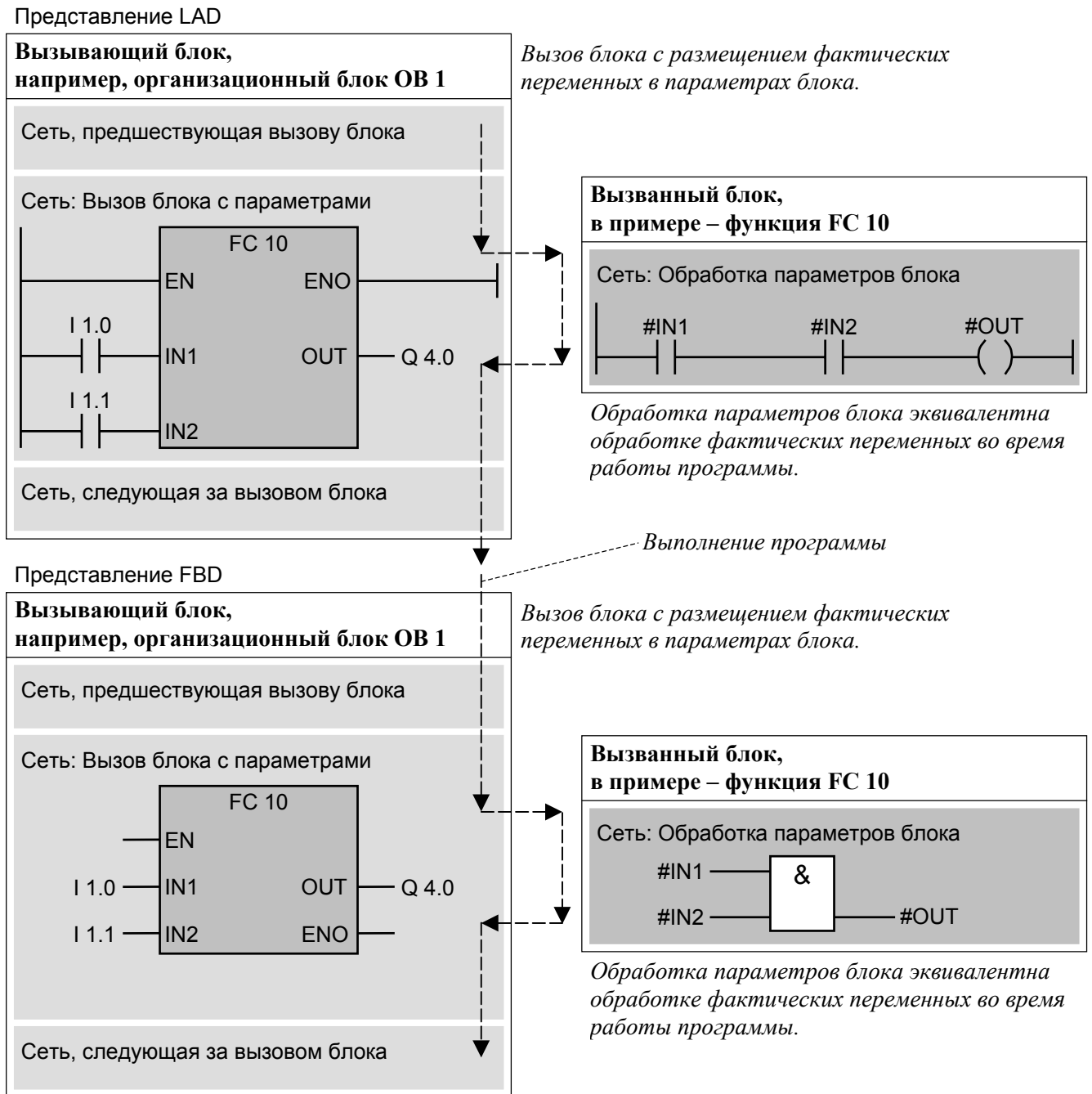


Рисунок 18.2 Пример вызова блока

18.1.2 Блочный элемент вызова

Блочный элемент вызова используется для вызова блоков FB, FC, SFB и SFC. (Вызывать организационные блоки нельзя, так как они управляются событиями и запускаются операционной системой.)

Для введения условий вызова можно воспользоваться входом EN блочного элемента вызова. Если вход EN подключен непосредственно к левой направляющей (шине питания), то такой вызов будет абсолютным, и он выполняется всегда. Если перед входом EN имеется логическая операция, то блочный элемент вызова выполняется,

только если предшествующая логическая операция выполнена. Выход ENO имеет то же сигнальное состояние, что и бинарный результат BR при осуществлении выхода из вызванного блока.

ЕСЛИ EN == "1" или не используется		ИНАЧЕ
ТО		
Вызванный блок обрабатывается		Вызванный блок не обрабатывается
ЕСЛИ вызванный блок возвращает BR = "1"		
ТО	ИНАЧЕ	
ENO := "1"	ENO := "0"	ENO := "0"

Вы должны промаркировать (назначить) параметры вызванного блока с использованием абсолютных или символических операндов. Если тип параметра BOOL, то этому параметру должны предшествовать

- контакт или цепь (LAD) или
- двоичная переменная или бинарная логическая операция (FBD).

Логический выходной параметр в дальнейшем не может быть скомбинирован.

LAD: Вы можете скомпоновать несколько блочных элементов вызова последовательно, соединив их друг с другом через EN или ENO. Блочный элемент вызова может быть вставлен в параллельную цепь, если только он непосредственно соединен с левой направляющей (шиной питания).

FBD: Блочные элементы вызова можно соединить последовательно, подключив выход ENO элемента к входу EN следующего. Выходы ENO нескольких блочных элементов могут быть объединены по AND или OR.

Когда блок вызывается, MCR-зависимость деактивируется. MCR отменяется в вызванном блоке независимо от того, было MCR разрешено или заблокировано до вызова блока. При выходе из блока MCR-зависимость принимает установки, которые были у нее перед вызовом блока.

В зависимости от параметров блока вы можете изменить содержимое регистров блока данных при осуществлении смены блока. Если *вызванный* блок является функциональным блоком, то экземплярный блок всегда открывается в этом блоке через регистр DI. Если *вызывающий* блок относится к функциональным блокам, то содержимое регистра DI (экземплярный блок данных) после вызова блока сохраняется. Содержимое регистра DB зависит, помимо прочего, от передаваемых параметров блока.

Вызов функциональных блоков

Вызвать функциональный блок вы можете, выбрав соответствующий функциональный блок из раздела «FB Blocks» («Блоки FB») каталога программных элементов (Program Elements Catalog). Необходимым условием является наличие вызываемого функционального блока в пользовательской программе. Экземплярный блок данных, принадлежащий вызову, вы должны записать над блочным элементом. Оба блока (функциональный блок и экземпляр блока данных) могут иметь абсолютные и символические адреса.

В случае функциональных блоков при вызове не требуется инициализировать все параметры блока. Инициализируемые параметры блока сохраняют свои текущие значения.

Вы также можете вызвать «функциональные блоки с возможностью мультиэкземплярности» в качестве локального экземпляра из других «функциональных блоков с возможностью мультиэкземплярности». При этом вызываемый функциональный блок использует экземпляр блока данных вызывающего функционального блока как хранилище для своих локальных данных. Перед вызовом вы должны объявить локальный экземпляр в статических локальных данных вызывающего функционального блока (блока, который вы в настоящий момент программируете). Локальный экземпляр вызывается путем выбора одного из доступных локальных экземпляров в разделе «Multiple Instances» («Мультиэкземпляры») в каталоге программных элементов; экземплярный блок данных определять необязательно (см. также параграф 18.1.6 «Статические локальные данные»).

Вызов функций

Вызов функций осуществляется путем выбора соответствующей функции из раздела «FC Blocks» («Блоки FC») в каталоге программных элементов. Функция должна иметь абсолютный или символический адрес.

Когда вы вызываете функции, вы должны инициализировать все доступные параметры.

Вызов функций со значением функции принимает точно такую же форму, как и вызов функций без значения функции. Только первый выходной параметр, соответствующий значению функции, имеет имя RET_VAL.

Вызов системных блоков

Операционная система CPU содержит системные функции (SFC) и системные функциональные блоки (SFB), которыми вы можете воспользоваться. Количество и тип системных блоков зависит от CPU. Все системные блоки вы можете вызвать с помощью блочного элемента вызова.

Вызвать системный блок вы можете тем же способом, что и блок, написанный вами; настройте соответствующий экземпляр блока данных в пользовательской памяти с тем же типом данных, который имеет SFB.

Системная функция и функция, созданная вами, вызываются одинаково.

Системные блоки существуют только в операционной системе CPU. Если вы хотите вызвать системные блоки во время автономного программирования, программному редактору требуется описание интерфейса вызова, чтобы он мог инициализировать параметры. Вы найдете это описание интерфейса в разделе «System Function Blocks» («Системные функциональные блоки») в библиотеке «Standard Library» («Стандартная библиотека»). Отсюда программный редактор копирует описание интерфейса в контейнер автономных блоков, когда вы вызываете системный блок. Скопированное таким образом описание интерфейса затем появляется как «нормальный» блочный объект.

Каталог программных элементов предоставляет системные блоки, доступные в настоящий момент в пользовательской программе, в разделе «SFC Blocks» («Блоки SFC») или «SFB Blocks» («Блоки SFB»). Вы можете, к примеру, выбрать мышью системный блок из каталога программных элементов и перетащить его в блок, обрабатываемый в настоящий момент, где он позже вызывается. Одновременно этот блок (или точнее описание его интерфейса) копируется в контейнер блоков.

18.1.3 Катушка / блочный элемент CALL

Вы можете вызывать функции и системные функции с использованием катушки / блочного элемента CALL. При этом вызываемые блоки не должны иметь параметров. Если блок очень длинный или не достаточно понятен для вас, можно использовать катушку / блочный элемент CALL, просто «разбив» блок на разделы и вызывая разделы один за другим. В одной сети допускается одна катушка или один блочный элемент CALL.

LAD: Если катушка CALL соединена непосредственно с левой направляющей (шиной питания), то вызов выполняется всегда (безусловный вызов).

Если имеется логическая операция, предшествующая катушке CALL, то вызов осуществляется, только если выполняется предшествующая логическая операция, то есть если в катушке CALL течет ток. Если указанная логическая операция не выполнена, то вызова не происходит, и обрабатывается следующая сеть.

FBD: Если блочному элементу CALL предшествует логическая операция, то вызов выполняется, только когда успешно обработана логическая операция, то есть когда $RLO = \langle 1 \rangle$ присутствует на блочном элементе CALL. Если предыдущая логическая операция не выполнена, вызов не осуществляется, немедленно начинает обрабатываться следующая сеть.

Когда производится смена блоков, сбрасывается бит состояния OS; биты состояния CC0, CC1 и OV не затрагиваются.


При вызове блока деактивируется MCR-зависимость. В вызванном блоке MCR отключается, несмотря на то, был ли MCR разрешен или отменен перед вызовом блока. Когда осуществляется выход из блока, MCR-зависимость приобретает те же установки, которые она имела до вызова блока.


Вызов блока с применением катушки / блочного элемента CALL сохраняет регистры блока данных в В-стеке; завершение блока восстанавливает их содержимое, когда производится выход из блока. Текущий перед вызовом блока глобальный блок данных и экземплярный блок данных также открываются вслед за вызовом блока. Если до вызова блока не был открыт блок данных (например, в ОВ 1 нет экземпляра блока данных), то и после вызова блока блок данных также не открыт, несмотря на это, блоки данных могут открываться в вызванном блоке.

18.1.4 Функция завершения блока

Вы можете преждевременно завершить обработку в блоке с помощью функции окончания блока RET.

Условное завершение блока

Представление LAD: 

Представление FBD: 

Функция завершения блок представляется в виде катушки или блочного элемента, для которых требуется предшествующая логическая операция. В сети катушка / блочный элемент RET должен быть единственным.

Если предшествующая логическая операция выполнена, то производится выход из блока. Возвратный переход осуществляется в предыдущий обрабатываемый блок, в котором имел место вызов блока. Если завершается организационный блок, то CPU передаст управление системной программе.

Если предшествующая логическая операция не реализована, обрабатывается следующая сеть в блоке.

ЕСЛИ предшествующая логическая операция == "1"	
ТО	ИНАЧЕ
Выполняется выход из блока	Обрабатывается следующая сеть
BR := "1"	BR := "0"

Катушка / блочный элемент RET одновременно сохраняет RLO (протекал ли ток или нет) в бинарном результате BR независимо от того, была выполнена логическая операция или нет. Бинарный результат играет решающую роль в управлении выходом ENO блочного элемента вызова (см. также главу 15 «Биты состояния»).

18.1.5 Временные локальные данные

Вы можете использовать временные локальные данные для буферизации результатов, получаемых при обработке блока. Временные локальные данные доступны только во время обработки блока; по завершению обработки блока ваши буферизованные данные теряются.

Временные локальные данные – это операнды, которые располагаются в стеке локальных данных (L-стеке) в системной памяти. Операционная система CPU делает временные локальные данные доступными для кодового блока, когда этот блок вызывается. При вызове блока значения в L-стеке фактически случайные. Чтобы использование локальных данных имело смысл, вы перед чтением должны их заполнить. Когда блок завершается, L-стек назначается следующему вызываемому блоку.

Количество байт временных локальных данных, требующееся блоку, записано в заголовке блока. Прочитав заголовок, операционная система узнает, сколько байт должно быть зарезервировано в L-стеке при вызове блока. Вы также можете увидеть из записи в заголовке блока, сколько байт локальных данных необходимо блоку. Для этого в редакторе при открытом блоке выбирается команда File → Properties (Файл → Свойства), или пункт меню SIMATIC-менеджера Edit → Object Properties (Правка → Свойства объекта), в каждом случае используется вкладка «General – Part 2» («Общие – часть 2»).

Объявление временных локальных данных

Вы должны объявить временные локальные данные в разделе описаний кодового блока:

- под «temp» («временные») в случае пошагового программирования или
- между VAR_TEMP и END_VAR в случае программирования, ориентированного на источник (исходные файлы).

Рисунок 18.3 демонстрирует пример объявления временных локальных данных. Переменная *temp1* располагается во временных локальных данных и имеет тип INT; переменная *temp2* имеет тип REAL.

Временные локальные данные хранятся в L-стеке в порядке их описания и в соответствии с их типом данных.

Address (Адрес)	Declaration (Описание)	Name (Имя)	Type (Тип)	Initial value (Начальное значение)	Comment (Комментарий)
0.0	in	Man_on	BOOL	FALSE	Input parameter (Входной параметр)
2.0	out	Switch_on	BOOL	FALSE	Output parameter (Выходной параметр)
4.0	in_out	Length	INT	0	I/O parameter (Входной / выходной параметр)
6.0	stat	Total	INT	0	Static local data (Статические локальные данные)
8.0	stat	Setpoint	DINT	L#0	
0.0	temp	Deviation	INT		Temporary local data (Временные локальные данные)
2.0	temp	Intermediate	REAL		

Рисунок 18.3 Пример объявления локальных данных в функциональном блоке

Символическая адресация временных локальных данных

Ссылка на временные локальные данные происходит по их символическим именам. Вы должны назначать эти имена в соответствии с правилами для внутриблочных (локальных) символов.

Все операции, разрешенные для меркеров, также допустимы для временных локальных данных. Обратите внимание, однако, что бит временных локальных данных не подходит для использования в качестве меркера фронта, потому что он не сохраняет свое сигнальное состояние вне соответствующего блока.

Вы можете адресовать временные локальные данные для блока только внутри этого блока (исключение: временные локальные данные для вызывающего блока могут быть доступны через параметры блока)

Размер L-стека

Общий размер L-стека зависит от версии CPU. Доступное количество байтов временных локальных данных в приоритетном классе, то есть в программе организационного блока, также предопределено. В S7-300 объем временных локальных данных фиксирован, например, в CPU 314 на приоритетный класс выделено 256 байтов. В S7-400 вы можете при инициализации CPU задать требуемое количество байтов временных локальных данных. Эти байты должны быть поделены блоками, вызываемыми в соответствующих организационных блоках, а также блоками, которые они вызывают.

Обратите внимание, что редактор также использует временные локальные данные, к примеру, для передачи параметров блока, этот факт проходит незамеченным в ходе программирования интерфейса.

Первичная информация

Когда вызывается организационный блок, операционная система CPU передает первичную (стартовую) информацию во временные локальные данные. Эта первичная информация включает 20 байтов для каждого организационного блока и почти идентична для всех блоков ОВ. О первичной информации для различных организационных блоков подробно рассказывается в главах 20 «Главная программа», 21 «Обработка прерываний», 22 «Характеристики рестарта» и 23 «Обработка ошибок».

Эти 20 байтов должны быть всегда доступны в каждом используемом приоритетном классе. Если вы программируете процедуру для обнаружения синхронных ошибок (программные ошибки и ошибки доступа), то вы должны установить отдельно дополнительные 20 байтов (по крайней мере) для первичной информации этих организационных блоков обработки ошибок, так как эти ОВ обработки ошибок принадлежат одному приоритетному классу.

Вы описываете стартовую информацию для организационного блока при программировании этого блока. Эти данные являются обязательными. Простые описания на английском языке находятся в *Standard Library* (*Стандартная библиотека*) в разделе *Organization Blocks* (*Организационные блоки*). Если первичная информация вам не нужна, просто объявите первые 20 байтов как что-то другое, например, как массив (как показано на рисунке 18.4).

Address (Адрес)	Declaration (Описание)	Name (Имя)	Type (Тип)
0.0	temp	SINFO	ARRAY [1..20]
*1.0	temp		BYTE
20.0	temp	LByte	ARRAY [1..16]
*1.0	temp		BYTE

Рисунок 18.4 Пример объявления временных локальных данных в организационном блоке

Абсолютная адресация временных локальных данных

Обычно ссылка на временные локальные данные происходит по их символическим именам. Использование абсолютных адресов является исключением. Познакомившись со способом хранения данных в L-стеке, вы можете самостоятельно вычислить адреса статических локальных данных. Также вы можете увидеть список адресов в таблице описания (объявления) переменных компилированного блока.

Идентификатор операнда временных локальных данных – L; бит адресуется с L, байт – с LB, слово – с LW, а двойное слово – с LD.

Пример: Вам требуется зарезервировать 16 байт временных локальных данных для абсолютной адресации, и вы хотите обращаться к значениям в этих байтах по байтам и битам. Для этого создайте массив в начале области локальных данных, так чтобы

адресация начиналась с 0. В организационном блоке в должны поместить объявление этого массива сразу после описания первичной информации, в этом случае адресация начнется с 20.

Тип данных ANY

Переменная во временных локальных данных может быть – хотя это исключение – объявлена как переменная типа данных ANY (Любой). Вы можете использовать это свойство для изменения указателя ANY во время исполнения программы (см. параграф 24.2.5 «Переменный» указатель ANY»).

18.1.6 Статические локальные данные

Статические локальные данные – это операнды, которые функциональный блок хранит в своем экземпляре блока данных.

Статические локальные данные являются «памятью» функционального блока. Они сохраняют свои значения до изменения этих значений программой, как в случае операндов данных в глобальных блоках данных.

Число байт статических локальных данных ограничено типами данных переменных и определяемым версией CPU размером блока данных.

Объявление статических локальных данных

Вы можете объявить статические локальные данные в разделе описаний функционального блока:

- с типом объявления переменной (declaration) «stat» в случае пошагового программирования или
- между VAR и END_VAR в случае программирования, ориентированного на источник (исходные файлы).

Рисунок 18.3 параграфа 18.1.5 «Временные локальные данные» содержит пример объявления переменной в функциональном блоке. Параметры блока объявляются первыми, затем статические локальные данные и, наконец, временные локальные данные.

Статические локальные данные хранятся в экземпляре блока данных после параметров блока в порядке их описаний и в соответствии с их типами данных.

Символическая адресация статических локальных данных

Вы можете обращаться к статическим локальным данным с помощью символических имен. Назначаются эти имена в соответствии с правилами для внутриблочных (локальных) символов.

Все операции, которые могут адресовать операнды данных в глобальных блоках данных, могут также адресовать статические локальные данные.

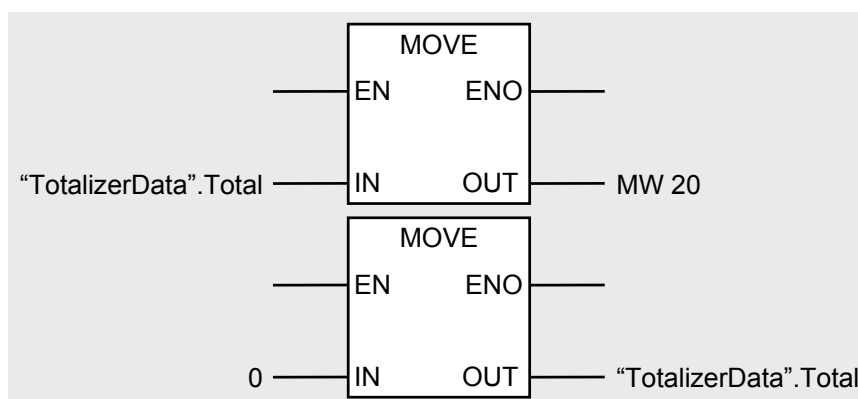
Пример: Функциональный блок «Totalizer» прибавляет входное значение к значению, хранящемуся в статических локальных данных, и сохраняет сумму в статических локальных данных. При следующем вызове входное значение вновь складывается с этой суммой и так далее (см. верхнюю часть рисунка 18.5).

Total – переменная в блоке данных «TotalizerData», экземпляром блока данных для функционального блока данных «Totalizer» (вы можете определить имена всех блоков самостоятельно в таблице символов, но при этом вы должны придерживаться применяемым правилам). Экземпляр блока данных имеет структуру данных функционального блока; в примере он содержит две переменных *In* и *Total* типа INT.

Доступ к статическим локальным данным вне функционального блока

Как правило, статические локальные данные обрабатываются в самом функциональном блоке, так как они хранятся в блоке данных. Однако, вы можете получить доступ к статическим локальным данным в любой момент времени с помощью «*Data Block Name*». *Operand Name* («Имя блока данных». *Имя операнда*), так же как к переменной в глобальном блоке данных.

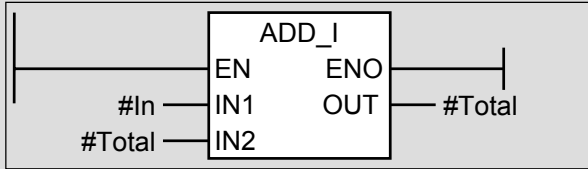
В нашем небольшом примере блок данных именуется *TotalizerData*, а операнд данных – *Total*. Применяемые инструкции доступа могут быть следующие.



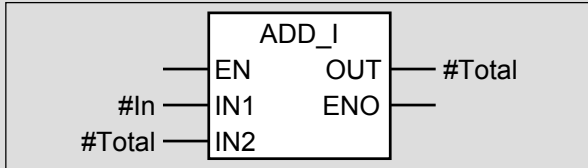
FB "Totalizer"

Address	Declaration	Name	Type
+ 0.0	in	In	INT
+ 2.0	stat	Total	INT

Представление LAD



Представление FBD



DB "TotalizerData"

Address	Declaration	Name	Type
+ 0.0	in	In	INT
+ 2.0	stat	Total	INT

В режиме просмотра данных (окне просмотра data view) блок данных отображает каждую отдельную переменную, так что переменные локального экземпляра выводятся с их полными именами.

В то же время вы можете видеть абсолютный адрес каждой переменной.

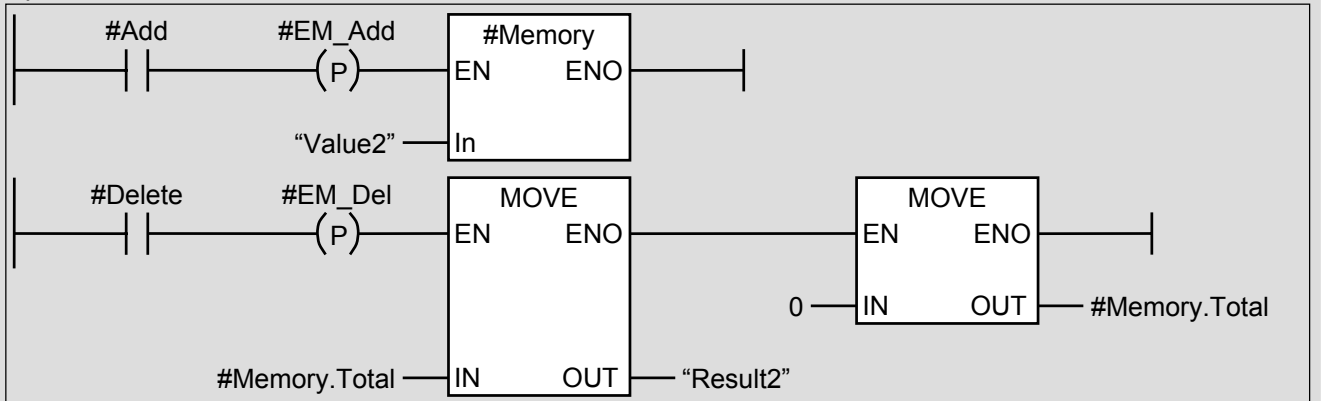
FB "Evaluation"

Address	Declaration	Name	Type
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

DB "EvaluationData"

Address	Declaration	Name	Type
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

Представление LAD



Представление FBD

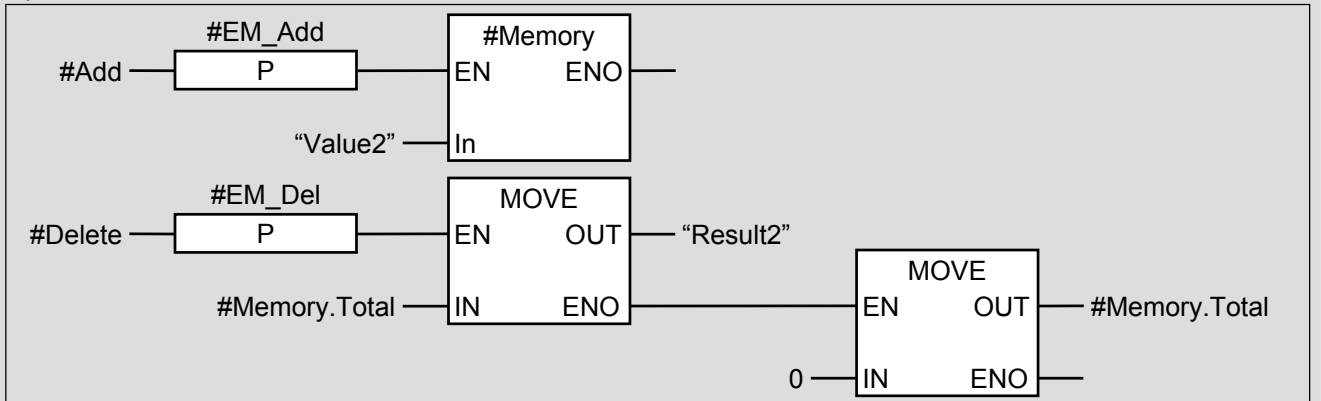


Рисунок 18.5 Пример статических локальных данных и локальных экземпляров

Локальные экземпляры

Когда вызывается функциональный блок, обычно определяется экземпляр блока данных для этого вызова. Затем функциональный блок сохраняет в этом экземпляре блока данных свои параметры блока и статические локальные данные.

Начиная со STEP 7 V2, имеется возможность генерирования «мультиэкземпляры», то есть вы можете вызывать функциональный блок как локальный экземпляр из другого функционального блока. Статические локальные данные (и параметры блока) вызванного функционального блока становятся подмножеством статических локальных данных вызывающего блока. Обязательным условием является то, что вызывающий и вызываемый блоки должны быть версии 2, то есть оба эти блока должны иметь «возможность мультиэкземплярности». Это позволяет вам «вкладывать» вызовы блоков до глубины вложения 8.

Пример (рисунок 18.5, нижняя часть): В статических локальных данных функционального блока «Evaluation» объявляется переменная *Memory*, которая соответствует функциональному блоку «Totalizer», и имеет собственную структуру. Теперь вы можете вызвать функциональный блок «Totalizer» через переменную *Memory*, но без определения блока данных, так как данные для *Memory* хранятся «внутриблочно» («локально») в статических локальных данных (*Memory* является локальным экземпляром функционального блока «Totalizer»).

Вы можете обращаться к статическим локальным данным *Memory* в программе в функциональном блоке «Evaluation» точно так же, как и к компонентам структуры, то есть путем указания имени структуры (*Memory*) и имени компонента (*Total*).

Экземплярный блок данных «EvaluationData», таким образом, содержит переменные *Memory.In* и *Memory.Total*, которые также можно адресовать как глобальные переменные, например, как «EvaluationData». *Memory.Total*.

Приведенный пример по использованию локального экземпляра находится в функциональных блоках FB 6, FB 7 и FB 8 в программе «Program Flow Control» («Управление программным потоком») на прилагаемой к книге дискете.

В примере «Устройство подачи» параграфа 19.5.3 «Пример устройства подачи» показаны дополнительные способы применения локальных экземпляров.

Абсолютная адресация статических локальных данных

Как правило, статические локальные данные адресуются символически. Абсолютная адресация является исключением. В функциональном блоке экземпляр блока данных открывается через регистр DI. DI, следовательно, служит операндам в этом блоке данных, и сюда включаются статические локальные данные и параметры блока, в качестве идентификатора операнда. Бит адресуется с DIX, байт – с DIB, слово – с DIW и двойное слово – с DID.

Узнав, как данные хранятся в блоке данных, вы сами можете вычислить абсолютные адреса статических локальных данных. Вы также можете найти адреса в скомпилированном блоке в таблице описания переменных. Но будьте очень осторожны! *Эти адреса являются адресами, соответствующими началу экземпляра.* Они применимы, только когда вы вызываете функциональный блок с блоком данных. Если функциональный блок вызывается как локальный экземпляр, то локальные данные локального экземпляра располагаются в середине экземплярного блока данных вызывающего функционального блока. Вы можете просмотреть абсолютные адреса, например, в скомпилированном экземпляре блока данных, который содержит все локальные экземпляры. Если вы хотите прочитать адреса отдельных операндов локальных данных, выберите команду меню View → Data View (Вид → Просмотр данных).

Используя наш пример в качестве основы, переменная *Total* в функциональном блоке «Totalizer» может быть адресована с помощью DIW 2, если функциональный блок «Totalizer» был вызван с блоком данных (см. также операнды в блоке данных «TotalizerData»), и при помощи DIW 6, если функциональный блок «Totalizer» был вызван как локальный экземпляр в функциональном блоке «Evaluation» (см. также операнды в блоке данных «EvaluationData»).

18.2 Функции блоков для блоков данных

Данные программы хранятся в блоках данных. В принципе, для хранения данных вы также можете использовать область памяти меркеров; однако, блоки данных дают значительно больше возможностей благодаря объему данных, структурированию данных и типам данных. В этом параграфе демонстрируется

- как работать с операндами данных,
- как вызывать блоки данных и
- как создавать, удалять и тестировать блоки данных во время исполнения программы.

Вы можете использовать блоки данных в двух видах: как *глобальные блоки данных* (*global data blocks*), которые не назначаются кодовым блокам, и как *экземплярные блоки данных* (*instance data blocks*), которые назначаются функциональному блоку. Данные в глобальных блоках данных являются «свободными» данными, которыми может воспользоваться любой кодовый блок. Вы сами можете определить их объем и структуру непосредственно при программировании глобального блока данных. Экземплярный блок данных содержит только данные, с которыми работает соответствующий функциональный блок. Этот функциональный блок также определяет структуру и место хранения данных в «своем» экземпляре блока данных.

Количество и размер блоков данных зависит от версии CPU. Нумерация блоков данных начинается с 1; нет блока данных DB 0. Каждый блок данных может использоваться либо как глобальный блок данных, либо экземплярный блок данных.

Сначала вы должны создать блоки данных, используемые в вашей программе, либо запрограммировав, как кодовые блоки, либо при выполнении программы с использованием системной функции SFC 22 CREAT_DB.

18.2.1 Два регистра блоков данных

Каждый CPU S7 оснащен двумя регистрами блоков данных. Эти регистры содержат номера текущих блоков данных; это блоки данных, операнды которых в настоящий момент времени обрабатываются. Перед доступом к операнду блока данных в должны открыть блок данных, содержащий операнд. Если вы используете доступ по полному адресу (*fully-addressed access*) к операндам данных (со спецификацией блока данных см. ниже), то вам не нужно заботиться об открытии блоков данных или о содержимом регистров блоков данных. Редактор генерирует необходимые инструкции из ваших спецификаций.

Программный редактор использует первый регистр блоков данных главным образом для доступа к глобальным блокам данных, а второй регистр блоков данных – для обращения к экземплярам блоков данных. По этой причине указанные регистры называются «Регистр глобальных блоков данных» («Global Data Block Register», для краткости – регистр DB) и «Регистр экземплярных блоков данных» («Instance Data

Block Register», кратко – регистр DI). CPU обрабатывает регистры абсолютно идентичным образом. Каждый блок данных может быть открыт посредством одного из регистров (или с использованием двух одновременно).

Существенной особенностью является то, что вы можете воздействовать с помощью LAD или FBD только на регистр DB. Если вы открываете блок данных катушкой / блочным элементом OPN (см. ниже), то вы всегда делаете это через регистр DB. В LAD и FBD экземпляр блока данных всегда открывается через регистр DI; это относится к блочному элементу вызова, когда вызывается блок.

Когда вы загружаете слово данных, вы должны указать, какой из двух возможных открытых блоков данных содержит слово данных. Если блок данных открыт через регистр DB, то слово данных именуется DBW; если слово данных находится в блоке данных, открытом посредством регистра DI, оно называется DIW. Остальные операнды данных именуется соответственно (таблица 18.1).

Таблица 18.1 Операнды данных

Операнды данных	Расположены в блоке данных, открытом через	
	регистр DB	регистр DI
Бит данных	DBX y.x	DIХ y.x
Байт данных	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 Доступ к операндам данных

Вы можете использовать следующие методы доступа к операндам данных:

- Символическая адресация по полному адресу,
- Абсолютная адресация по полному адресу и
- Абсолютная адресация по частичному адресу.

Символический доступ к операндам данных в глобальных блоках данных требует минимум системных знаний. Для абсолютного обращения или для применения обоих регистров блоков данных вам необходимо изучить материалы, приведенные ниже.

Символическая адресация операндов данных

Рекомендуется адресовать операнды данных символически всякий раз, когда это возможно. Символическая адресация

- делает программу более читаемой и понятной (если в качестве символов используются значащие термины),
- сокращает количество ошибок при написании программ (программный редактор сравнивает термины, используемые в таблице символов и в программе; «ошибки смены номера», например, DBB 156 и DBB 165, которые могут возникнуть при использовании абсолютных адресов, здесь не происходят) и
- не требуется знание программирования на уровне машинного кода (какой блок данных CPU открыл в настоящий момент?).

Символическая адресация использует доступ по полному адресу (блок данных вместе с операндом данных), поэтому операнд данных всегда имеет уникальный адрес.

Определить символический адрес операнда данных вы можете в два шага:

- 1) Назначение блока данных в таблице символов
Блоки данных являются глобальными данными, имеющими в программе уникальные адреса. В таблице символов вы должны назначить символ (например, Motor1) абсолютному адресу блока данных (например, DB 51).
- 2) Назначение операнда данных в блоке данных
Вы должны указать имена операндов данных (и типы данных) при программировании блока данных. Имя применяется только в соответствующем блоке (оно «внутриблочное» или локальное). Вы можете также использовать это же имя в другом блоке для другой переменной.

Доступ к операндам данных по полному адресу

В случае обращения по полному адресу вы определяете блок данных вместе с операндом данных. Этот метод адресации может быть символическим или абсолютным:

```
"MOTOR1".ACTVAL
DB 51.DBW 20
```

MOTOR1 – это символический адрес, назначенный блоку данных в таблице символов. ACTVAL является операндом данных, определенным при программировании блока данных. Символическое имя "MOTOR1".ACTVAL представляет собой такую же уникальную спецификацию (описание) операнда данных, как и DB 51.DBW 20.

Доступ по полному адресу возможен только в сочетании с регистром глобальных блоков данных (регистром DB). В случае полностью адресованных операндов данных редактор программ сначала открывает блок данных через регистр DB, а затем обращается к операнду данных.

Вы можете применять доступ по полному адресу вместе со всеми функциями, допустимыми для типа данных адресованного операнда данных. Тогда, используя параметры блока, к примеру, вы можете указать полностью адресованный операнд в качестве фактического параметра.

Абсолютная адресация операндов данных

Для работы с абсолютной адресацией вы должны знать адреса, по которым редактор программ размещает операнды данных при их установке. Адреса вы можете узнать путем их вывода после программирования и компиляции блока данных: в столбце адресов вы можете увидеть абсолютный адрес, с которого начинается соответствующая переменная. Эта процедура подходит для всех блоков данных, как для используемых вами в качестве глобальных блоков данных, так и для экземплярных блоков данных (о локальных экземплярах рассказывается в параграфе 18.2.4 «Особые замечания по адресации данных»). Таким образом, вы можете также увидеть, где программный редактор хранит параметры блока и статические локальные данные для функциональных блоков.

Операнды данных адресуются побайтно (байт за байтом), как, например память меркеров; функции, работающие с операндами данных, те же, что и для памяти меркеров.

18.2.3 Открытие блока данных

Для того, чтобы открыть блок данных через регистр DB используются катушка OPN (LAD) или блочный элемент OPN (FBD).



Катушка / блочный элемент OPN соединен непосредственно с левой направляющей (шиной питания) и является единственным в цепи. Указываемый блок данных всегда открывается через регистр DB. В LAD или FBD невозможно открыть блок данных катушкой / блочным элементом OPN посредством регистра DI. (Регистр DI использует блочный элемент вызова для открытия текущего экземпляра блока данных.)

Для пошагового программирования катушку / блочный элемент OPN вы можете найти в каталоге программных элементов (Program Elements Catalog) в разделе «DB Call» («Вызов DB»).

Открытый блок данных во время исполнения программы должен находиться в пользовательской памяти.

В сетях, следующих за катушкой / блочным элементом OPN, частичную адресацию вы можете использовать для доступа к тем операндам данных, которые расположены в открытом блоке данных. Если вы хотите скопировать из одного блока данных в другой, то вы можете, например, обратиться к временным локальным данным через промежуточный буфер или (лучше) использовать полностью адресованные операнды.

ды данных. Замечание: полностью адресованные операнды данных перезаписывают регистр DB «своим» блоком данных.

Пример: Значение слова данных DBW 10 блок данных DB 13 должно быть передано в слово данных DBW 16 блока данных DB 14. С помощью блочного элемента MOVE можно копировать частично адресованные операнды данных только внутри открытого в текущий момент блока данных. Для копирования между блоками данных вам потребуется промежуточный буфер, например, слово локальных данных (см. рисунок 18.6). Удобнее использовать полную адресацию.

Когда блок данных открыт, он остается «действующим» до открытия другого блока данных. По определенным обстоятельствам вы не сможете это увидеть в программном редакторе, к примеру, если блок меняется блочным элементом вызова (обратитесь к параграфу 18.2.4 «Особые замечания по адресации данных»).

При смене блока с использованием катушки / блочного элемента CALL содержимое регистров блоков данных сохраняется. По возврату в вызывающий блок операция смена блока восстанавливает старое содержимое регистров.



Рисунок 18.6 Пример частично и полностью адресованных данных

18.2.4 Особые замечания по адресации данных

Изменения содержимого регистров DB

Используя следующие функции, редактор программ генерирует дополнительные функции, которые могут внести изменения в содержимое двух регистров DB:

Полная адресация операндов данных

Каждый раз при полной адресации операндов данных редактор программ сначала открывает блок данных, затем обращается к операнду данных. Регистр DB перезаписывается каждый раз. Это также относится к поддержке параметров блока с использованием полностью адресуемых операндов данных.

Доступ к параметрам блоков

Доступ к следующим параметрам блоков изменяет содержимое регистра DB: все параметры блоков сложных типов данных в случае функций и входные/выходные (in/out) параметры сложных типов данных в случае функциональных блоков.

Вызов функционального блока с помощью блочного элемента вызова

Перед фактическим вызовом функционального блока блочный элемент вызова записывает номер текущего экземпляра блока данных в регистр DB (путем свопинга регистров блоков данных, то есть обмена их содержимым) и открывает экземплярный блок данных для вызываемого функционального блока. Таким образом, экземплярный блок данных, ассоциированный с вызываемым функциональным блоком, всегда открыт. После фактического вызова блока блочный элемент вызова снова проводит свопинг регистров блоков данных, поэтому текущий экземплярный блок данных снова доступен в вызывающем функциональном блоке. Таким способом блочный элемент вызова изменяет содержимое регистра DB.

Регистр DI в функциональных блоках

В функциональных блоках регистр DI всегда содержит номер текущего экземплярного блока данных. Весь доступ к параметрам блока и статическим локальным данным осуществляется через регистр DI. Адрес локальных данных, заданный в таблице описаний (declaration table) функционального блока, действует, только когда вы открываете функциональный блок с экземплярным блоком данных, в этом случае операнды данных начинаются с нулевого байта.

Когда вы вызываете функциональный блок как локальный экземпляр, его данные находятся «в середине» экземплярного блока данных вызывающего блока данных. Они содержат абсолютный адрес локальных данных, когда вы выведете экземплярный блок данных в формате просмотра данных. При этом каждая отдельная пере-

менная отобразится с именем и адресом, включая переменные вызванного локального экземпляра.

Когда вы программируете функциональный блок, который позже вы хотите также использовать как локальный экземпляр, вам еще не известен абсолютный адрес переменной во время написания программы. В этом случае – как поступает программный редактор при символическом программировании – содержимое адресного регистра AR2 прибавляется к адресу переменной. Однако, это возможно только в языке программирования STL.

Изменение содержимого блоков данных в более позднее время

В окне свойств (Properties) контейнера автономных объектов *Blocks* (Блоки) в регистре «Blocks» вы можете указать, абсолютный или символический адрес операнда данных будет иметь приоритет для последующих отображений и операций сохранения при изменении в содержимом блоков данных кодовых блоков, которые уже сохранены.

По умолчанию установлено «Absolute address has priority» («Приоритет имеет абсолютный адрес») (так же как в предыдущих версиях STEP 7). Эта установка по умолчанию означает, что когда происходит изменение в описании, абсолютный адрес сохраняется в программе, пока символический адрес соответственно изменяется. Если выбрано «Symbolic address has priority» («Приоритет имеет символический адрес»), абсолютный адрес меняется, в то время как символический адрес остается неизменным.

Пример: Допустим, что слово данных DBW 10 в блоке данных DB 1 содержит символический адрес *ActValue*. В программе вы можете адресовать это слово данных так:

```
"Data".ActValue    DB1.DBW 10
```

Здесь “Data” – это символический адрес блока данных DB 1. Если вы теперь добавите дополнительное слово данных с символическим адресом *MaxCurrent* сразу перед словом данных DBW 10, то результат при следующем открытии (или сохранении) кодового блока будет следующим:

в случае «Absolute address has priority» («Приоритет имеет абсолютный адрес»):

```
"Data".MaxCurrent  DB1.DBW 10
```

в случае «Symbolic address has priority» («Приоритет имеет символический адрес»):

```
"Data".ActValue    DB1.DBW 12
```

Сказанное о доступе к операндам данных в глобальных блоках данных относится и к доступу к глобальным операндам (входам экземпляров), для которых символические адреса были введены в таблице символов. Подробную информацию по этому предмету вы можете найти в параграфе 2.5.5 «Приоритет адреса».

18.3 Системные функции для блоков данных

Существуют три системных функции для работы с блоками данных. Их параметры описаны в таблице 18.2. Это функции

- SFC 22 CREAT_DB
Создает блок данных;
- SFC 23 DEL_DB
Удаляет блок данных;
- SFC 24 TEST_DB
Тестирует блок данных.

18.3.1 Создание блока данных

Системная функция **SFC 22 CREAT_DB** создает блок данных в пользовательской памяти. В качестве номера блока данных системная функция берет наименьший свободный номер из диапазона номеров, заданного входными параметрами **LOW_LIMIT** и **UP_LIMIT**. Номера, указанные в этих параметрах, включаются в диапазон номеров. Если эти значения одинаковы, блок данных генерируется с указанным номером. Выходной параметр **DB_NUMBER** предоставляет фактический номер созданного блока. Во входном параметре **COUNT** задается размер создаваемого блока. Размер соответствует количеству байт данных и должен быть четным числом.

Создание блока не то же самое, что его вызов. Текущий блок данных все еще действителен. Созданный с использованием рассматриваемой системной функции блок данных содержит случайные данные. Для осмысленного использования созданный таким образом блок данных сначала должен быть заполнен данными, и после этого может быть прочитан.

При возникновении ошибки блок данных не создается, содержимое выходного параметра неопределено, и в качестве значения функции возвращается номер ошибки.

18.3.2 Удаление блока данных

Системная функция **SFC 23 DEL_DB** удаляет блок данных, находящийся в памяти RAM (рабочая и загрузочная память), номер которого указан во входном параметре **DB_NUMBER**. Блок данных не должен быть открыт в момент удаления, иначе CPU переходит в состояние STOP.

Блоки данных, созданные с указанием ключевого слова **UNLINKED** (Неприсоединенный), и блоки данных на карте памяти **FEPR0M** не могут быть удалены системной функцией SFC 23.

В случае возникновения ошибки блок данных не удаляется, и в качестве значения функции возвращается номер ошибки.

18.3.3 Тестирование блоков данных

Системная функция **SFC 24 TEST_DB** возвращает количество доступных байтов для блока данных в пользовательской памяти (в выходном параметре **DB_LENGTH**) и состояние защиты от записи (в выходном параметре **WRITE_PROT**). Вы можете задать номер выбранного блока данных во входном параметре **DB_NUMBER**.

В случае ошибки содержимое выходных параметров неопределено, и в качестве значения функции возвращается номер ошибки.

Таблица 18.2 Функции SFC для управления блоками данных

SFC	Имя	Объявление	Тип данных	Назначение, описание
22	LOW_LIMIT	INPUT	WORD	Наименьший номер создаваемого блока данных
	UP_LIMIT	INPUT	WORD	Наибольший номер создаваемого блока данных
	COUNT	INPUT	WORD	Размер блока данных в байтах (четное число)
	RET_VAL	OUTPUT	INT	Информация об ошибке
	DB_NUMBER	OUTPUT	WORD	Номер созданного блока данных
23	DB_NUMBER	INPUT	WORD	Номер удаляемого блока данных
	RET_VAL	OUTPUT	INT	Информация об ошибке
24	DB_NUMBER	INPUT	WORD	Номер тестируемого блока данных
	RET_VAL	OUTPUT	INT	Информация об ошибке
	DB_LENGTH	OUTPUT	WORD	Размер блока данных (в байтах)
	WRITE_PROT	OUTPUT	BOOL	«1» = защищен от записи

Содержание главы 19

19	<u>Параметры блоков</u>	4
19.1	<u>Общая информация о параметрах блоков</u>	4
19.1.1	<u>Определение параметров блоков</u>	4
19.1.2	<u>Обработка параметров блоков</u>	4
19.1.3	<u>Описание параметров блоков</u>	5
19.1.4	<u>Описание значения функции</u>	7
19.1.5	<u>Инициализация параметров блоков</u>	8
19.2	<u>Формальные параметры</u>	9
19.3	<u>Фактические параметры</u>	13
19.4	<u>«Передача» параметров блоков</u>	18
19.5	<u>Примеры</u>	19
19.5.1	<u>Пример «Ленточный транспортер конвейера»</u>	19
19.5.2	<u>Пример «Счетчик деталей»</u>	20
19.5.3	<u>Пример «Устройство подачи»</u>	21

19 Параметры блоков

В настоящей главе рассказывается о том, как использовать параметры блоков. Вы научитесь

- описывать параметры блоков,
- работать с параметрами блоков,
- инициализировать параметры блоков и
- «передавать» параметры блоков.

Параметры блоков представляют интерфейс передачи между вызывающим и вызываемым блоками. Все функции и функциональные блоки могут работать с параметрами.

19.1 Общая информация о параметрах блоков

19.1.1 Определение параметров блоков

Параметры блока дают возможность параметризовать обрабатываемую в блоке инструкцию, функцию блока. Пример: Вам требуется создать блок, суммирующий значения, и который вы сможете использовать в программе несколько раз с различными переменными. Переменные будут передаваться в качестве параметров блока; в нашем примере три входных параметра и один выходной параметр (рисунок 19.1).

Так как сумматору не нужно хранить внутренних значений, в качестве типа блока подойдет функция.

Вы определяете параметр блока как входной (*input parameter*), если в программе блока значение этого параметра только считывается или загружается. Если вы только описываете параметр блока (назначаете, устанавливаете, сбрасываете, пересылаете), то вы используете выходной параметр (*output parameter*).

Вы должны всегда использовать входной/выходной параметр (*in/out parameter*), если он должен и считываться, и перезаписываться. Программный редактор не проверяет использование параметров блоков.

19.1.2 Обработка параметров блоков

В программе сумматора имена параметров блока применяются как ячейки для последних фактических переменных. Параметры блоков используются так же, как и символически адресованные переменные; в программе они называются формальными параметрами (*formal parameters*).

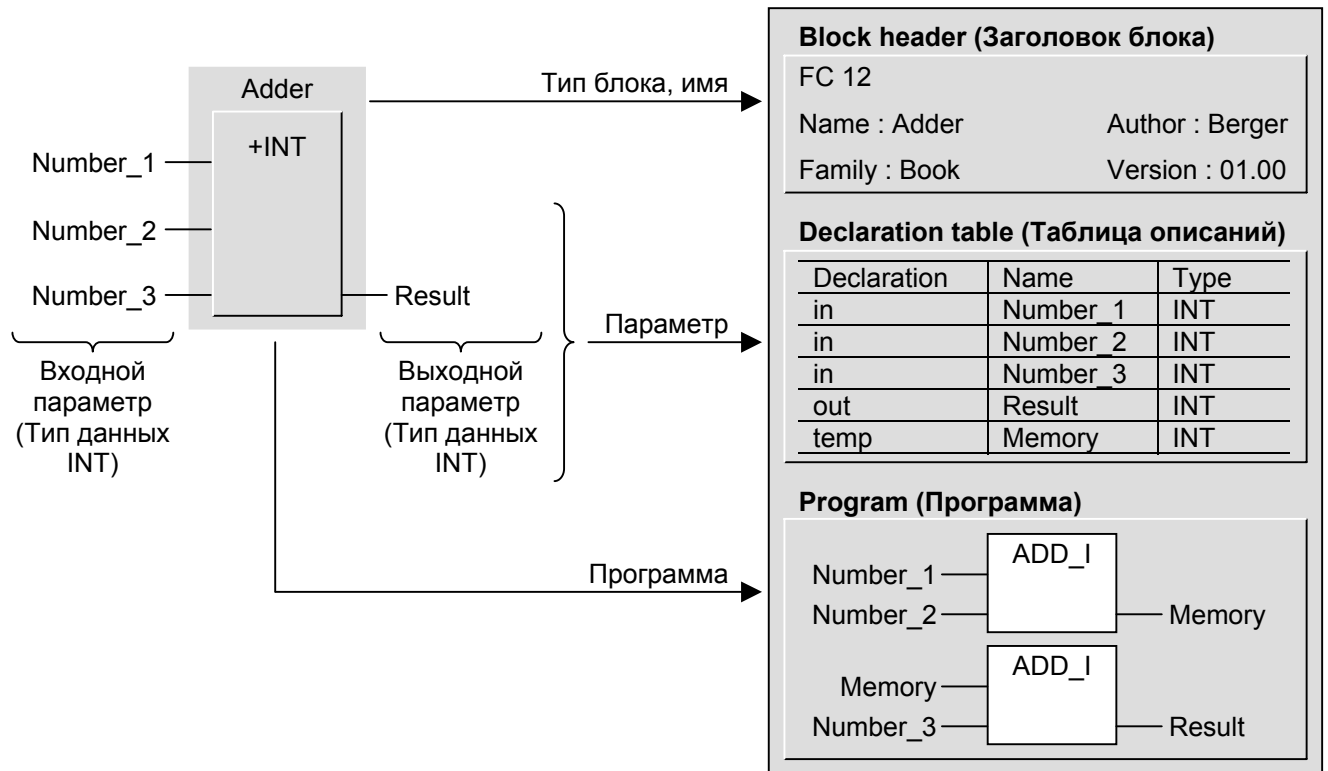


Рисунок 19.1 Пример параметров блока

В программе вы можете вызвать функцию «Adder» («Сумматор») несколько раз. При каждом вызове вы можете передавать другие значения в сумматор в параметрах блока (рисунок 19.2). Значения могут быть константами, операндами или переменными; они называются фактическими параметрами (actual parameters).

Во время исполнения программы CPU заменяет формальные параметры фактическими параметрами. В примере первый вызов складывает содержимое слов памяти MW 30, MW 32 и MW 34 и сохраняет результат в слове памяти MW 40. Тот же самый блок с фактическими параметрами при втором вызове суммирует слова данных DBW 30, DBW 32 и DBW 34 блока данных DB 13 и сохраняет результат в слове данных DBW 40 блока данных DB 14.

19.1.3 Описание параметров блоков

Параметры блока определяются в разделе описаний блока при его программировании. Ниже показана пустая таблица описаний (declaration table). Кроме параметров блока – in (входной), out (выходной), in_out (входной/выходной) – в этой таблице вы также опишите статические локальные данные (stat) и временные локальные данные (temp).

Значения по умолчанию являются необязательными и применяются только в практических целях для функциональных блоков, если параметр блока сохранен как значение. Это относится ко всем параметрам блоков простых типов данных и к входным и выходным параметрам сложных типов данных.

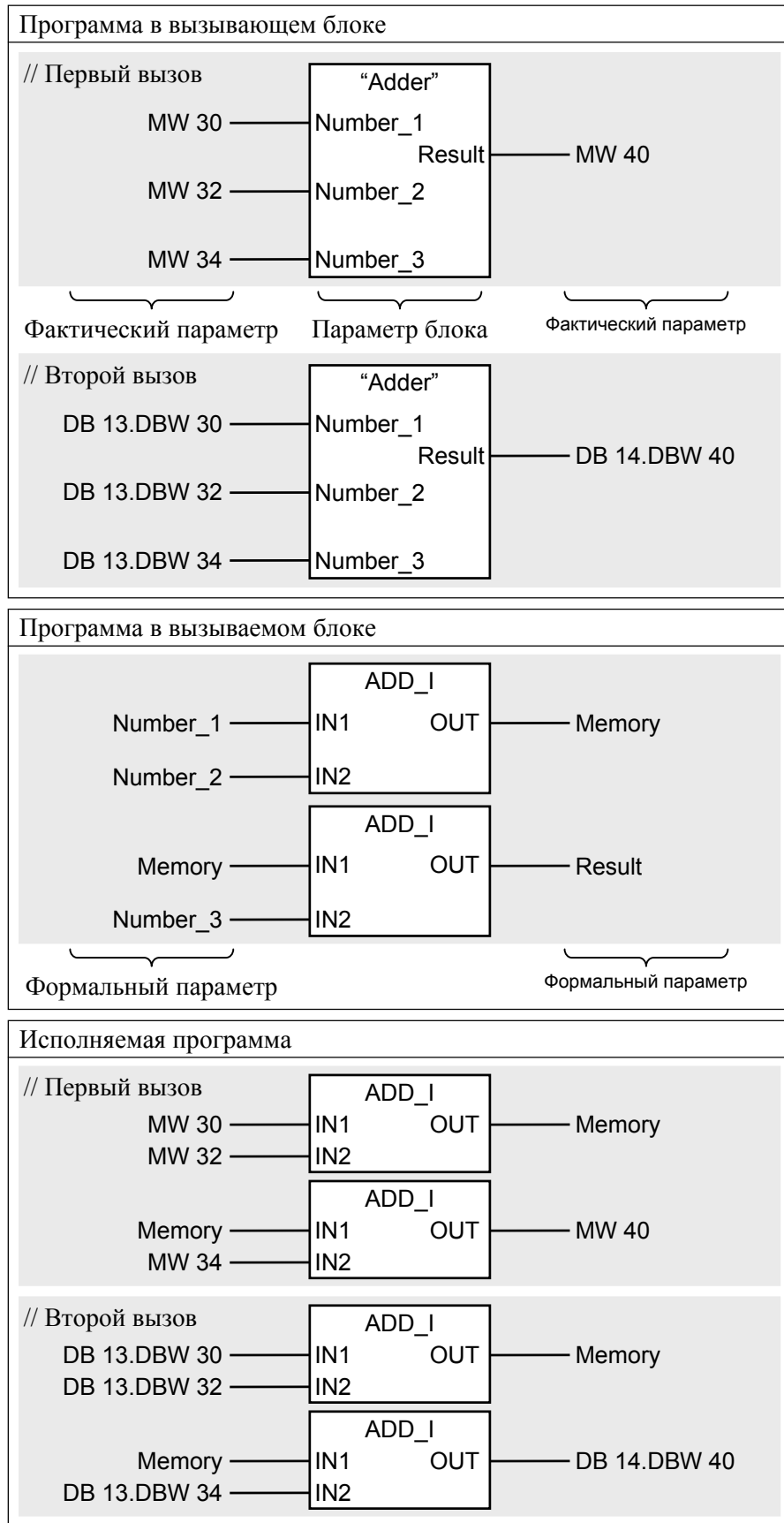


Рисунок 19.2 Вызов блока с параметрами

Таблица 19.1 Пустая таблица описаний

Address (Адрес)	Declaration (Описание)	Name (Имя)	Type (Тип)	Initial value (Начальное значение)	Comment (Комментарий)
	in				Input parameter (for FCs and FBs) / Входной параметр (для FC и FB)
	out				Output parameter (for FCs and FBs) / Выходной параметр (для FC и FB)
	in_out				In/out parameter (for FCs and FBs) / Входной/выходной параметр (для блоков FC и FB)
	stat				Static local data (for FBs) / Статические локальные данные (для FB)
	temp				Temporary local data (for OBs, FCs and FBs) / Временные локальные данные (для OB, FC и FB)

Также могут быть указаны комментарии для параметров.

Имя параметра блока может состоять из 24 знаков максимум. Имя может включать в себя только буквы (кроме национальных литер, таких как немецкий умлаут), цифры и знак подчеркивания. Между верхним и нижним регистром различия не делается. Имя не должно быть ключевым словом.

Для *типа данных* параметра блока допустимы все простые, сложные и определенные пользователем типы данных, а также параметрические типы (см. параграф 3.5 «Переменные, константы и типы данных»).

STEP 7 хранит имена параметров блоков в неисполняемом разделе блоков на носителе информации программирующего устройства. Пользовательская память CPU (в скомпилированном блоке) содержит только типы описаний (declaration types) и типы данных. Поэтому изменения программы, произведенные с блоками в онлайн-режиме в CPU, должны быть отражены в среде хранения программирующего устройства для того, чтобы сохранить исходные имена. Если обновление не осуществлено, или если блоки переданы из CPU в устройство программирования, то неисполняемые разделы блоков перезаписываются или стираются. Программный редактор генерирует символы замены для вывода на экран или при печати.

19.1.4 Описание значения функции

В функциях их значение – особым образом обрабатываемый выходной параметр. Он имеет имя RET_VAL (или ret_val) и определяется как первый выходной параметр. Вы должны объявить значение функции путем назначения имени RET_VAL первому выходному значению в таблице описаний.

В качестве типа данных значения функции разрешено использовать все простые типы, а также типы DATE_AND_TIME, STRING, POINTER, ANY и определенные пользователем типы данных UDT. Типы ARRAY и STRUCT не допускаются.

Как первый выходной параметр, значение функции не играет особой роли в языках программирования LAD и FBD. Он приобретает значение только в языке программирования STL, где тип блока FUNCTION (Функция) используется как «подлинная» функция. Здесь же функция FC может указываться вместо операнда при выводе; в этом случае значение функции представляет собой значение этой функции.

19.1.5 Инициализация параметров блоков

При вызове блока с параметрами вы должны инициализировать параметры блока фактическими параметрами. Это могут быть константы, операнды с абсолютными адресами, полностью адресованные операнды данных или символически адресованные переменные. Фактический параметр должен быть того же типа данных, что и параметр блока.

Каждый параметр функции должен быть инициализирован при каждом вызове. В случае функционального блока инициализация отдельных параметров блока или всех параметров необязательна.

19.2 Формальные параметры

В этом разделе вы узнаете о способах доступа к параметрам внутри блока. Вы также узнаете, что доступ к параметрам блоков простых типов данных, элементам массива или компонентам структуры, таймерам и счетчикам возможен без ограничения. Параграф 19.4 ««Передача» параметров блока» показывает вам, как можно «передать» параметры в вызываемые блоки.

Обращение к параметрам сложных типов данных и параметрических типов POINTER и ANY в LAD или FBD не поддерживается. Тем не менее, вы можете инициализировать готовые или системные блоки, которые имеют такие параметры, используя соответствующую переменную. На дискете, прилагаемой к книге, имеются примеры по этой теме в программе «Data Types» («Типы данных»).

Таблица 19.2 Доступ к параметрам блоков (обобщение)

Типы данных	Разрешено для			Доступ в блоке возможен
	IN	I_O	OUT	
Простые типы данных				
BOOL	x	x	x	да
BYTE, WORD, DWORD, CHAR, INT, DINT, REAL, S5TIME, TOD, DATE	x	x	x	да
Сложные типы данных				
DT, STRING	x	x	x	нет
ARRAY, STRUCT				
Отдельные бинарные компоненты	x	x	x	да
Отдельные числовые компоненты	x	x	x	да
Переменные полностью	x	x	x	нет
Параметрические типы				
TIMER	x	-	-	да
COUNTER	x	-	-	да
BLOCK_FC, BLOCK_FB	x	-	-	да
BLOCK_DB	x	-	-	да
BLOCK_SDB	x	-	-	нет
POINTER, ANY	x	x	x ¹⁾	нет

¹⁾ только функции FC

Параметры блока типа данных BOOL

Параметры блока типа BOOL могут быть отдельными двоичными переменными или бинарными компонентами массивов и структур. Вы можете считать входные параметры и входные/выходные параметры с помощью контактов или двоичных входов

блочных элементов, и можно воздействовать на выходные параметры и входные/выходные параметры, используя двоичные выходы блочных элементов.

После того, как CPU использовал фактический параметр, определенный в качестве параметра блока, он обрабатывает функции, как показано в соответствующих главах.

Параметры блоков числового типа данных

Параметры блока числового типа занимают 8, 16 или 32 бита (все простые типы данных за исключением BOOL). Они могут быть отдельными числовыми переменными или числовыми компонентами массивов и структур.

Вы можете подавать входные и входные/выходные параметры на цифровые входы блочных элементов, и можно записывать выходные и входные/выходные параметры в цифровые входы блочных элементов.

Обмен значениями между параметрами блоков различных типов и разных размеров может быть произведен с использованием блочного элемента MOVE, как это описано в главе 6 «Функции передачи».

Параметры блоков типов данных DT и STRING

Прямой доступ к параметрам блоков типов DT и STRING невозможен. В функциональных блоках вы можете «передать» входные и выходные параметры типов DT и STRING параметрам вызванного блока.

Параметры блоков типов данных ARRAY и STRUCT

Возможно прямое покомпонентное обращение к параметрам блоков типов ARRAY и STRUCT, то есть вы можете обратиться к отдельным двоичным или числовым компонентам с помощью соответствующих операций.

Доступ ко всей переменной (ко всему массиву или всей структуре) невозможен, как и доступ к отдельным компонентам сложных или определенных пользователем типов данных. В функциональных блоках вы можете передать входные и выходные параметры типов ARRAY и STRUCT параметрам вызванного блока.

Параметры блоков определенного пользователем типа данных

Обработка параметров блока определенного пользователем типа данных осуществляется тем же способом, что и параметры блока типа STRUCT.

Возможен прямой покомпонентный доступ к параметрам типа UDT, то есть вы можете обратиться к отдельным двоичным или числовым компонентам с использованием соответствующих операций.

Доступ ко всей переменной невозможен. Также нельзя обратиться к отдельным компонентам сложного или определенного пользователем типа данных.

В функциональных блоках вы можете «передать» входные и выходные параметры типа UDT в параметры вызываемого блока.

Параметры блоков типа TIMER

Вы можете использовать параметры блоков типа TIMER во всех функциях, как описано в главе 7 «Таймеры». При запуске таймера значение времени также может быть параметром блока типа S5TIME.

Параметры блоков типа COUNTER

Вы можете использовать параметры блоков типа COUNTER во всех функциях, как описано в главе 8 «Счетчики». При установке счетчика его значение может быть также параметром блока типа WORD.

Параметры блоков типа BLOCK_DB

Вы можете переслать блок данных через параметр блока типа BLOCK_DB. Вызовите этот блок данных с помощью катушки / блочного элемента OPN путем маркирования катушки / блочного элемента OPN формальными параметрами.

При открытии блока данных через параметр блока CPU всегда использует регистр глобального блока данных (регистр DB).

Параметры блоков типа BLOCK_FC

Можно передать функцию FC через параметр блока типа BLOCK_FC. Вызовите эту функцию при помощи катушки / блочного элемента CALL. Вы можете использовать катушку / блочный элемент CALL с формальным параметром и предшествующей логической операцией или без нее, если в данный момент вы программируете функциональный блок.

Если вы используете катушку / блочный элемент CALL с формальными параметрами в функции, то предшествующей логической операции не допускается (только абсолютный вызов).

Функция FC, передаваемая через параметр блока, не должна иметь параметров.

Параметры блоков типа BLOCK_FB

Вы можете передать функциональный блок FB через параметр блока типа BLOCK_FB. Прямой доступ к параметрам блоков типа BLOCK_FB с использованием функций LAD и FBD невозможен.

Функциональный блок FB, пересылаемый посредством параметра, не имеет параметров.

Параметры блоков типа POINTER и ANY

Прямой доступ к параметрам блока типа POINTER и ANY с использованием функций LAD и FBD невозможен.

19.3 Фактические параметры

Когда блок вызывается, вы инициализируете его параметры значениями констант, операндов или переменных, с которыми он будет работать. Эти параметры являются фактическими. Если в программе вы вызываете блок часто, то каждый раз обычно используются различные параметры.

Фактический параметр должен соответствовать по типу данных параметру блока. Для параметра блока типа `BOOL` вы можете применить только двоичный фактический параметр (например, меркерный бит); инициализировать параметр блока типа `ARRAY` вы можете только переменной, являющейся массивом с идентичной размерностью. Таблица 19.3 представляет обзор, какие операнды вы можете использовать в качестве фактических параметров и с какими типами данных.

Таблица 19.3 Инициализация фактическими параметрами

Тип данных параметра блока	Допустимые фактические параметры
Простой тип данных	<ul style="list-style-type: none"> ➤ Простые операнды, полностью адресованные операнды данных, константы ➤ Элементы массивов или компоненты структур простого типа данных ➤ Параметры вызывающего блока ➤ Компоненты параметров вызывающего блока простого типа данных
Сложный тип данных	<ul style="list-style-type: none"> ➤ Переменные или параметры вызывающего блока
<code>TIMER</code> , <code>COUNTER</code> и <code>BLOCK_xx</code>	<ul style="list-style-type: none"> ➤ Таймеры, счетчики и блоки
<code>POINTER</code>	<ul style="list-style-type: none"> ➤ Простые операнды, полностью адресованные операнды данных ➤ Интервальный указатель (<code>range pointer</code>) или <code>DB-</code>указатель
<code>ANY</code>	<ul style="list-style-type: none"> ➤ Переменные любого типа данных ➤ Указатель <code>ANY</code>

При вызове функций вы должны инициализировать фактическими параметрами все параметры блоков.

Когда вызываются функциональные блоки, инициализация фактическими параметрами параметры блоков не является обязательным требованием. Если вы не определяете параметры блока, то в качестве фактических параметров используются (старые) значения, сохраненные в экземплярном блоке данных. Это могут быть, к примеру, значения по умолчанию или значения фактических параметров более раннего вызова. Входные/выходные параметры сложного типа данных не могут быть назначенными значениями по умолчанию и параметрическими типами. Вы должны обеспечить эти параметры блоков фактическими параметрами, по крайней мере, при первом вызове.

Вы также можете использовать прямой доступ для обращения к параметрам функционального блока. Так как они расположены в блоке данных, вы можете оперировать параметрами блока как операндами данных. Пример: Функциональный блок с экземпляром блока данных «*Station_1*» управляет бинарным выходным параметром с именем «*Up*». После обработки в функциональном блоке (после его вызова), вы можете считать параметр по символическому адресу «*Station_1*». *Up* в отсутствие инициализированного выходного параметра.

Инициализация параметров блоков с простыми типами данных

Фактические параметры, приведенные в таблице 19.4, допускаются в качестве фактических параметров простого типа данных.

Таблица 19.4 Фактические параметры простого типа данных

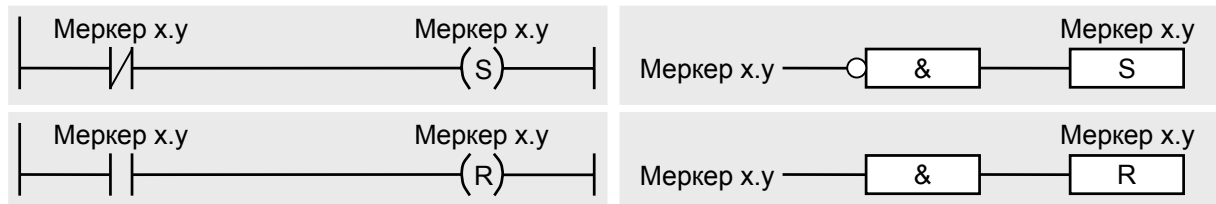
Операнды	Разрешено с			Двоичный операнд или символическое имя	Числовой операнд или символическое имя
	IN	I_O	OUT		
Входы (образ процесса)	x	x	x	I y.x	IB y, IW y, ID y
Выходы (образ процесса)	x	x	x	Q y.x	QB y, QW y, QD y
Меркерная память	x	x	x	M y.x	MB y, MW y, MD y
Периферийные входы	x	-	-		PIB y, PIW y, PID y
Периферийные выходы	-	-	x		PQB y, PQW y, PQD y
Глобальные данные					
Частичная адресация	x	x	x	-	DBB y, DBW y, DBD y
Полная адресация	x	x	x	DB z.DBX y.x	DB z.DBB y и т.д.
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	-	-	-	Все числовые константы
Компоненты ARRAY или STRUCT	x	x	x	Полное имя компонента	Полное имя компонента

x = номер бита, y = адрес байта, z = номер блока данных

Вы можете назначить операндам входа, выхода и меркера либо абсолютный, либо символический адрес. Входные операнды должны использоваться только для входных параметров, а выходные операнды – для выходных параметров (однако это не обязательно). Операнды памяти меркеров подходят для всех описанных в программе типов. Вы должны сочетать периферийные входы только с входными параметрами и периферийные выходы только с выходными параметрами.

В LAD и FBD булевские константы TRUE и FALSE используются для установки значений по умолчанию в описании параметров блоков, статических локальных данных или операндов данных. Инициализация параметров блоков этими константами не допускается. Вы также не можете создавать эти константы на контакте (LAD) или

на входе функции (FBD). Если вы хотите инициализировать параметры блока значениями TRUE или FALSE, то вам потребуется переменная, которой постоянно присвоено сигнальное состояние «1» или «0», и которую вы используете вместо булевских констант. Меркеры в этом случае являются несомненными кандидатами для переменных. Следующий пример демонстрирует вам, как можно устанавливать меркеры в сигнальное состояние «1» или «0» на постоянной основе, скажем при запуске.



Частично адресуемые биты данных в качестве фактических параметров в LAD и FBD не допускаются. Причиной этому служит тот факт, что программный редактор пересылает сигнальное состояние каждого двоичного фактического операнда в бит временных локальных данных перед фактическим вызовом. Поэтому в LAD и FBD возможно инициализировать двоичный вход блока, используя логическую операцию. Во время копирования перед вызовом блока назначение блока данным битам данных теряется. Даже если вы открываете «корректный» блок данных в программе вызванного блока перед обращением к параметру блока, занятому частично адресованным битом данных, то сканироваться будет только предварительно созданная копия сигнального состояния.

Это относится и к двоичным входным/выходным параметрам и выходным параметрам: здесь также назначение с частично адресованными битами данных не разрешается (редактор программ не отклонит такое назначение, но в большинстве случаев это приведет к ошибочным функциям).

Когда используются частично адресованные операнды данных для числовых параметров, вы должны для себя отметить, что при обращении к параметрам блока (в вызванном блоке) открытый в текущий момент блок данных также является «корректным» блоком данных. Так как редактор в определенных обстоятельствах (например, во время вызова блока или если ранее произошло обращение к параметрам блока) может незаметно изменить блок данных, использование частично адресованных числовых операндов данных не рекомендуется. По этой причине применяйте только полностью адресованные данные.

Временные локальные данные обычно адресуются символически. Они расположены в L-стеке вызывающего блока и описываются (объявляются) в вызывающем блоке.

Если вызывающий блок является функциональным блоком, вы можете также использовать его статические локальные данные в качестве фактических параметров (см. параграф 19.4 ««Передача» параметров блоков»). Статические данные обычно адресованы символически; если вы хотите присвоить абсолютный адрес через операнды DI, помните о сказанном в параграфе 18.2.4 «Особые замечания по адресации данных».

Имея параметр блока типа `BOOL`, можно применять константу `TRUE` (сигнальное состояние «1») или `FALSE` (сигнальное состояние «0»), а с использованием параметра блока числового типа данных вы можете применять все константы, соответствующие типу данных. Инициализация с применением констант допускается только для входных параметров.

Вы также можете инициализировать параметр блока простого типа данных элементами массивов или компонентами структур, если последние относятся к тому же типу данных, что и параметр блока.

Инициализация параметров блоков сложных типов данных

Все параметры блока могут быть сложного типа. Переменные того же типа могут использоваться в качестве фактических операндов.

Для инициализации параметров блоков типа `DT` или `STRING` допускаются отдельные переменные или компоненты массивов или структур того же типа.

Переменные `STRING` могут иметь различную длину. Если вы задали размер `STRING` при описании входного или выходного параметра функционального блока, то редактор резервирует указанное пространство в экземплярном блоке данных; если длина не указана, то для переменной `STRING` резервируется 256 байт. Максимальный размер переменной `STRING`, который вы задаете в разделе описаний, должен совпадать с размером фактического и формального параметров. Исключение: в случае функций `FC` при объявлении переменной `STRING` вы либо не задаете длину, либо указываете размер 254 байта; здесь в качестве фактических параметров вы можете использовать переменные `STRING` любой длины.

Для инициализации параметров блоков типа `ARRAY` или `STRUCT` допускаются переменные такой же структуры.

Инициализация параметров блоков определенного пользователем типа данных

Для сложных или громоздких структур данных рекомендуется использование определяемых пользователем типов данных (`UDT`). Сначала вы должны описать `UDT`, затем сможете использовать его, например, для генерирования переменной в блоке данных или объявления параметра блока. Впоследствии переменная может служить для инициализации параметра блока. Здесь также фактический параметр (переменная) должен быть того же типа данных (иметь тот же `UDT`), что и параметр блока.

Инициализация параметров блоков типов данных `TIMER`, `COUNTER` и `BLOCK_xx`

Параметр блока типа `TIMER` инициализируется при помощи таймера, а параметр блока типа `COUNTER` – счетчика. Для параметров блоков параметрических типов

BLOCK_FC и BLOCK_FB применяются блоки без их собственных параметров. Инициализация BLOCK_DB осуществляется с использованием блока данных.

Параметры блоков типов TIMER, COUNTER и BLOCK_хх должны быть входными параметрами.

Инициализация параметров блоков типа POINTER

Указатели (константы) и операнды допустимы для использования в качестве параметров блоков параметрического типа POINTER. Эти указатели являются либо интервальными указателями (range pointer), либо DB-указателями. В первом случае это 32-битные указатели, во втором – 48-битные. Операнды простых типов данных могут быть также полностью адресованными операндами данных.

Для функциональных блоков выходные параметры типа POINTER не допускаются.

Инициализация параметров блоков типа ANY

В качестве параметров блоков параметрического типа ANY могут использоваться переменные всех типов. При программировании вызываемого блока определяется, какие переменные (операнды или типы данных) должны быть применены к параметрам блока или какие переменные допустимы. Вы также можете определить константу в формате ANY-указателя “P#[data block.]Operand Datatype Number” (“P#[блок данных.]Операнд Тип данных Номер”) и определить таким образом абсолютно адресованную область.

Для функциональных блоков выходные параметры типа ANY не могут быть использованы.

19.4 «Передача» параметров блоков

«Передача» параметров блоков – это особая форма доступа к параметрам блоков и их инициализации. Параметры вызывающего блока «передаются» в параметры вызываемого блока. Формальный параметр вызывающего блока, таким образом, становится фактическим параметром вызванного блока.

Вообще, в этом случае также тип фактического параметра должен совпадать с типом формального параметра (то есть соответствующие параметры блока должны согласовываться по их типам). Кроме того, вы можете применить входной параметр вызывающего блока только к входному параметру вызываемого блока и, аналогично, выходной параметр к выходному параметру. Входной/выходной параметр вызывающего блока вы можете применить ко всем типам описания вызываемого блока.

Имеются ограничения, связанные с типами данных, которые вызваны различиями в хранении параметров блоков функциями и функциональными блоками. Параметры блоков простого типа данных могут передаваться беспрепятственно в соответствии с информацией предыдущего параграфа. Входные и выходные параметры сложного типа могут быть переданы, только если вызывающий блок является функциональным блоком. Параметры блоков типов TIMER, COUNTER и BLOCK_xx могут передаваться из одного входного параметра в другой только в том случае, если вызывающий блок является функциональным блоком. Эти утверждения представлены в таблице 19.5.

Таблица 19.5 Возможные комбинации при передаче параметров блоков

Вызывающий → вызванный Тип описания	FC вызывает FC			FB вызывает FC			FC вызывает FB			FB вызывает FB		
	Е	С	Р	Е	С	Р	Е	С	Р	Е	С	Р
Input → Input	x			x	x		x		x	x	x	x
Output → Output	x			x	x		x			x	x	
In/out → Input	x						x					
In/out → Output	x			x			x			x		
In/out → In/out	x			x						x		

Е – простые (Elementary) типы данных

С – сложные (Complex) типы данных

Р – параметрические (Parameter) типы TIMER, COUNTER и BLOCK_xx

19.5 Примеры

19.5.1 Пример «Ленточный транспортер конвейера»

Пример показывает передачу сигнальных состояний через параметры блоков. Для этой цели мы используем функцию системы управления ленточным транспортером конвейера, описание которой содержится в главе 5 «Функции для работы с памятью». Система управления транспортером конвейера будет запрограммирована в функциональном блоке, и все входы и выходы должны быть связаны с параметрами блока, что даст возможность вызывать функциональный блок повторно (для нескольких транспортеров конвейера).

На рисунке 19.3 показаны входные и выходные параметры функционального блока, а также используемые статические и временные локальные данные.

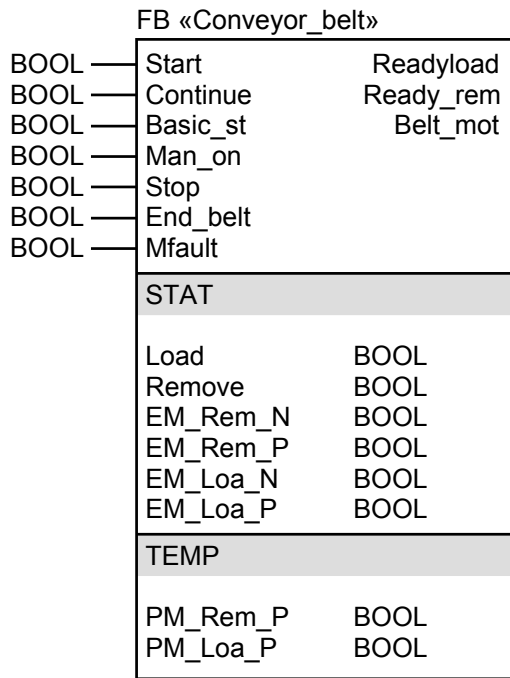
Распределение параметров в этом случае вполне простое. Все двоичные операнды, которые были входами, стали входными параметрами, все выходы стали выходными параметрами, а все меркеры стали статическими локальными данными. Вы также заметили, что имена были незначительно изменены, потому что для внутриблочных (локальных) переменных допустимы только буквы, цифры и знак подчеркивания.

Функциональный блок «Conveyor_Belt» будет управлять двумя транспортерами конвейера. Для этого он будет вызываться дважды; первый раз с входами и выходами транспортера конвейера 1 (conveyor belt 1), второй раз с входами и выходами транспортера конвейера 2 (conveyor belt 2).

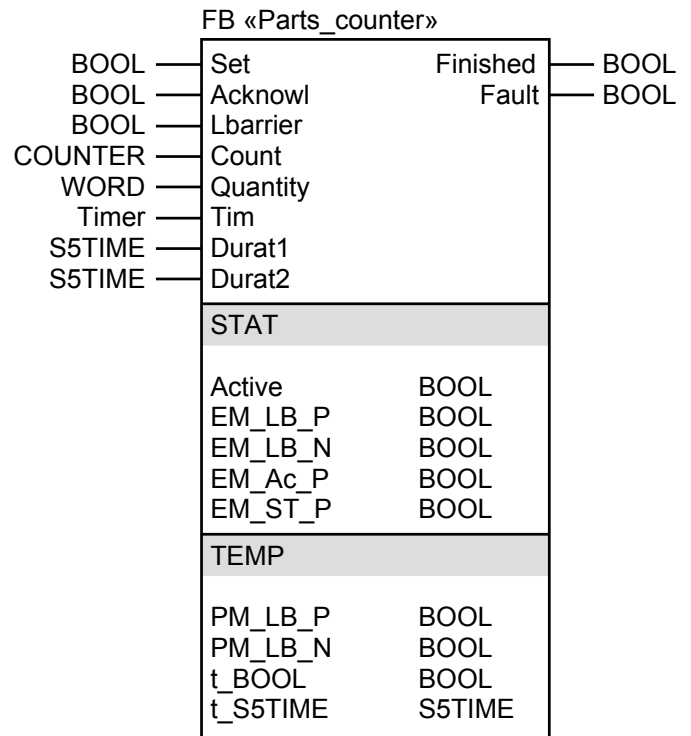
При каждом вызове функциональному блоку требуется экземплярный блок данных, где он хранит данные транспортера конвейера. Блок данных транспортера конвейера 1 будет называться «BeltData1», а транспортера конвейера 2 – «BeltData2».

Выполненный пример программирования вы можете найти в программе «Conveyor Example» («Пример «Конвейер»») в библиотеке «LAD_Book» или «FBD_Book» на прилагаемой к книге дискете. В нем продемонстрировано программирование функционального блока FB 21 с входными параметрами, выходными параметрами и статическими локальными данными. В качестве экземплярных блоков можно использовать любые блоки данных; в примере DB 21 используется для «BeltData1», и DB 22 – для «BeltData2». В таблице символов эти блоки данных имеют тип данных функционального блока (в примере FB 21, если «Conveyor_Belt» является символом для FB 21).

При вызове функционального блока вы можете использовать входы и выходы из таблицы символов в качестве фактических параметров. В тех случаях, где эти глобальные символы содержат специальные знаки, вы должны в программе заключить эти символы в кавычки. Таблица символов разработана в этой главе для всех трех примеров (таблица 19.7 в конце главы).

**Рисунок 19.3**

Функциональный блок для примера
«Ленточный транспортер конвейера»

**Рисунок 19.4**

Функциональный блок для примера
«Счетчик деталей»

19.5.2 Пример «Счетчик деталей»

Пример демонстрирует обработку параметров боков простого типа данных. Основой функции является пример «Счетчик деталей» из главы 8 «Счетчики». Та же функция реализуется здесь как функциональный блок со всеми глобальными переменными, описанных либо как параметры блока, либо как статические локальные данные. Управление таймерами и счетчиками осуществляется здесь через отдельные элементы.

На рисунке 19.4 показаны входные и выходные параметры создаваемого функционального блока, а также используемые статические и временные локальные данные.

Таймеры и счетчики передаются через параметры блока типа TIMER и COUNTER. Эти параметры блока должны быть входными параметрами. Начальные значения счетчика (Quantity) и таймера (Durat1 и Durat2) также могут быть переданы как параметры блока; типы данных параметров блока здесь соответствуют типам фактических параметров.

Меркеры фронта хранятся в статических локальных данных, а меркеры импульса хранятся во временных локальных данных.

Пример программы вы можете найти в библиотеке «LAD_Book» или «FBD_Book» в разделе «Conveyor Example» («Пример «Конвейер»») на прилагаемой к книге дискете. Он содержит функциональный блок FB 22 «Parts_counter» и соответствующий эк-

земпляр блока данных «CountDat». Вы можете использовать входы, выходы, таймер и счетчик из таблицы символов (таблица предыдущего примера) в качестве фактических параметров при вызове функционального блока.

19.5.3 Пример «Устройство подачи»

Те же функции, описанные в двух предыдущих примерах, также могут быть вызваны как локальные экземпляры. В нашем примере это означает, что мы программируем функциональный блок, называемый «Feed» («Подача»), который будет управлять четырьмя транспортерами конвейера и считать проходящие детали. В этом функциональном блоке FB «Conveyor_Belt» вызывается четыре раза, а FB «Parts_counter» – один раз. Вызов не каждый раз происходит со своим собственным экземпляром блока данных, но вызываемые FB должны хранить их данные в экземплярном блоке данных функционального блока «Feed».

На рисунке 19.5 показано, как связаны отдельные элементы управления транспортера конвейера (FB «Parts_counter» здесь не представлен). Стартовый сигнал (Start) подается на вход *Start* контроллера транспортера 1 (belt 1), выход *Ready_rem* соединен с входом *Start* транспортера 2 (belt 2) и так далее. Наконец, выход *Ready_rem* транспортера 4 (belt 4) подключен к выходу *Remove* блока «Feed». Такая же последовательность сигналов проходит в обратном направлении от *Remove* через *Continue* и *Readyload* к *Load*.

Belt_mot, *Lbarr* и */Mfault* являются отдельными сигналами транспортера конвейера; *Reset*, *Start* и *Stop* управляют всеми транспортерами конвейера посредством *Basic_st*, *Man_on* и *Stop*.

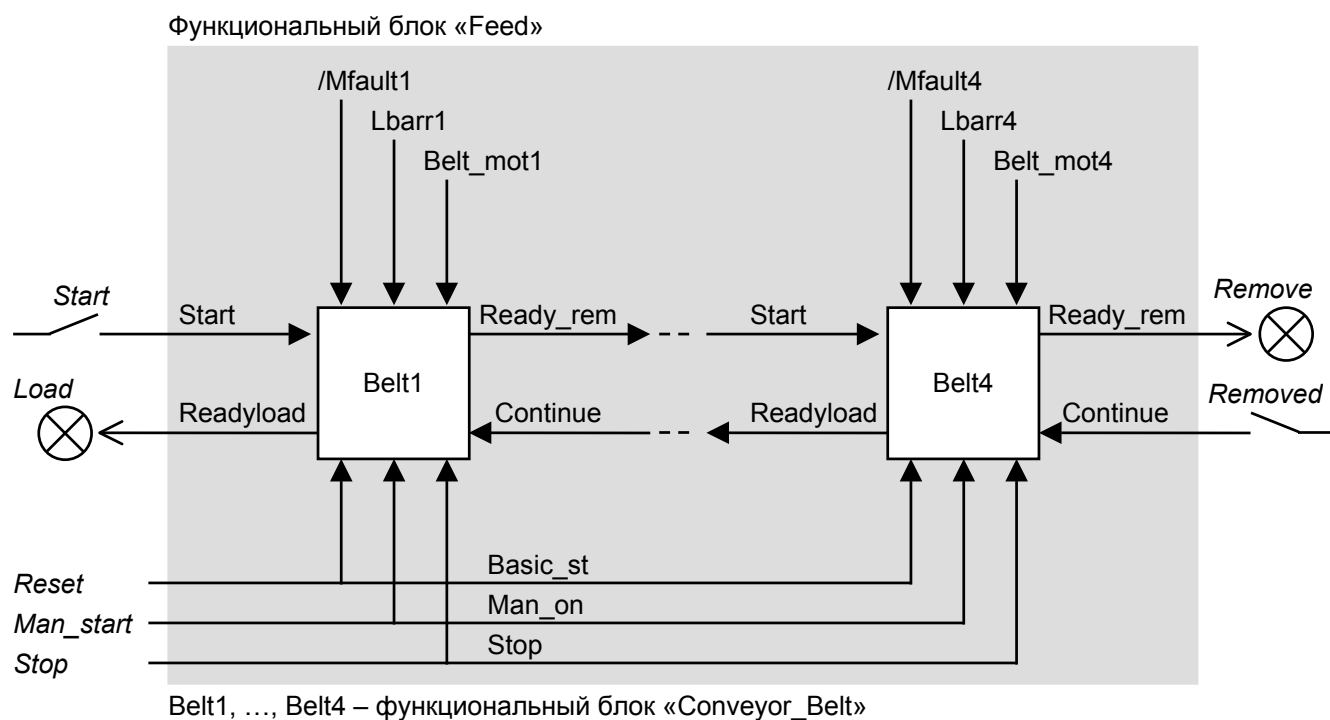


Рисунок 19.5 Пример программирования «Устройство подачи»

Следующая программа для функционального блока «Feed» разработана точно таким же способом. Входные и выходные параметры функционального блока могут быть взяты из рисунка 19.5. Кроме того, числовые значения счетчика деталей *Quantity*, *Durat1* и *Durat2* здесь спроектированы как входные параметры. Мы описываем данные отдельных элементов управления транспортером конвейера и данные счетчика деталей в статических локальных данных точно так же, как и для определенного пользователем типа данных, то есть с именем и типом данных. Переменная «Belt1» предназначена для приема структуры данных функционального блока «Conveyor_Belt», как и переменная «Belt2» и так далее; переменная «Check» принимает структуру данных функционального блока «Parts_counter».

Программа в функциональном блоке начинается с инициализации сигналов, общих для всех транспортеров конвейера. В нашем случае мы используем тот факт, что параметры функциональных блоков, вызываемых как локальные экземпляры, являются статическими локальными данными в текущем блоке и могут рассматриваться как таковые. Параметр блока *Man_on* в текущем функциональном блоке управляет входными параметрами *Man_on* всех четырех элементов управления транспортерами конвейера с использованием одиночного элемента назначения (присваивания). Таким же образом обрабатываются сигналы *Stop* и *Reset*. И теперь элементы управления транспортерами конвейера инициализированы общими сигналами. (Вы, конечно, можете также инициализировать эти входные параметры при вызове функционального блока.)

Последующие вызовы функциональных блоков для элемента управления транспортером конвейера содержат параметры блока только для отдельных сигналов для каждого транспортера конвейера и соединение с параметрами блока «Feed». Отдельные сигналы – это световые барьеры, команды для мотора транспортера и сбой мотора. (Мы здесь используем тот факт, что при вызове функционального блока могут быть инициализированы не все параметры блока.) Мы программируем соединения между отдельными контроллерами транспортеров, используя элементы назначения (присваивания).

FB «Parts_counter» вызывается как локальный экземпляр, даже если он не имеет близкого соединения с сигналами элементов управления транспортерами конвейера. Экземплярный блок данных блока «Feed» хранит данные FB.

Входные параметры *Quantity*, *Durat1* и *Durat2* блока «Feed» должны быть установлены только один раз. Это может быть сделано с использованием значений по умолчанию (как в примере) или в процедуре рестарта в ОВ 100 (например, через непосредственное назначение, если эти три параметра рассматриваются как глобальные данные).

Таблица 19.6 Раздел описаний FB «Feed»

Address (Адрес)	Declaration (Описание)	Name (Имя)	Type (Тип)	Initial value (Начальное значение)	Comment (Комментарий)
0.0	in	Start	BOOL	FALSE	Start conveyor belts / Запуск транспортеров конвейера
0.1	in	Removed	BOOL	FALSE	Parts have been removed from belt / Детали удалены с транспортера
0.2	in	Man_start	BOOL	FALSE	Start conveyor belts manually / Запуск транспортеров конвейера вручную
0.3	in	Stop	BOOL	FALSE	Stop conveyor belts / Останов транспортеров конвейера
0.4	in	Reset	BOOL	FALSE	Set control to the basic setting / Установить элемент управления в основное состояние (вернуть базовые установки)
2.0	in	Count	COUNTER		Counter for the parts / Счетчик деталей
4.0	in	Quantity	WORD	W#16#200	Number of parts / Количество деталей
6.0	in	Tim	TIMER		Timer for the monitor / Таймер монитора (устройства наблюдения)
8.0	in	Durat1	S5TIME	S5T#5s	Monitoring time for parts / Время наблюдения для деталей
10.0	in	Durat2	S5TIME	S5T#10s	Monitoring time for gap / Время наблюдения для промежутка
12.0	out	Load	BOOL	FALSE	Load new parts onto belt / Помещение новых деталей на транспортер
12.1	out	Remove	BOOL	FALSE	Remove parts from belt / Удаление деталей с транспортера
	in_out				
14.0	stat	Belt1	Conveyor_belt		Control for belt1 / Элемент управления транспортером 1
20.0	stat	Belt2	Conveyor_belt		Control for belt2 / Элемент управления транспортером 2
26.0	stat	Belt3	Conveyor_belt		Control for belt3 / Элемент управления транспортером 3
32.0	stat	Belt4	Conveyor_belt		Control for belt4 / Элемент управления транспортером 4
38.0	stat	Check	Parts_counter		Control for counting and monitoring / Управление подсчетом и наблюдением
	temp				

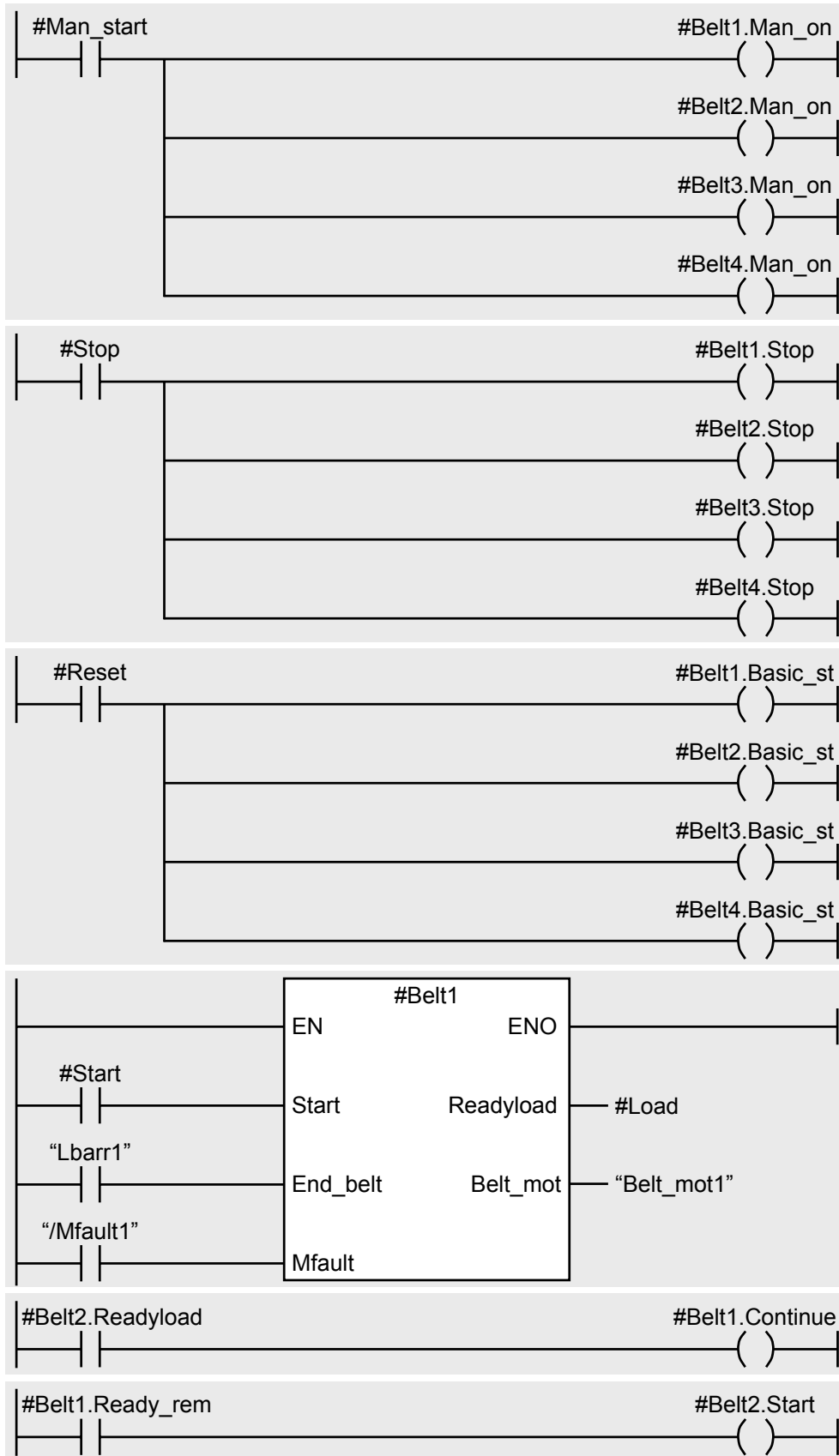
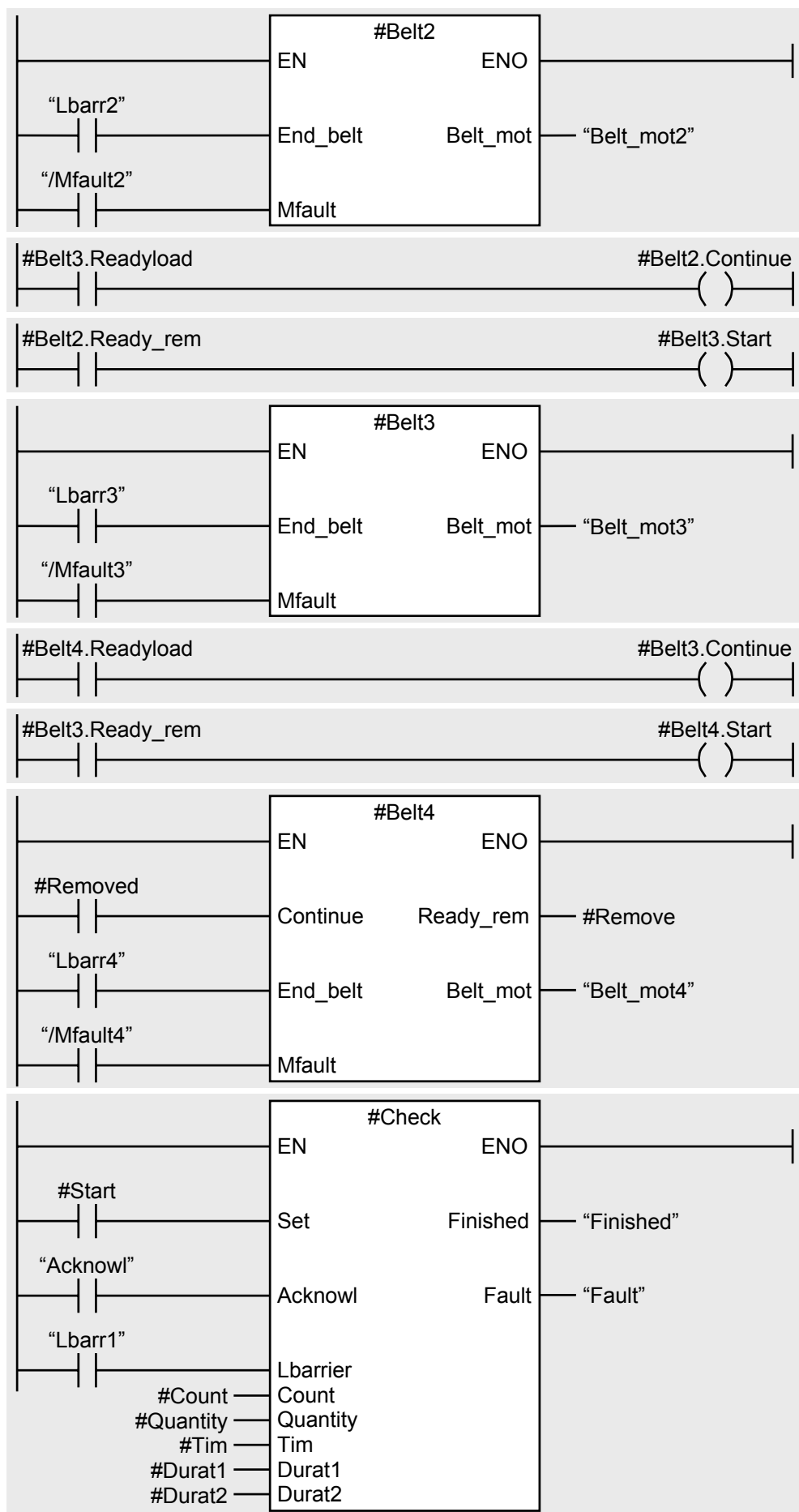


Рисунок 19.6
 Программа для приме-
 ра «Устройство пода-
 чи» (LAD)

Продолжение на следующей странице



Продолжение

Рисунок 19.6
Программа для приме-
ра «Устройство пода-
чи» (LAD)

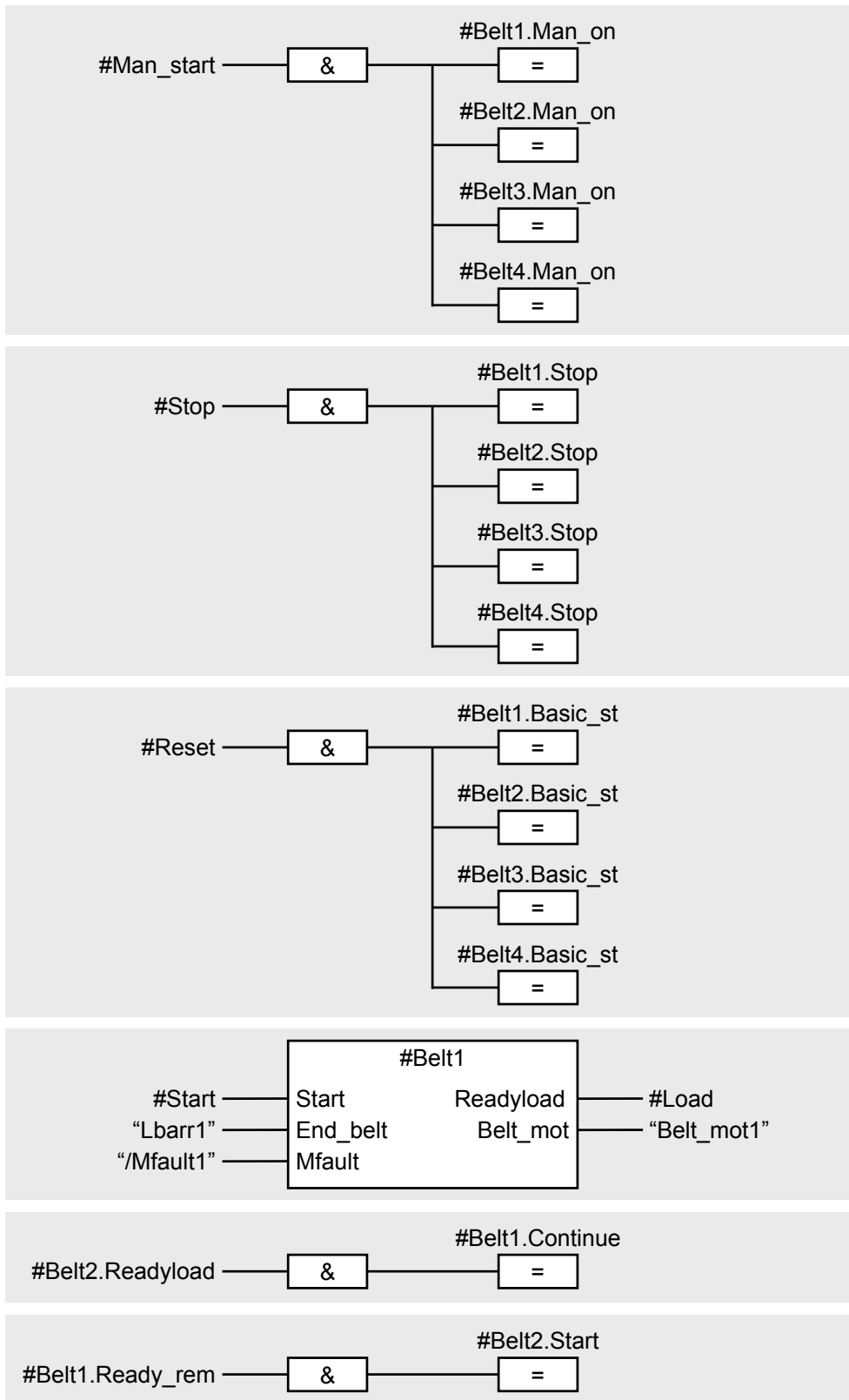
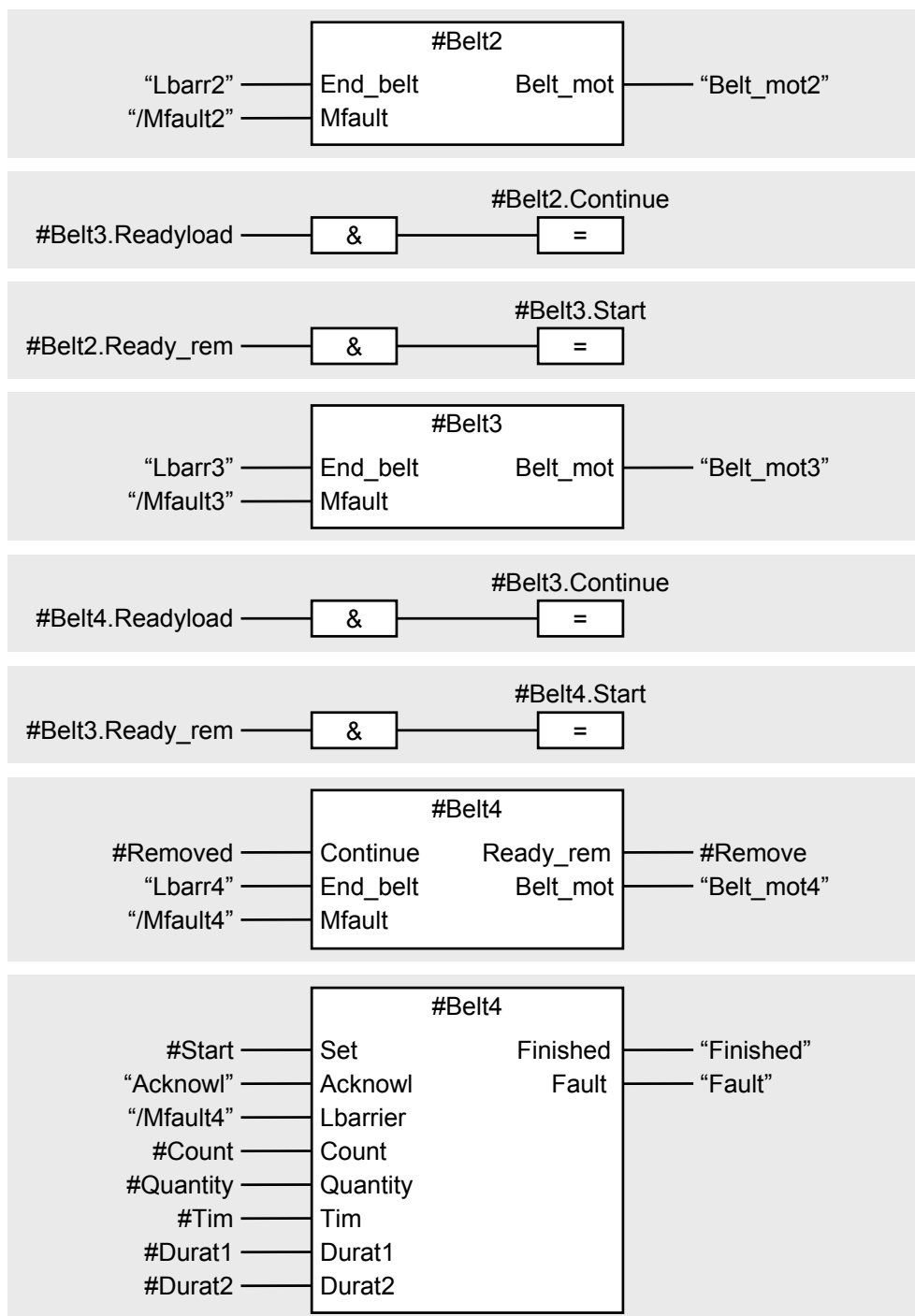


Рисунок 19.7 Программа для примера «Устройство подачи» (FBD)

Продолжение на следующей странице



Продолжение

Рисунок 19.7 Программа для примера «Устройство подачи» (FBD)

Таблица 19.7 Таблица символов для примеров «Ленточный транспортер конвейера», «Счетчик деталей» и «Устройство подачи»

Symbol (Символ)	Address (Адрес)	Data Type (Тип данных)	Comment (Комментарий)
Conveyor_belt	FB 21	FB 21	Conveyor belt control / Элемент управления транспортером конвейера
BeltData1	DB 21	FB 21	Data for conveyor belt1 / Данные для транспортера конвейера1
BeltData2	DB 22	FB 21	Data for conveyor belt2 / Данные для транспортера конвейера2
Parts_counter	FB 22	FB 22	Counter control and monitor / Элемент управления счетчиком и монитор (устройство наблюдения)
CounterDat	DB 29	FB 22	Data for parts counter / Данные для счетчика деталей
Feed	FB 20	FB 20	Feed with several belts / Устройство подачи с несколькими транспортерами
FeedDat	DB 20	FB 20	Data for Feed / Данные для устройства подачи
Cycle	OB 1	OB 1	Main program, cyclic execution / Главная программа, циклическое выполнение
Basic_st	I 0.0	BOOL	Set controller to the basic state / Установка контроллера в основное состояние
Man_on	I 0.1	BOOL	Switch on conveyor belt motors / Включение моторов транспортеров конвейера
/Stop	I 0.2	BOOL	Stop conveyor belt motors (zero active) / Останов моторов транспортеров конвейера (нулевая активность)
Start	I 0.3	BOOL	Start conveyor belt / Запуск транспортера конвейера
Continue	I 0.4	BOOL	Acknowledgement that parts have been removed / Уведомление о том, что детали удалены
Acknowl	I 0.6	BOOL	Acknowledge fault / Подтверждение ошибки
Set	I 0.7	BOOL	Set counter, activate monitor / Установка счетчика, активизация монитора
Lbarr1	I 1.0	BOOL	(Light barrier) «End of belt» sensor signal for conveyor belt 1 / (Световой барьер) сигнал датчика «Конец транспортера» для транспортера конвейера 1
Lbarr2	I 1.1	BOOL	(Light barrier) «End of belt» sensor signal for conveyor belt 2 / (Световой барьер) сигнал датчика «Конец транспортера» для транспортера конвейера 2
Lbarr3	I 1.2	BOOL	(Light barrier) «End of belt» sensor signal for conveyor belt 3 / (Световой барьер) сигнал датчика «Конец транспортера» для транспортера конвейера 3
Lbarr4	I 1.3	BOOL	(Light barrier) «End of belt» sensor signal for conveyor belt 4 / (Световой барьер) сигнал датчика «Конец транспортера» для транспортера конвейера 4
/Mfault1	I 2.0	BOOL	Motor protection switch conveyor belt 1 (zero active) / Включение защиты мотора транспортера конвейера 1 (нулевая активность)
/Mfault2	I 2.1	BOOL	Motor protection switch conveyor belt 2 (zero active) / Включение защиты мотора транспортера конвейера 2 (нулевая активность)
/Mfault3	I 2.2	BOOL	Motor protection switch conveyor belt 3 (zero active) / Включение защиты мотора транспортера конвейера 3 (нулевая активность)

Таблица 19.7 Продолжение

Symbol (Символ)	Address (Адрес)	Data Type (Тип данных)	Comment (Комментарий)
/Mfault4	I 2.3	BOOL	Motor protection switch conveyor belt 4 (zero active) / Включение защиты мотора транспортера конвейера 4 (нулевая активность)
Readyload	Q 4.0	BOOL	Load new parts onto belt / Загрузка новых деталей на транспортер
Ready_rem	Q 4.1	BOOL	Remove parts from belt / Удаление деталей с транспортера
Finished	Q 4.2	BOOL	Number of parts reached / Достигнутое число деталей
Fault	Q 4.3	BOOL	Monitor activated / Устройство наблюдения активировано
Belt_mot1	Q 5.0	BOOL	Switch on belt motor for conveyor belt 1 / Включение мотора транспортера конвейера 1
Belt_mot2	Q 5.1	BOOL	Switch on belt motor for conveyor belt 2 / Включение мотора транспортера конвейера 2
Belt_mot3	Q 5.2	BOOL	Switch on belt motor for conveyor belt 3 / Включение мотора транспортера конвейера 3
Belt_mot4	Q 5.3	BOOL	Switch on belt motor for conveyor belt 4 / Включение мотора транспортера конвейера 4
Quantity	MW 4	WORD	Number of parts / Количество деталей
Durat1	MW 6	S5TIME	Monitoring time for light barrier covered / Время наблюдения для закрытого светового барьера
Durat2	MW 8	S5TIME	Monitoring time for light barrier not covered / Время наблюдения для открытого светового барьера
Count	C 1	COUNTER	Counter for parts / Счетчик деталей
Monitor	T 1	TIMER	Timer for monitor / Таймер устройства наблюдения

Обработка программы

В этом разделе книги обсуждаются различные методы обработки программы.

Главная программа (main program) выполняется циклически. После каждого прохода программы CPU возвращается к началу программы и выполняет ее снова. Это «стандартный» метод обработки программ PLC.

Многочисленные системные функции поддерживают работу системных служб, таких как управление таймером реального времени или коммуникация через шинные системы. В отличие от статических установок, проводимых при параметризации CPU, системные функции являются динамическими, и это можно использовать во время исполнения программы.

Главная программа может быть временно приостановлена для выполнения **обслуживания прерываний (interrupt servicing)**. Различные типы прерываний (аппаратные прерывания, циклические прерывания, прерывания по времени суток, прерывания задержки времени, мультипроцессорные прерывания) делятся на приоритетные классы, приоритет обработки которых вы можете сами, в большой степени, определять. Система обслуживания прерываний позволяет вам немедленно реагировать на сигналы, приходящие от управляемого процесса, или выполнять периодические контрольные процедуры независимо от времени обработки главной программы.

Перед запуском главной программы CPU иницирует **программу запуска (start-up program)**, в которой вы можете задавать установки относительно обработки программы, определять для переменных значения по умолчанию или параметризовать модули.

Обработка ошибок (error handling) также является частью процесса обработки программы. В STEP 7 сделаны различия между синхронными ошибками, возникающие во время обработки оператора, и асинхронными ошибками, которые могут быть обнаружены независимо от обработки программы. В обоих случаях вы можете приспособить подпрограмму обработки ошибок к вашим требованиям.

20 Главная программа

Структура программы; управление циклом сканирования; время отклика; программные функции; мультивычислительные операции; обмен данными с использованием системных функций; стартовая информация

21 Обработка прерываний

Аппаратные прерывания; циклические прерывания; прерывания по времени суток; прерывания задержки времени; мультипроцессорные прерывания; обработка событий по прерываниям

22 Параметры запуска

Включение электропитания, сброс памяти, способность к сохранению; полный рестарт; «теплый» рестарт; вычисление адреса модуля; параметризация модулей

23 Обработка ошибок

Синхронные ошибки (программные ошибки, ошибки доступа); обработка событий по синхронным ошибкам; асинхронные ошибки; системная диагностика

Содержание главы 20

20	<u>Главная программа</u>	4
20.1	<u>Организация программы</u>	4
20.1.1	<u>Структура программы</u>	4
20.1.2	<u>Организация программы</u>	5
20.2	<u>Управление циклом сканирования</u>	8
20.2.1	<u>Обновление образа процесса</u>	8
20.2.2	<u>Время наблюдения цикла сканирования</u>	10
20.2.3	<u>Минимальное время цикла сканирования, фоновое сканирование</u>	11
20.2.4	<u>Время отклика</u>	13
20.2.5	<u>Стартовая информация</u>	14
20.3	<u>Программные функции</u>	17
20.3.1	<u>Таймер реального времени</u>	17
20.3.2	<u>Чтение системного таймера</u>	18
20.3.3	<u>Счетчик учета рабочего времени</u>	18
20.3.4	<u>Сжатие памяти CPU</u>	20
20.3.5	<u>Ожидание и останов</u>	20
20.3.6	<u>Режим мультипроцессорной обработки</u>	21
20.4	<u>Коммуникация через распределенные входы/выходы</u>	23
20.4.1	<u>Адресация распределенных входов/выходов</u>	23
20.4.2	<u>Конфигурирование распределенных входов/выходов</u>	28
20.4.3	<u>Системные функции для распределенных входов/выходов</u>	42
20.5	<u>Коммуникация глобальных данных</u>	47
20.5.1	<u>Основы</u>	47
20.5.2	<u>Конфигурирование GD-коммуникации</u>	51
20.5.3	<u>Системные функции для GD-коммуникации</u>	53
20.6	<u>SFC-коммуникация</u>	55
20.6.1	<u>SFC-коммуникация внутри станции</u>	55
20.6.2	<u>Системные функции для обмена данными в станции</u>	56
20.6.3	<u>SFC-коммуникация вне станции</u>	58
20.6.4	<u>Системные функции для SFC-коммуникации вне станции</u>	60
20.7	<u>SFB-коммуникация</u>	65
20.7.1	<u>Основы</u>	65
20.7.2	<u>Двухсторонний обмен данными</u>	67
20.7.3	<u>Односторонний обмен данными</u>	69
20.7.4	<u>Передача данных для печати</u>	71
20.7.5	<u>Функции управления</u>	71
20.7.6	<u>Функции наблюдения</u>	73

20 Главная программа

Главная программа – это циклически сканируемая программа пользователя; циклическое сканирование является «нормальным» способом выполнения программы в программируемых логических контроллерах. Подавляющее большинство систем управления используют только этот вид выполнения программ. Если применяется событийное программное сканирование, то в большинстве случаев только как дополнение к главной программе.

Главная программа вызывается в организационном блоке ОВ 1. Она выполняется на низшем приоритетном уровне и может быть прервана любым другим типом программной обработки. Переключатель режимов на передней панели CPU должен быть в положении RUN или RUN-P. В положении RUN-P CPU может программироваться через устройство программирования. В позиции RUN вы можете вынуть ключ, и никто не сможет изменить операционный режим без соответствующей авторизации; когда переключатель режимов находится в положении RUN, возможно только чтение программы.

20.1 Организация программы

20.1.1 Структура программы

Анализ комплексную задачу автоматизации означает ее подразделение на меньшие задачи или функции в соответствии со структурой управляемого процесса. Затем вы устанавливаете отдельные задачи в результате разделения процесса путем определения функций и задания интерфейсных сигналов для процесса или других задач. Такое разбиение на отдельные задачи может быть осуществлено в вашей программе. Таким образом, структура вашей программы соответствует подразделению задачи автоматизации.

Разделенная на части пользовательская программа может быть более легко сконфигурирована и может программироваться секционно (даже несколькими людьми в случае очень больших программ). И, наконец, но не менее важно, разбиение программы упрощает процесс ее отладки, обслуживание и поддержка.

Структурирование программы пользователя зависит от ее размеров и функции. Различаются три различных «метода»:

В линейной программе вся главная программа находится в организационном блоке ОВ 1. Каждая текущая цепь расположена в отдельной сети. STEP 7 нумерует сети последовательно. При редактировании и отладке вы можете обратиться к любой сети непосредственно по ее номеру.

Разделенная на части программа в основе своей является линейной программой, которая поделена на блоки. Причинами деления программы могут быть ее слишком большой размер для организационного блока ОВ 1, или то, что вы хотите сделать ее более читаемой. Блоки вызываются последовательно. Вы также можете разбить про-

грамму в другом блоке точно так же, как в организационном блоке ОВ 1. Этот метод позволяет вам вызывать соответствующие, связанные с процессом функции для обработки из одного и того же блока. Преимущество этой структуры программы в том, что, хотя программа линейная, ее можно отладить по частям (просто отменяя или добавляя вызовы блоков).

Структурированная программа используется, когда концептуальная формулировка особенно обширна, когда вы хотите повторно использовать программные функции, или когда должны быть решены комплексные задачи. Структурирование означает разбиение программы на разделы (блоки), которые представляют собой автономные функции, или имеют специальное функциональное назначение, и которые обмениваются с другими блоками наименьшим возможным количеством сигналов.

Назначение каждому разделу программы специальной (связанной с процессом) функции позволит создать легко читаемые блоки с простыми интерфейсами с другими блоками при программировании.

Языки программирования LAD и FBD поддерживают структурное программирование с использованием функций, с помощью которого вы можете создавать «блоки» (автономные программные разделы). Глава 3 «Программа SIMATIC S7» содержит параграф «Блоки», где рассматриваются различные виды блоков и их применение. Подробную информацию о функциях для вызова и завершения блоков вы можете найти в главе 18 «Функции блоков». Блоки получают для обработки сигналы и данные через интерфейс вызова (параметры блоков) и передают результаты через тот же интерфейс. Возможности по передаче параметров детально обсуждаются в главе 19 «Параметры блоков».

20.1.2 Организация программы

Организация программы определяет возможность и порядок обработки центральным процессором (CPU) блоков, которые вы создали. Для организации вашей программы вы должны запрограммировать вызовы блоков в нужной последовательности в блоках более высокого уровня. Вы должны выбрать порядок, в котором блоки будут вызываться, с тем, чтобы он отражал связанное с процессом (ориентированное на процесс) или связанное с функциональностью (функционально-ориентированное) подразделение управляемого предприятия.

Глубина вложения

Максимальная глубина вложения для приоритетного класса (для программы в организационном блоке) определяется версией CPU. В CPU 314, к примеру, глубина вложения может достигать до восьми уровней, то есть, начиная с одного организационного блока (уровень глубины вложения 1), вы можете добавить еще семь блоков в «горизонтальном» направлении (это называется «вложение»). Если блоков вызывается больше, CPU переходит в состояние STOP, и выдается ошибка «Block overflow» («Переполнение блоков»). Не забывайте при подсчете уровней глубины вложения

учитывать вызовы системных функциональных блоков (SFB) и системных функций (SFC).

Вызов блока данных, который фактически является только открытием или выбором области данных, на глубину вложения блоков не влияет, также не оказывает воздействия на глубину вложения вызовов нескольких блоков подряд (линейные вызовы блоков).

Практически-ориентированная организация программы

В организационном блоке ОВ 1 вы должны вызывать блоки в главной программе таким образом, чтобы примерно организовать вашу программу. Программа может быть организована либо ориентированной на процесс, либо на функционально-ориентированной основе.

Следующие моменты в обсуждении могут дать только приблизительное, очень общее представление с намерением довести до начинающего специалиста некоторые идеи по программному структурированию и осуществлению его задачи управления. Программисты со стажем имеют достаточный опыт, чтобы организовать программу, удовлетворяющую требованиям имеющейся конкретной задачи управления.

Процессно-ориентированная структура программы непосредственно исходит из структуры управляемого предприятия. Отдельные разделы программы соответствуют отдельным участкам предприятия или стадиям процесса, управление которым требуется наладить. На подчиненном этой приблизительной структуре уровне находятся сканирование конечных выключателей и панелей оператора, а также управление приводами и устройствами отображения (в различных участках предприятия).

Функционально-ориентированная структура программы основана на выполняемой функции управления. Изначально этот метод структурирования программы вообще не принимает во внимание управляемое предприятие. Структура предприятия становится видимой сначала в подчиненных блоках, в то время как функция управления, определяемая приблизительной структурой, разбивается на подзадачи в дальнейшем.

На практике обычно используется гибрид этих двух концепций. На рисунке 20.1 приведен пример: функциональная структура отражена в программе рабочего режима и в программе обработки данных, которая проходит над и вне самого предприятия. Разделы программы «Feeding Conveyor 1» («Устройство подачи конвейера 1»), «Feeding Conveyor 2» («Устройство подачи конвейера 2»), «Process» («Процесс») и «Discharging Conveyor» («Разгрузка конвейера») являются ориентированными на процесс.

Пример также показывает использование различных типов блоков. Главная программа расположена в ОВ 1; именно в этой программе вызываются блоки рабочего режима, различных частей оборудования предприятия и обработки данных. Эти блоки являются функциональными блоками с экземплярным блоком данных для хранения данных. «Feeding Conveyor 1» и «Feeding Conveyor 2» структурированы одинаково; FB 20 с DB 20 в качестве экземпляра блока данных для «Feeding Conveyor 1» и

с DB 21 в качестве экземпляра блока данных для «Feeding Conveyor 2» используется для управления. В программе управления конвейером функция FC 20 обрабатывает (взаимные) блокировки (средства синхронизации выполнения различных секций программы); он опрашивает входы или меркеры и управляет локальными данными блока FB 20.

Функциональный блок FB 101 содержит программу управления транспортером конвейера и вызывается один раз для каждого транспортера. Вызов является локальным экземпляром, поэтому его локальные данные находятся в экземплярном блоке данных DB 20. Это относится и к программе сбора данных в FB 29. Программа обработки данных в FB 50, который использует DB 50, обрабатывает данные, приобретенные блоком FB 29 (и другими блоками) и расположенные в глобальном блоке данных DB 60. Функция FC 51 подготавливает эти данные для передачи. Передача управляется блоком FB 51 (с DB 51), в котором вызываются системные блоки SFB 8, SFB 9 и SFB 62. Здесь также SFB хранят свои экземплярные данные в блоке данных «верхнего уровня» DB 51.

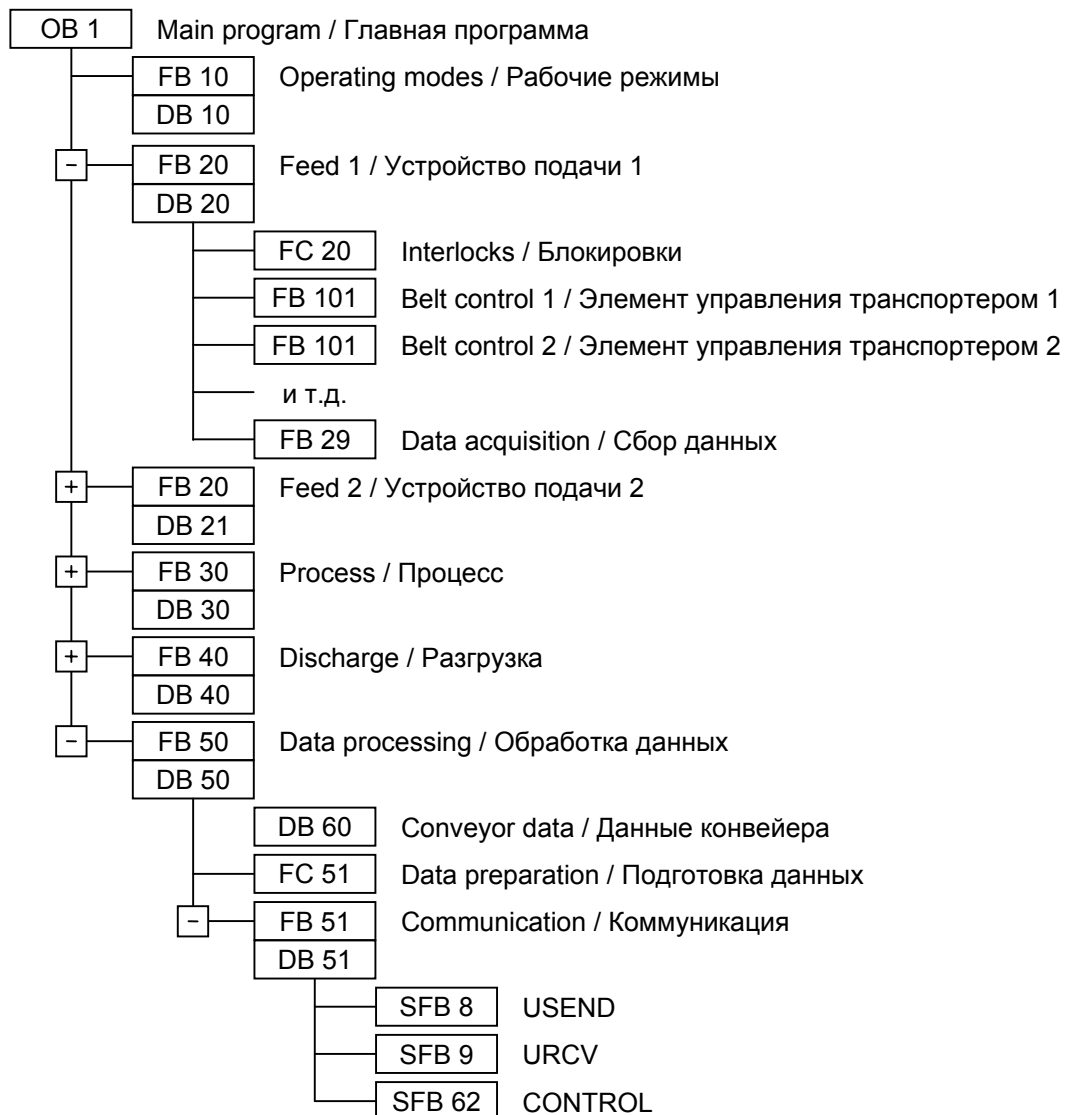


Рисунок 20.1 Пример структурирования программы

20.2 Управление циклом сканирования

20.2.1 Обновление образа процесса

Образ процесса является частью внутренней системной памяти CPU (параграф 1.1.4 «Области памяти CPU»). Он начинается с нулевого адреса I/O (входа/выхода) и завершается верхней границей, определяемой CPU. В соответствующем образе оборудованном CPU вы можете сами установить это предельное значение.

Обычно все цифровые модули располагаются в области адресов образа процесса, в то время как все аналоговые модули имеют адреса вне этой области. Если CPU имеет свободное адресное пространство, то вы можете, используя конфигурационную таблицу, адресовать любой модуль поверх образа процесса или вне области образа процесса.

Образ процесса состоит из входной таблицы образа процесса (входы I) и выходной таблицы образа процесса (выходы Q), соответственно process-image input table и process-image output table.

После рестарта CPU и перед первым выполнением ОБ 1 операционная система пересылает сигнальные состояния выходной таблицы образа процесса в выходные модули и принимает сигнальные состояния входных модулей во входную таблицу образа процесса. Затем следует выполнение ОБ 1, где входы обычно комбинируются между собой, и формируются выходы. После завершения ОБ 1 начинается новый цикл с обновления образа процесса (рисунок 20.2).

Если при автоматическом обновлении образа процесса возникает ошибка, например, если модуль больше недоступен, то вызывается организационный блок ОБ 85 «Program Execution Error» («Ошибки выполнения программы»). Если ОБ 85 недоступен, CPU переходит в режим STOP.

Образы подпроцессов

Используя соответствующим образом оборудованные CPU, вы можете поделить образ процесса на образы подпроцессов, число которых может быть до 9 или до 16. Это разбиение вы должны выполнять при параметризации сигнальных модулей путем определения образа подпроцесса, через который модуль должен быть адресован, при назначении адреса. Вы можете выполнить разбиение отдельно в соответствии с входной таблицей образа процесса и выходной таблицей образа процесса.

Модули, которые вы не назначили одному из образов подпроцесса от 1 до 8 или 15, хранятся в образе подпроцесса 0. Образ подпроцесса 0 обновляется автоматически операционной системой CPU как часть циклического выполнения.

Имея соответствующим образом оборудованные CPU, вы можете также назначить образы подпроцесса организационным блокам прерываний с целью их автоматического обновления при вызове этих ОБ.

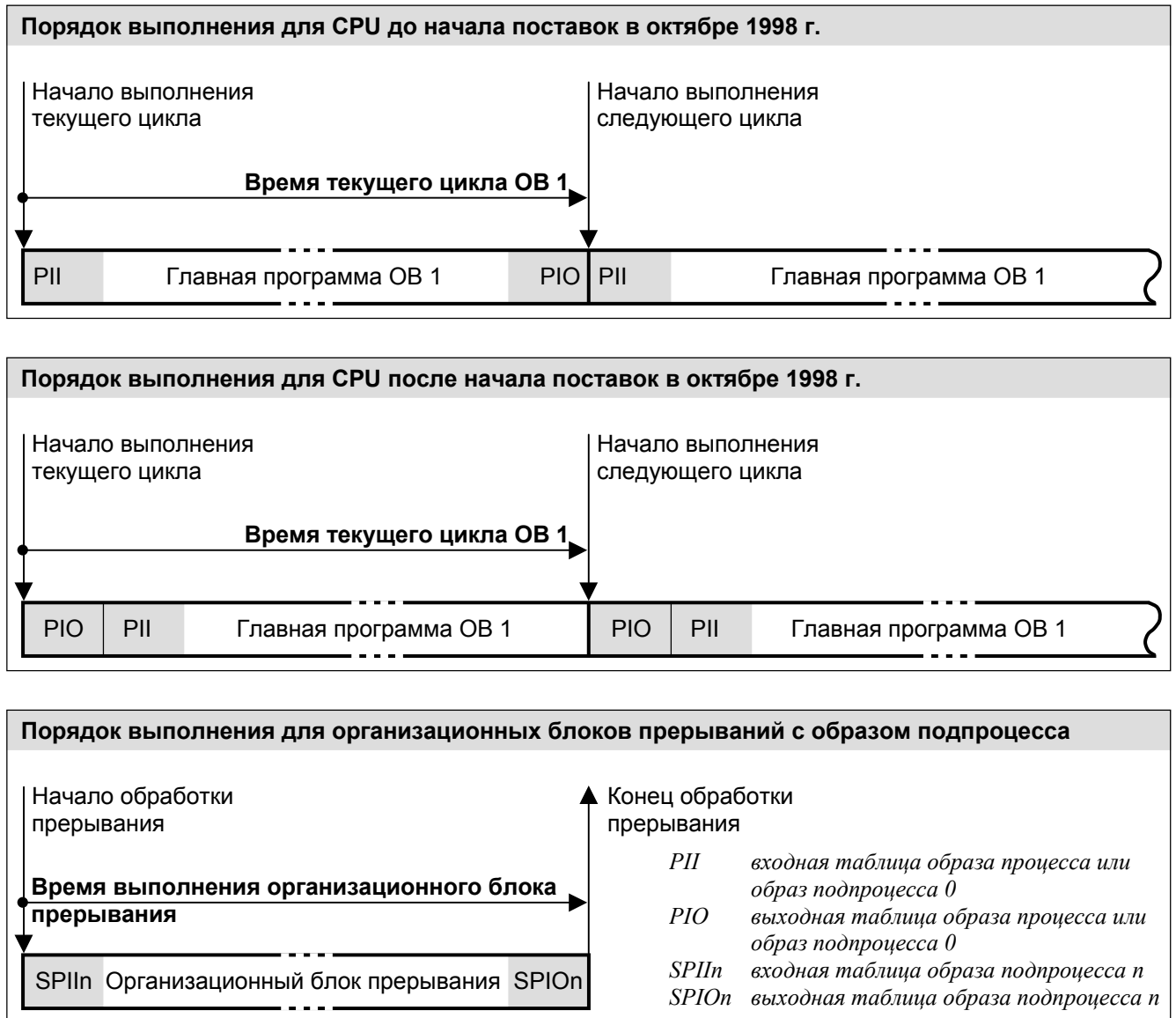


Рисунок 20.2 Обновление образа процесса

SFC 26 UPDAT_PI SFC 27 UPDAT_PO

Обновление образов подпроцесса осуществляется путем вызова в пользовательской программе системной функции. Для входной таблицы образа подпроцесса применяется SFC 26 UPDAT_PI, для выходной таблицы образа подпроцесса используется SFC 27 UPDAT_PO.

В таблице 20.1 показаны параметры этих SFC. С помощью этих SFC вы также можете обновить образ подпроцесса 0.

Вы можете выполнять обновление отдельных образов подпроцесса, вызывая эти SFC в любое время и в любом месте программы. Например, вы можете указать образ подпроцесса для приоритетного класса (уровень выполнения программы), и затем вызвать этот образ подпроцесса, обновление которого требуется провести, в начале и

в конце соответствующего организационного блока, когда обрабатывается данный приоритетный класс.

Обновление образа процесса может быть прервано вызовом более высокого приоритетного класса. Если во время обновления процесса возникнет ошибка, например, если модуль больше недоступен, то сообщение об этой ошибке поступи через значения функции SFC.

Таблица 20.1 Параметры SFC для обновления образа процесса

Имя параметра	SFC		Объявление	Тип данных	Комментарии, описание
PART	26	27	INPUT	BYTE	Номер образа подпроцесса (0 ... 15)
RET_VAL	26	27	OUTPUT	INT	Информация об ошибке
FLADDR	26	27	OUTPUT	WORD	При ошибке доступа: адрес первого байта, вызвавшего ошибку

20.2.2 Время наблюдения цикла сканирования

Сканирование программы в организационном блоке OB 1 наблюдается так называемым «монитором цикла сканирования» («scan cycle monitor») или «наблюдателем цикла сканирования» («scan cycle watchdog»). Значение по умолчанию для времени наблюдения цикла сканирования (scan cycle monitoring time) составляет 150 мс. Вы можете изменить его значением от 1 мс до 6 с путем установки соответствующего параметра CPU.

Если главная программа затрачивает на сканирование времени больше, чем заданное время наблюдения цикла сканирования, то CPU вызывает OB 80 («Timeout», «Таймаут»). Если OB 80 не был запрограммирован, то CPU переходит в состояние STOP.

Время наблюдения цикла сканирования включает полное время сканирования для OB 1. В него также входят временные интервалы сканирования для более высоких приоритетных классов, прерывающих выполнение главной программы (в текущем цикле). Коммуникационные процессы, выполняемые операционной системой, такие как GD-коммуникация или доступ PG к CPU (состояние блока!) также увеличивает время выполнения главной программы. Увеличение времени можно частично уменьшить, задав соответствующие параметры CPU («Cyclic load from communication», «Циклическая загрузка из устройств коммуникации» на вкладке «Cycle/Clock memory bits», «Цикл/Тактовый меркер»).

Статистика цикла

Если вы задействовали онлайнное соединение устройства программирования и работающего CPU, выберите пункт меню PLC → Module Information (PLC → Информация модуля) для вызова диалогового окна, содержащего несколько вкладок. На вкладке «Cycle Time» («Время цикла») будет отображено время текущего цикла, а также самое короткое и самое продолжительное время цикла. Также будут представ-

лены параметрическая минимальная длительность цикла и время наблюдения цикла сканирования.

Время последнего цикла, минимальное и максимальное время цикла с момента последнего запуска PLC также могут быть прочитаны во временных локальных данных в стартовой информации блока OB 1.

SFC 43 RE_TRIGR

Повторный запуск времени наблюдения цикла сканирования

Системная функция SFC 43 RE_TRIGR возобновляет время наблюдения цикла сканирования; таймер перезапускается с новым значением, установленным посредством параметризации CPU. SFC 43 не имеет параметров.

Время работы операционной системы

Время цикла сканирования также включает время работы операционной системы. Оно складывается из следующих элементов:

- Системное управление циклическим сканированием («no-load cycle», «цикл без загрузки»), фиксированное значение;
- Обновление образа процесса; зависит от количества обновляемых байтов;
- Обновление таймеров; зависит от количества обновляемых таймеров;
- Коммуникационная загрузка.

Коммуникационные функции CPU включают в себя передачу программных блоков пользователя или обмен данными между модулями CPU при помощи системных функций. Время, которое CPU будет использовать для этих функций, может быть ограничено при параметризации CPU.

Все значения времени исполнения (работы) операционной системы определяются версией CPU.

20.2.3 Минимальное время цикла сканирования, фоновое сканирование

Имея CPU с соответствующими возможностями, вы можете определять минимальное время цикла сканирования. Если главная программа (включая прерывания) затрачивает меньше времени, CPU ожидает до окончания установленного минимального времени сканирования цикла, прежде чем начать следующий цикл путем повторного вызова OB 1.

Значение по умолчанию для минимального времени сканирования цикла равно 0 мс, то есть функция деактивирована. Вы можете установить минимальное время скани-

рования цикла в пределах от 1 мс до 6 с во вкладке «Cycle/Clock memory bits», («Цикл/Тактовый меркер») при параметризации CPU.

Фоновое сканирование ОВ 90

В интервале между фактическим окончанием цикла и завершением минимального времени цикла CPU выполняет организационный блок ОВ 90 «Background scanning» («Фоновое сканирование»). Это действие проиллюстрировано на рисунке 20.3. ОВ 90 выполняется «кусками». Когда операционная система вызывает ОВ 1, выполнение ОВ 90 прерывается. Оно продолжается после завершения ОВ 1 в точке прерывания.

ОВ 90 может быть прерван после каждого оператора, однако, любой системный блок, вызываемый в ОВ 90, сначала сканируется полностью.

Размер «куска» зависит от времени текущего цикла сканирования ОВ 1. Чем ближе время сканирования ОВ 1 к минимальному времени цикла сканирования, тем меньше времени остается для выполнения ОВ 90. Время сканирования программы в ОВ 90 не наблюдается.

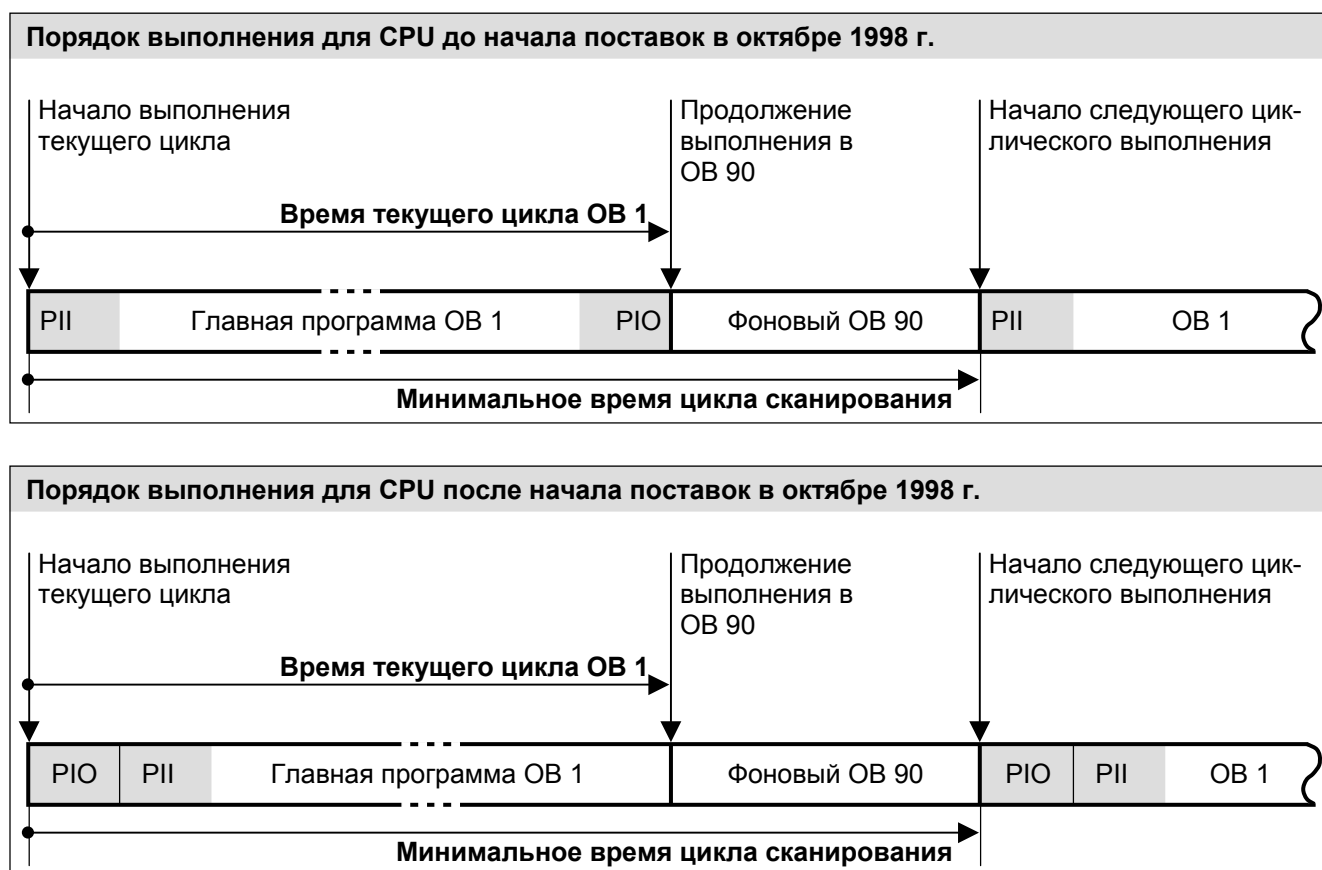


Рисунок 20.3 Минимальная длительность цикла и фоновое сканирование

ОВ 90 сканируется, только когда CPU находится в режиме RUN. Он может быть приостановлен событиями прерываний или ошибок, как и ОВ 1. Стартовая информация во временных локальных данных (байт 1) также сообщает, какие события приводят к выполнению ОВ 90 с начала (или к его запуску):

- В#16#91
после рестарта CPU;
- В#16#92
после удаления или замены блока, обрабатываемого в ОВ 90;
- В#16#93
после (повторной) загрузки ОВ 90 в режиме RUN;
- В#16#95
после окончания сканирования программы в ОВ 90 и начала нового фонового цикла.

20.2.4 Время отклика

Если программа пользователя в ОВ 1 работает с сигнальными состояниями образов процесса, это сказывается на времени отклика (response time), которое зависит от времени выполнения программы (времени цикла сканирования). Время отклика располагается между одним и другим циклами сканирования, как объясняется в следующем примере.

Пусть активированный концевой выключатель (limit switch) меняет свое сигнальное состояние с «0» на «1». Программируемый контроллер обнаруживает это изменение во время последующего обновления образа процесса, и устанавливает в «1» входы, отнесенные к концевому выключателю. Программа определяет данное изменение путем сброса выхода, например, чтобы выключить соответствующий мотор. Новое сигнальное состояние выхода, который был сброшен, передается в конце сканирования программы; только тогда сбрасывается соответствующий бит цифрового выходного модуля.

В лучшем случае образ процесса обновляется немедленно после изменения сигнала концевой выключателя. Тогда для отклика соответствующего выхода понадобится только один цикл (рисунок 20.4). В худшей ситуации обновление образа процесса было только что завершено, когда изменился сигнал концевой выключателя. Теперь необходимо ждать приблизительно один цикл, пока программируемый контроллер обнаружит изменение сигнала и установит выход. После еще одного цикла программа сможет ответить.

При таком рассмотрении время исполнения программы пользователя содержит все процедуры в одном программном цикле (включая, к примеру, обслуживание прерываний, выполняемые операционной системой функции, такие как обновление таймеров, управление MPI-интерфейсом и обновление образов процесса).

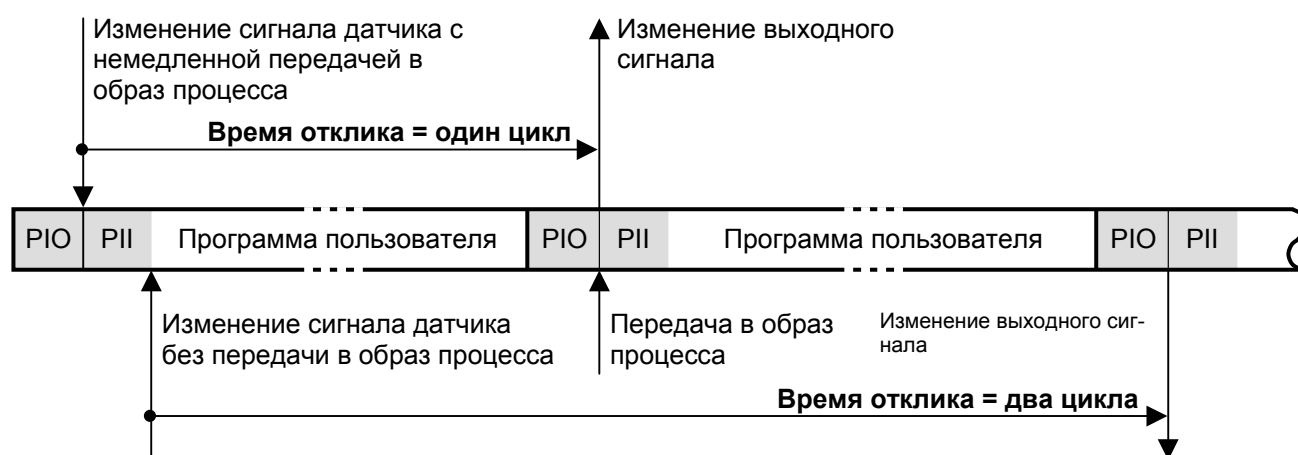


Рисунок 20.4 Время откликов программируемых контроллеров

Время отклика для изменения входного сигнала, таким образом, может быть между первым и вторым циклами. К времени отклика прибавляются задержки входных модулей, время переключения контакторов и так далее.

В некоторых случаях вы можете уменьшить время отклика путем прямой адресации входов/выходов, или вызывая программные разделы по возникновению событий.

20.2.5 Стартовая информация

Операционная система CPU передает стартовую информацию в первые 20 байт временных локальных данных организационного блока ОВ 1. То же самое происходит при вызове любого организационного блока. Вы можете сами создать описание стартовой информации или прибегнуть к использованию информации из стандартной библиотеки *Standard Library*, раздел *Organization Blocks* (*Организационные блоки*).

В таблице 20.2 показана эта стартовая информация для ОВ 1, а также символическое обозначение по умолчанию и типы данных. Вы можете поменять обозначение в любое время и выбрать более подходящие для вас имена. Даже если вы не используете стартовую информацию, вы должны зарезервировать под нее первые 20 байтов временных локальных данных (например, в виде 20-байтового массива).

В SIMATIC S7 все сообщения о событиях имеют фиксированную структуру, определяемую классом события. Стартовая информация для ОВ 1, к примеру, сообщает о событии V#16#11, как о вызове стандартного ОВ. Из содержимого следующего байта вы можете узнать, выполняется ли первый цикл главной программы после включения питания, и вследствие чего она вызывает, например, процедуры инициализации в циклической программе.

Приоритет и номер ОВ главной программы постоянны. Используя три значения типа INT, стартовая информация предоставляет данные о времени последнего цикла сканирования, минимальном и максимальном значениях времени цикла с момента последнего включения питания. Последнее значение в формате DATE_AND_TIME со-

общает о том, когда программа управления приоритетами получила событие вызова OB 1.

Обратите внимание на то, что прямое чтение стартовой информации организационного блока возможно только в этом организационном блоке, потому что эта информация состоит из временных локальных данных. Если вам требуется стартовая информация в боках, которые расположены на более глубоких уровнях, вызовите системную функцию SFC RD_SINFO в соответствующей точке программы.

Таблица 20.2 Стартовая информация для OB 1

Имя	Тип данных	Описание	Содержимое
OB1_EV_CLASS	BYTE	Класс события	V#16#11 = Вызов стандартного OB
OB1_SCAN_1	BYTE	Стартовая информация	V#16#01 = Первый цикл после полного рестарта V#16#02 = Первый цикл после «теплого» рестарта V#16#03 = Любой другой цикл
OB1_PRIORITY	BYTE	Приоритет	V#16#01
OB1_OB_NUMBER	BYTE	Номер OB	V#16#01
OB1_RESERVED_1	BYTE	Резерв	-
OB1_RESERVED_2	BYTE	Резерв	-
OB1_PREV_CYCLE	INT	Время предыдущего цикла сканирования	Значение в мс
OB1_MIN_CYCLE	INT	Минимальное время цикла сканирования	Значение в мс
OB1_MAX_CYCLE	INT	Максимальное время цикла сканирования	Значение в мс
OB1_DATE_TIME	INT	Возникшее событие	Время вызова OB (циклическое)

SFC 6 RD_SINFO

Считывание стартовой информации

Стартовая информация о текущем организационном блоке (то есть блоке OB на вершине дерева вызовов) и о запуске последнего выполненного OB, даже на более глубоком уровне вызова, предоставляется вам системной функцией SFC 6 RD_SINFO (таблица 20.3).

Выходной параметр TOP_SI содержит первые 12 байтов стартовой информации о текущем OB, в выходном параметре START_UP_SI содержатся первые 12 байтов стартовой информации по последнему запуску выполненного OB. Ни в каком из этих параметров нет временной метки (time stamp).

SFC 6 RD_SINFO может быть вызвана не только из любой точки программы, но и из любого приоритетного класса, даже из организационного блока обработки ошибки или процедуры запуска. Если SFC вызвана в организационном блоке обслуживания прерывания, то, к примеру, TOP_SI содержит стартовую информацию OB прерыва-

ния. В случае вызова при рестарте TOP_SI и START_UP_SI имеют одинаковые значения.

Таблица 20.3 Параметры SFC 6 RD_SINFO

SFC	Наименование параметра	Объявление	Тип данных	Содержимое, описание
6	RET_VAL	OUTPUT		Информация об ошибке
	TOP_SI	OUTPUT		Стартовая информация для текущего ОБ (структура та же, что и у START_UP_SI)
	START_UP_SI .EV_CLASS .EV_NUM .PRIORITY .NUM .TYP2_3 .TYP1 .ZI1 .ZI2_3	OUTPUT	STRUCT BYTE BYTE BYTE BYTE BYTE WORD DWORD	Старт. информация для последнего запущенного ОБ ID (идентификатор) события и класс события Номер события Приоритет исполнения (номер уровень исполнения) Номер ОБ ID дополнительной информации 2_3 ID дополнительной информации 1 Дополнительная информация 1 Дополнительная информация 2_3

20.3 Программные функции

Наряду с параметризацией CPU при помощи утилиты Hardware Configuration вы также можете назначать и опрашивать некоторые параметры PLC динамически во время выполнения программы, используя встроенные системные функции.

20.3.1 Таймер реального времени

Для управления таймером реального времени (real-time clock) CPU можно использовать следующие системные функции:

- SFC 0 SET_CLK
Установка даты и времени
- SFC 1 READ_CLK
Чтение даты и времени
- SFC 48 SNC_RTCS
Синхронизация таймеров CPU

В таблице 20.4 приведен список параметров системных функций.

Когда несколько CPU соединены между собой в подсети, инициализируйте таймеры (внутренние часы) одного из CPU как «главный таймер». При параметризации CPU также введите интервал синхронизации, по истечении которого все таймеры в подсети должны быть автоматически синхронизированы по главному таймеру.

Вызов SFC 48 SNC_RTCS в CPU с главным таймером синхронизирует все таймеры в подсети независимо от автоматической синхронизации. При установке главного таймера с использованием SFC 0 SET_CLK все остальные таймеры в подсети автоматически синхронизируются по этому значению.

Таблица 20.4 Параметры SFC для таймера реального времени

SFC	Имя параметра	Объявление	Тип данных	Содержимое, описание
0	PDT	INPUT	DT	Дата и время (новые)
	RET_VAL	OUTPUT	INT	Информация об ошибке
1	RET_VAL	OUTPUT	INT	Информация об ошибке
	CDT	OUTPUT	DT	Дата и время (текущие)
48	RET_VAL	OUTPUT	INT	Информация об ошибке

20.3.2 Чтение системного таймера

Системный таймер (system clock) CPU запускается при подаче питания или после полного перезапуска. Системный таймер продолжает работать, пока CPU выполняет подпрограмму рестарта или находится в режиме RUN. Когда CPU переходит в состояние STOP или HOLD, текущее системное время «замораживается».

Если вы инициируете «теплый» рестарт на CPU S7-400, то системный таймер запускается вновь, используя сохраненное значение в качестве начала отсчета времени. «Холодный» рестарт или полный перезапуск («теплый» рестарт) сбрасывает системное время.

Системное время имеет формат данных TIME, следовательно, оно может принимать только положительные значения:

от TIME#0ms до TIME#24d20h31m23s647ms.

В случае переполнения часы запускаются вновь с 0. CPU 3xx (кроме CPU 318) обновляют системный таймер каждые 10 миллисекунд, CPU 318 и CPU 4xx – каждую миллисекунду.

SFC 64 TIME_TCK

Считывание системного времени

Вы можете прочитать текущее системное время с помощью системной функции SFC 64 TIME_TCK. Параметр RET_VAL содержит системное время в формате данных TIME.

Системный таймер может использоваться, например, для считывания текущего времени прогона CPU или, через вычисление разности, для подсчета времени между двумя вызовами SFC 64. Разность между двумя значениями в формате TIME вычисляется с использованием вычитания DINT.

20.3.3 Счетчик учета рабочего времени

Счетчик учета рабочего времени (run-time meter) в CPU считает часы. Вы можете использовать счетчик рабочего времени выполнения для таких задач, как определение времени работы CPU или измерение времени работы устройств, подключенных к этому CPU.

Количество счетчиков рабочего времени для CPU определяется его версией. Когда находится в состоянии STOP или HOLD, счетчик рабочего времени также останавливается; когда CPU перезапускается, счетчик рабочего времени продолжает отсчет времени с предыдущего значения.

По достижении счетчиком рабочего времени 32767 часов он останавливается и сообщает о переполнении. Счетчик рабочего времени может быть установлен в новое значение или сброшен в ноль только посредством вызова SFC.

Для управления счетчиком рабочего времени доступны следующие системные функции:

- SFC 2 SET_RTM
Установка счетчика рабочего времени
- SFC 3 CTRL_RTM
Запуск и останов счетчика рабочего времени
- SFC 4 READ_RTM
Опрос счетчика рабочего времени

В таблице 20.5 представлены параметры этих системных функций.

Параметр NR содержит номер счетчика рабочего времени и имеет тип BYTE. Он может быть инициализирован с использованием константы или переменной (как для любого параметра простого типа данных). Параметр PV (тип данных INT) применяется для установки счетчика рабочего времени в начальное значение. Параметр S SFC 3 запускает (по сигнальному состоянию «1») или останавливает (сигнальным состоянием «0») выбранный счетчик рабочего времени. CQ сообщает, счетчик рабочего времени работает (сигнальное состояние «1») при опросе или остановлен (сигнальное состояние «0»). Параметр CV записывает часы в формате INT.

Таблица 20.5 Параметры SFC для счетчика рабочего времени

SFC	Параметр	Объявление	Тип данных	Содержимое, описание
2	NR	INPUT	BYTE	Номер счетчика рабочего времени (от V#16#01 до V#16#08)
	PV	INPUT	INT	Новое значение для счетчика рабочего времени
	RET_VAL	OUTPUT	INT	Информация об ошибке
3	NR	INPUT	BYTE	Номер счетчика рабочего времени (от V#16#01 до V#16#08)
	S	INPUT	BOOL	Запуск (при «1») или останов (при «0») счетчика рабочего времени
	RET_VAL	OUTPUT	INT	Информация об ошибке
4	NR	INPUT	BYTE	Номер счетчика рабочего времени (от V#16#01 до V#16#08)
	RET_VAL	OUTPUT	INT	Информация об ошибке
	CQ	OUTPUT	BOOL	Счетчик рабочего времени работает («1») или остановлен («0»)
	CV	OUTPUT	INT	Текущее значение счетчика рабочего времени

20.3.4 Сжатие памяти CPU

Множественные удаления и перезагрузки блоков, часто имеющие место во время онлайн-изменения блоков, может вызвать появление пустых областей в рабочей памяти CPU и в загрузочной памяти RAM, что уменьшает количество полезного пространства памяти. При вызове функции «Compress» («Сжатие») вы запускаете программу CPU, которая заполняет упомянутые промежутки, сдвигая блоки вместе. Можно инициировать функцию «Compress» («Сжатие») через программирующее устройство, подключенное к CPU, или путем вызова системной функции **SFC 25 COMPRESS**. Параметры SFC 25 указаны в таблице 20.6.

Процедура сжатия распределена между несколькими программными циклами. SFC возвращает BUSY = «1», чтобы сообщить, что функция все еще находится в работе. DONE = «1» сообщает о том, что функция завершила операцию сжатия. SFC не может сжимать, когда осуществляется внешне инициированное сжатие, активна функция «Delete Block» («Удаление блока») или функции PG обращаются к блоку, предназначенному для сдвига (к примеру, функция определения состояния блока «Block Status»).

Обратите внимание, что блоки определенного, зависящего от версии CPU, максимального размера не могут быть сжаты, поэтому свободные промежутки в памяти CPU могут остаться. Все пробелы закрывает только функция «Compress» («Сжатие»), вызванная устройством PG, пока CPU находится в состоянии STOP.

Таблица 20.6 Параметры SFC 25 COMPRESS

SFC	Параметр	Объявление	Тип данных	Содержимое, описание
25	RET_VAL	OUTPUT	INT	Информация об ошибке
	BUSY	OUTPUT	BOOL	Сжатие не окончено (значение «1»)
	DONE	OUTPUT	BOOL	Сжатие завершено (значение «1»)

20.3.5 Ожидание и останов

Системная функция **SFC 47 WAIT** останавливает программное сканирование на определенный период времени.

SFC 47 WAIT имеет параметр WT типа данных INT, в котором вы можете задать время ожидания в микросекундах (μ s). Максимальное время ожидания составляет 32767 микросекунд; минимальное время ожидания соответствует времени исполнения системной функции, определяемому версией CPU. SFC 47 может быть прервана событиями с более высоким приоритетом. В случае S7-300 это увеличивает время ожидания на интервал сканирования процедуры обработки прерывания более высокого приоритета.

Системная функция **SFC 46 STP** завершает программное сканирование, и CPU переходит в состояние STOP. SFC 46 STP не имеет параметров.

20.3.6 Режим мультипроцессорной обработки

S7-400 разрешает мультипроцессорную обработку (одновременное выполнение задач несколькими процессорами). Может быть осуществлено управление четырьмя соответствующим образом разработанными CPU в одной стойке, на одной P-шине или K-шине.

Станция S7-400 автоматически перейдет в мультипроцессорный режим, если вы в утилите Hardware Configuration разместите более одного CPU в центральной стойке. Можно занимать произвольные слоты; CPU различаются по номерам, автоматически назначаемым в возрастающем порядке при монтаже CPU. Вы можете сами назначить эти номера на вкладке «Multicomputing» («Многопроцессорное вычисление»).

Конфигурационные данные для всех CPU должны быть загружены в PLC, даже если вы вносите изменения только для одного CPU. После назначения параметров для центральных процессоров вы должны каждый модуль в станции назначить процессору.

Это осуществляется путем параметризации модуля на вкладке «Addresses» («Адреса») в разделе «CPU Assignment» («Назначение CPU») (рисунок 20.5). Одновременно, назначая области адреса модуля, вы также можете назначить прерывания модуля для данного CPU. С помощью команды View → Filter → CPU No. x-modules (Вид → Фильтр → Номер CPU x-модули) вы можете выделить модули, назначенные CPU, в конфигурационных таблицах.

В мультипроцессорной сети все CPU находятся в одном и том же рабочем режиме. Это означает, что

- Все они должны быть параметризованы с одинаковым режимом рестарта;
- Все они переходят в режим RUN одновременно;
- Все они переходят в режим HOLD, когда вы производите отладку в пошаговом режиме в одном из CPU;
- Все они переходят в режим STOP, как только один из CPU перешел в этот режим.

Когда происходит сбой в одной из стоек станции, в каждом CPU вызывается организационный блок OB 86. Пользователь программирует в этих CPU независимое друг от друга выполнение; они не синхронизированы. SFC 35 MP_ALM вызывает одновременно во всех CPU организационный блок OB 60 «Multiprocessor interrupt» («Мультипроцессорное прерывание») (обратитесь к параграфу 21.6 «Мультипроцессорное прерывание»).

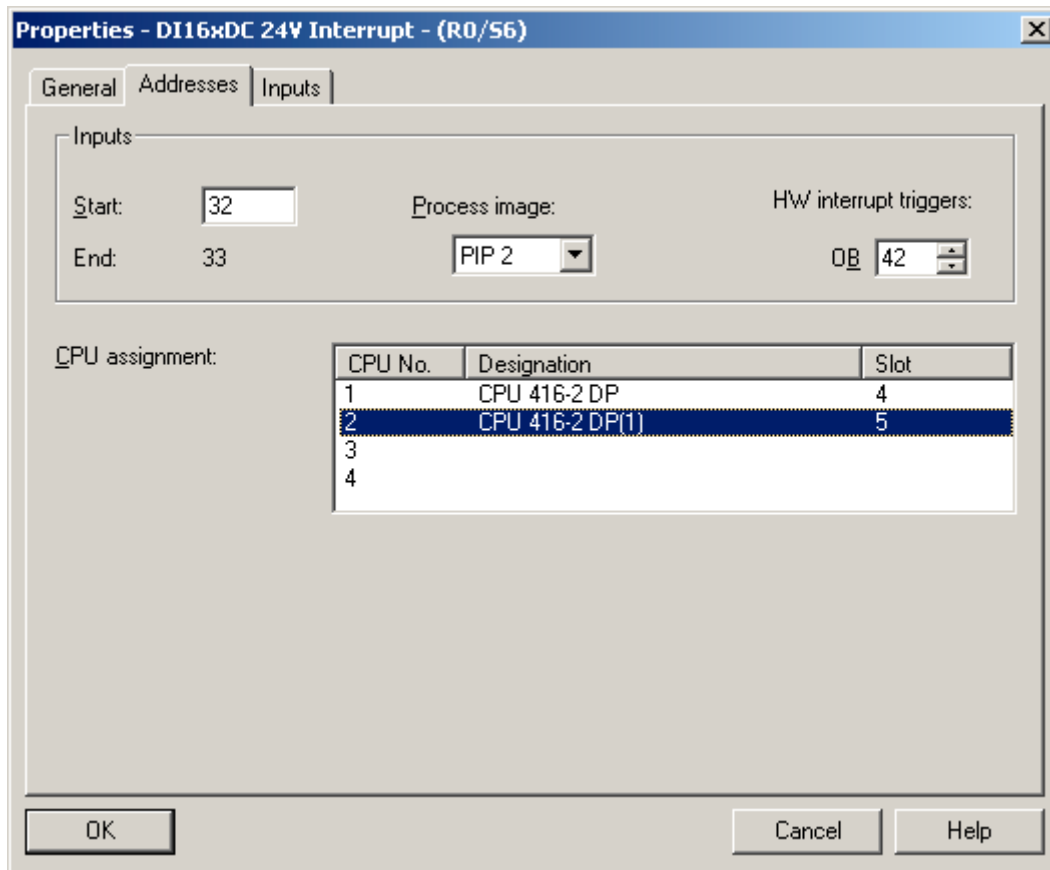


Рисунок 20.5 Назначение модуля в мультипроцессорном режиме

20.4 Коммуникация через распределенные входы/выходы

Подобно тому, как централизованно скомпонованные модули назначены центральному процессору и управляются из этого CPU, распределенные модули (станции, DP-ведомые) назначены DP-мастеру. DP-мастер (DP master) и все его DP-ведомые (DP slaves) образуют систему DP-мастера. Одна S7-станция может содержать несколько систем DP-мастера.

Как и централизованно организованные модули, DP-ведомые занимают адреса в области входов/выходов (I/O-области) CPU (область логических адресов). DP-мастер является «прозрачным», так сказать, для адресов DP-ведомых; CPU «видит» адреса DP-ведомых, таким образом, адреса DP-ведомых и адреса централизованно скомпонованных модулей не должны перекрываться. Адреса DP-ведомых также не должны перекрываться с адресами DP-ведомых в других системах DP-мастера, назначенных CPU.

Имеются DP-ведомые, которые действуют как централизованно скомпонованные модули: у них есть области пользовательских и системных данных, они могут инициировать прерывания процесса и диагностические прерывания при соответствующем оснащении. Эти станции называются «ведомыми DP S7» («DP S7 slaves»). «Стандартные ведомые DP V0» («DP V0 standard slaves») соответствуют EN 50170, том 2, PROFIBUS. Между ними есть существенное различие, заключающееся в их способе чтения и в структуре диагностических данных.

Конфигурирование распределенных входов/выходов (distributed I/O) также очень похоже с настройкой централизованных модулей. Начальной точкой является DP-мастер с представленной графически системой DP-мастера. Станции «вешаются» на него и затем адресуются и параметризуются.

Необходимость в особой последовательности пользовательских данных (консистентные данные) требует специальные системные функции для распределенных входов/выходов; так, несмотря на последовательную передачу на DP-ведомые, консистентные данные могут выходить за 4 байта, обеспечиваемые S7-системой, и вы можете соединять группы DP-ведомых таким образом, что они могут передавать или выводить данные синхронно.

20.4.1 Адресация распределенных входов/выходов

Каждый DP-ведомый дополнительно к адресу узла имеет три адреса: географический адрес, стартовый адрес модуля и диагностический адрес (рисунок 20.6).

Адрес узла

Каждый узел (node) в подсети PROFIBUS имеет уникальный адрес, адрес узла (номер станции) в этой подсети, позволяющий отличать его от других узлов в подсети. Обращение к станции в PROFIBUS происходит с использованием этого адреса узла.

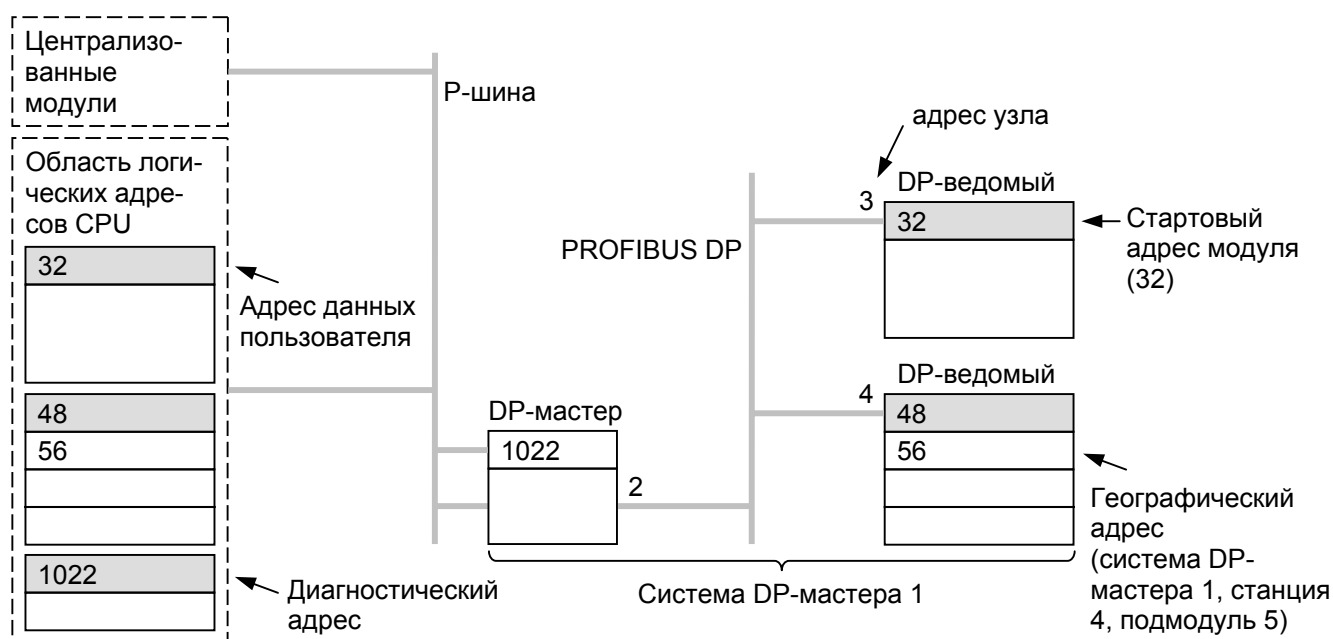


Рисунок 20.6 Адреса в системе DP-мастера

Обратите внимание на то, что между адресами активных узлов шины должен быть хотя бы 1 промежуток (например, в случае DP-мастера и узлов в перекрестном трафике). В STEP 7 этому уделяется внимание при автоматическом назначении адресов узлов.

Географический адрес

Географический адрес DP-ведомого соответствует адресу слота централизованного модуля. Он состоит из ID (идентификатора) системы DP-мастера (определяемой при конфигурировании) и адреса узла на PROFIBUS (соответствует номеру стойки).

В случае модульного построения DP-ведомых к этому добавляется номер слота, а в случае модулей с подмодулями добавляется номер слота подмодуля (в S7-300 нумерация начинается с 4).

Стартовый адрес модуля, консистентные данные

По стартовому адресу модуля («логический базовый адрес») вы можете получить доступ к пользовательским данным компактного DP-ведомого или модуля в модульном DP-ведомом. Этот адрес соответствует стартовому адресу централизованного модуля. Если адрес DP-ведомого имеет консистентность данных из 1, 2 или 4 байт, то вы можете обратиться к пользовательским данным с помощью операторов загрузки и передачи или при помощи блочного элемента MOVE. Если стартовый адрес модуля находится в образе процесса, то пользовательские данные передаются в ходе передачи образа процесса. Также возможно назначение образу подпроцесса.

Если консистентность данных 3 байта или более 4 байт (до определенного центральным процессором количества), то вам для консистентного чтения и записи пользовательских данных потребуются системные функции SFC 14 DPRD_DAT и SFC 15 DPWR_DAT. SFC обходят образ процесса, чтобы прочитать или записать области данных, созданные в параметре RECORD, если образ процесса не является источником данных или назначением для данных в параметре RECORD.

Рисунок 20.7 иллюстрирует передачу пользовательских данных. DP-мастер передает пользовательские данные «своих» DP-ведомых циклически; в примере он пересылает из станции со стартовым адресом модуля 32 и консистентностью данных 4 байта (DP-ведомый 1) и из станции со стартовым адресом модуля 48 и консистентностью данных 8 байт (DP-ведомый 2).

Байты пользовательских данных DP-ведомого 1 расположены в области передачи DP-мастера на P-шине или в CPU и могут быть переданы, например, в блок данных в области памяти CPU с использованием операторов Load и Transfer (загрузка и передача) (блочный элемент MOVE), точно так же, как и централизованного модуля.

Байты пользовательских данных DP-ведомого 2 также находятся в области передачи DP-мастера, но только стартовый адрес модуля (48 в примере) доступен на P-шине.

Доступ к остальным адресам может быть осуществлен (теоретически) свободно, но они деактивированы утилитой Hardware Configuration для любого другого использования. SFC 14 и 15 адресуют пользовательские данные DP-ведомого 2 через этот стартовый адрес модуля, и они осуществляют обмен этими данными с областью памяти CPU, например, с использованием блока данных.

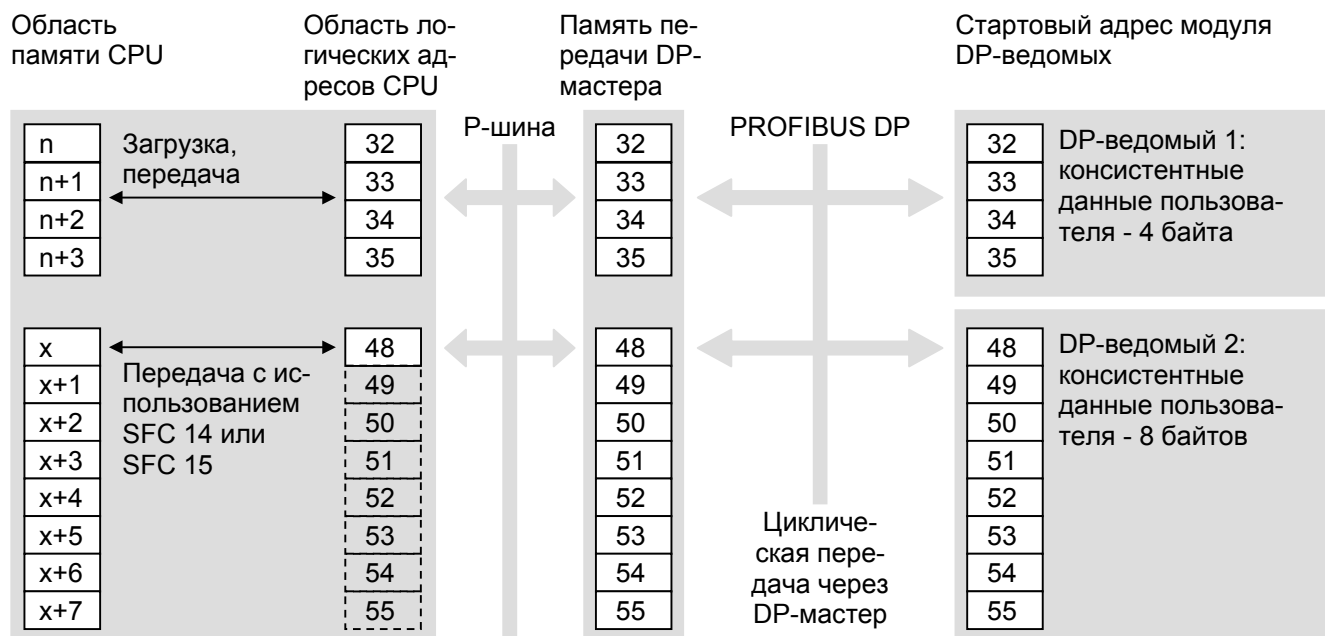


Рисунок 20.7 Передача пользовательских данных в распределенные входы/выходы

К этому адресу нельзя обратиться с помощью операторов Load и Transfer (загрузка и передача) или при помощи блочного элемента MOVE, ни даже через операционную систему при обновлении образа процесса; в примере образ процесса между адресами 48 и 55 не обновляется. Однако, вы можете задать для системных функций SFC 14/15 области в качестве источника или места назначения и, таким образом, по-прежнему использовать эти адреса.

Память передачи интеллектуальных DP-ведомых

В случае компактных и модульных DP-ведомых адреса входов и выходов расположены в области входов/выходов (I/O-области) централизованного CPU (называемого ниже главным CPU).

В случае интеллектуальных ведомых (с развитой логикой) главный CPU не имеет прямого доступа к модулям входов/выходов DP-ведомого. Следовательно, каждый DP-ведомый имеет память передачи (transfer memory), которая может быть разделена на несколько подобластей с различными размерами и консистентностью данных. С точки зрения главного CPU интеллектуальный DP-ведомый в этом случае выступает как компактный или модульный DP-ведомый в зависимости от подразделения.

Размер всей памяти передачи зависит от DP-ведомого. Вы можете определить адреса памяти передачи при конфигурировании: адреса с точки зрения ведомого CPU задаются, когда настраиваются интеллектуальные DP-ведомые, и адреса с точки зрения главного CPU определяются при вставке DP-ведомых в систему DP-мастера. Исключение: если CP 342-5DP формирует DP-интерфейс для интеллектуальных ведомых, деление его памяти передачи не конфигурируется до подключения ведомого к системе DP-мастера.

С точки зрения DP-мастера (точнее, с точки зрения централизованного главного CPU) адреса памяти передачи не должны перекрываться с адресами других модулей в (централизованной) S7-станции. С точки зрения ведомого CPU адреса памяти передачи не должны перекрываться с адресами модулей, скомпонованных в интеллектуальном DP-ведомом.

Области адресов памяти передачи функционируют так же, как отдельные модули с определяемыми пользователем доступом к данным и консистентностью данных. То есть наименьший адрес области адресов является «стартовым адресом модуля». В соответствии с установленным уровнем консистентности данных вы можете обратиться к пользовательским данным области адресов с помощью Load/Transfer/MOVE (операторов передачи, загрузки или соответствующего блочного элемента) или при помощи SFC 14/15 как в программе главного CPU, так и в программе ведомого CPU. В программе ведомого CPU вы можете также использовать SFC 7 DP_PRAL, чтобы вызвать прерывание процесса в главном CPU для области адресов.

Пример на рисунке 20.8 показывает память передачи с двумя адресными областями. Главный CPU видит область адресов 1, являющуюся модулем выхода, который может быть записан с помощью Transfer/MOVE (или также с использованием бинарных операций с памятью, если адреса расположены в образе процесса). Для ведомого CPU область адресов 1 – это входной модуль, который может быть прочитан при

помощи Load/MOVE или с использованием операций считывания, если адреса находятся в образе процесса. Область адресов 2 с консистентностью данных 8 байтов может быть записана ведомым CPU только с применением SFC 15 и прочитана главным CPU с использованием SFC 14.



Рисунок 20.8 Память передачи в интеллектуальных DP-ведомых

Диагностический адрес

Каждый DP-мастер и DP-ведомый занимает дополнительно один байт «диагностического адреса». Вы можете прочитать через диагностический адрес диагностические данные.

В стандартных ведомых DP V0 вы можете использовать системную функцию SFC 13 DPNRM_DG для чтения диагностических данных, в ведомых DP S7 можно использовать SFC 59 RD_REC для чтения записи данных DS 1, содержащей диагностические данные. Прочитанные с помощью SFC 13 DPNRM_DG диагностические данные имеют стандартную определенную структуру (рисунок 20.9). Запись данных DS 1, считываемая функцией SFC 59 RD_REC, содержит 4 байта диагностической информации модуля, которые также доступны, к примеру, в стартовой информации диагностического прерывания OB 82 (это соответствует записи диагностических данных DS 0).

Модули или устройства, не имеющие собственных пользовательских данных, также могут иметь диагностический адрес, если они могут обеспечить диагностические данные, как, например, DP-мастер или модуль электропитания с возможностью резервирования.

Общая структура диагностических данных стандартного ведомого DP V0

Байт	
0 ... 2	Состояние станции 1, 2 и 3
3	Номер главной станции
4 ... 5	ID изготовителя
6 ... n	Другие диагностические данные, определяемые ведомым

Общая структура диагностических данных ведомого DP S7

Байт	
0 ... 3	Запись данных DS 0 (стартовая информация в OB 82)
4 ... n	Другие диагностические данные, определяемые ведомым

Рисунок 20.9

Структурный принцип стандартных диагностических данных и записи диагностических данных DS 1

Диагностический адрес занимает один байт периферийных входов. STEP 7 назначает диагностический адрес, по умолчанию начиная с высшего адреса в области входов/выходов CPU. Обзор адресов в утилите Hardware Configuration диагностический адрес обозначается звездочкой.

20.4.2 Конфигурирование распределенных входов/выходов

Общая процедура

Вы можете конфигурировать распределенные входы/выходы, по существу, таким же образом, как и централизованные модули. Вместо компоновки модулей в монтажной стойке вы назначаете DP-станции (узлы PROFIBUS) системе DP-мастера. Рекомендуется следующий порядок необходимых действий:

- 1) Используя SIMATIC-менеджер, создайте новый проект или откройте существующий.
- 2) С помощью SIMATIC-менеджера в проекте создайте подсеть PROFIBUS и, если требуется, установите профиль шины.
- 3) Воспользуйтесь SIMATIC-менеджером, чтобы создать главную станцию (мастер-станцию) в проекте, то есть настроить DP-мастер, к примеру, станцию S7-400.

Если ваша система содержит интеллектуальные DP-ведомые, то создайте также соответствующие ведомые станции, например станции S7-300.

Запустите Hardware Configuration путем открытия мастер-станции.

- 4) С помощью Hardware Configuration поместите DP-мастер в главную станцию. Это может быть, например, CPU со встроенным DP-интерфейсом. Назначьте

предварительно созданную подсеть PROFIBUS DP-интерфейсу, после чего у вас будет система DP-мастера. Остальные модули вы можете сконфигурировать позднее. Сохраните и откомпилируйте станцию.

- 5) Если вы создали S7-станцию для интеллектуального DP-ведомого, откройте ее в Hardware Configuration и «подключите» («plug-in») модуль с требуемым DP-интерфейсом, например, CPU S7-300 со встроенным DP-интерфейсом или базовый модуль ET 200X BM 147/CPU. Если вы устанавливаете DP-интерфейс как «DP-ведомый», то назначьте предварительно созданную подсеть PROFIBUS DP-интерфейсу и сконфигурируйте интерфейс пользовательских данных с точки зрения DP-ведомого (память передачи). Остальные модули можно сконфигурировать позже. Сохраните и откомпилируйте станцию.

Проделайте то же самое для оставшихся станций, предназначенных для интеллектуальных DP-ведомых.

- 6) Откройте главную станцию с системой DP-мастера и с помощью мыши перетащите узлы PROFIBUS (компактные и модульные DP-ведомые) из каталога аппаратуры (Hardware Catalog) в систему DP-мастера. Назначьте адреса узлов и, если необходимо, установите стартовый адрес модуля и диагностический адрес.
- 7) Если вы создали интеллектуальные DP-ведомые, перетащите мышью соответствующую пиктограмму (в каталоге аппаратуры под «PROFIBUS DP» и «Already configured stations», «Сконфигурированные ранее станции») в систему DP-мастера.

Откройте пиктограмму и назначьте сконфигурированный ранее DP-ведомый («Connect», «Соединить»), назначьте адрес узла и сконфигурируйте интерфейс пользовательских данных с точки зрения DP-мастера (или с точки зрения центрального главного CPU). То же самое сделайте с каждым интеллектуальным DP-ведомым.

- 8) Сохраните и скомпилируйте все станции. Теперь система DP-мастера сконфигурирована. Вы можете дополнить конфигурацию централизованными модулями или другими DP-ведомыми.

Вы также имеете возможность сконфигурированную таким образом систему DP-мастера представить графически с помощью утилиты Network Configuration. Откройте Network Configuration, к примеру, двойным щелчком мыши на подсети. Выберите команду меню View → DP Slaves (Вид → DP-ведомые), чтобы отобразить ведомые. Также можно создать систему DP-мастера (или, точнее, назначить узлы подсети PROFIBUS) с использованием утилиты Network Configuration. Вы можете параметризовать станции, открыв их при помощи Hardware Configuration. Здесь также вы должны сначала установить интеллектуальные DP-ведомые перед тем, как встроить их в систему DP-мастера.

Конфигурирование DP-мастера

Убедитесь в том, что у вас есть созданные с помощью SIMATIC-менеджера проект и S7-станция. Откройте S7-станцию и создайте монтажную стойку (см. параграф 2.3 «Конфигурирование станций»). Теперь перетащите в конфигурационную таблицу монтажной стойки модуль DP-мастера. Вы, возможно, уже выбрали CPU с DP-соединением. Ниже в строке отображен DP-мастер с соединением с системой DP-мастера в окне станции (жирная штриховая черно-белая линия).

При помещении модуля DP-мастера вы должны выбрать в окне подсеть PROFIBUS, которой должна быть назначена система DP-мастера, и адрес узла, предназначенный для назначения DP-мастеру. В этом окне вы также можете создать новую подсеть PROFIBUS.

Если нет доступной системы DP-мастера (она может быть затемнена объектом или находиться вне видимой области), создайте ее, выбрав DP-мастер в конфигурационном окне и команду меню Insert → DP Master System (Вставка → Система DP-мастера). Можно изменить адрес узла и подключение к подсети PROFIBUS путем выбора модуля и внесения изменений после нажатия кнопки «Properties» («Свойства») или выбора вкладки «General» («Общие») диалогового окна, открываемого командой меню Edit → Object Properties (Правка → Свойства объекта).

CP 342-5DP в качестве DP-мастера

Если CP 342-5DP является DP-мастером, поместите его в конфигурационную таблицу станции, отметьте его и выберите команду Edit → Object Properties (Правка → Свойства объекта). На вкладке «Mode» («Режим») установите «DP Master» («DP-мастер»).

На вкладке «Addresses» («Адреса») показаны адреса пользовательских данных, которые CP занял в области адресов CPU. С точки зрения главного CPU CP 342-5DP является «аналоговым модулем» со стартовым адресом модуля и 16 байтами пользовательских данных.

Только стандартные DP-ведомые или ведомые DP S7, которые функционируют как стандартные DP-ведомые, могут быть подключены к CP 342-5DP как DP-мастеру. Подходящие DP-ведомые вы можете найти в каталоге аппаратуры (Hardware Catalog) под «PROFIBUS-DP» и «CP 342-5DP as DP master» («CP 342-5DP как DP-мастер»). Выберите требуемый тип ведомого и перетащите его в систему DP-мастера.

Память передачи (DP-мастера) имеет максимальный размер 240 байтов. Она пересылается целиком с помощью загружаемых блоков FC 1 DP_SEND и FC 2 DP_RECV (которые включены в стандартную библиотеку *Standard Library* в программе коммуникационных блоков *Communication Blocks*).

Консистентные данные охватывают всю память передачи.

Чтение диагностических данных подключенных DP-ведомых можно осуществить с помощью FC 3 DP_DIAG (например, список станций, диагностические данные определенной станции). FC 4 DP_CTRL передает задания управления в CP 342-5DP (к примеру, команды SYNC/FREEZE и CLEAR устанавливают операционное состояние CP 342-5DP).

Команда меню *View → Address Overview* (*Вид → Обзор адресов*) при выбранном CPU или CP 342-5DP покажет вам список назначенных адресов, входов и/или выходов. Вы также можете увидеть существующие промежутки в адресах.

Конфигурирование компактных DP-ведомых

Компактные DP-ведомые находятся в каталоге аппаратуры (Hardware Catalog) под «PROFIBUS-DP» в соответствующем поддиректории, например, ET 200B. Щелкните мышью на выбранном DP-ведомом и перетащите его на пиктограмму системы DP-мастера.

Вы увидите список свойств станции; здесь вы можете установить адрес узла и любой диагностический адрес. Затем DP-ведомый появляется в виде пиктограммы в верхней части окна станции, а в нижней части содержится конфигурационная таблица этой станции.

Двойной щелчок на пиктограмме в верхней части окна станции открывает диалоговое окно с одной или более вкладок, в которой вы можете установить требуемые свойства станции. В нижней части окна вы можете увидеть адреса входов/выходов. Двойной щелчок на строке адреса отобразит окно, где можно изменить предлагаемые адреса.

Нижняя секция окна может содержать конфигурационную таблицу выделенного DP-ведомого или системы мастера (переключается кнопкой со стрелкой).

Конфигурирование модульных DP-ведомых

Модульные DP-ведомые находятся в каталоге аппаратуры (Hardware Catalog) под «PROFIBUS-DP» в соответствующем поддиректории, например, ET 200M.

Щелкните на выбранном интерфейсном модуле (базовом модуле) и перетащите его на пиктограмму системы DP-мастера. Это действие отобразит список свойств станции; здесь вы можете установить адрес узла и диагностический адрес. Затем DP-ведомый появится в качестве пиктограммы в верхней части окна станции, а в нижней части содержится конфигурационная таблица этой станции.

Теперь в конфигурационную таблицу поместите модули, которые находятся в каталоге аппаратуры (Hardware Catalog) *под выбранным интерфейсным модулем (!)*. Двойной щелчок на строке откроет список свойств модуля и позволит вам задать параметры модуля.

Обратите внимание на то, что область адресов для ведомого в DP-мастере и область адресов интерфейсного модуля DP-ведомого ограничены. Например, граничное значение в CPU 315-2DP, являющемся DP-мастером, составляет 122 байта входов и 122 байта выходов на одного ведомого (восемь 8-канальных модулей в ET 200M могут превысить этот лимит: $8 \times 16 = 128$ байтов), а граничное значение в ET 200X, являющемся DP-ведомым, составляет 104 байта входов и 104 байта выходов.

Конфигурирование CPU со встроенным DP-интерфейсом как интеллектуального DP-ведомого

При наличии соответствующим образом оборудованного CPU вы можете параметризовать станцию либо как станцию DP-мастера, либо как станцию DP-ведомого. Перед подключением станции в качестве DP-ведомого к системе DP-мастера она должна быть создана. Соответствующая процедура полностью совпадает с набором действий для «нормальной» станции; вставьте S7-станцию в проект, используя SIMATIC-менеджер, и откройте объект *Hardware (Apparatur)*. В Hardware Configuration перетащите монтажную стойку в окно и разместите требуемые модули. Для конфигурирования DP-ведомого достаточно поместить CPU; другие модули вы можете добавить позже.

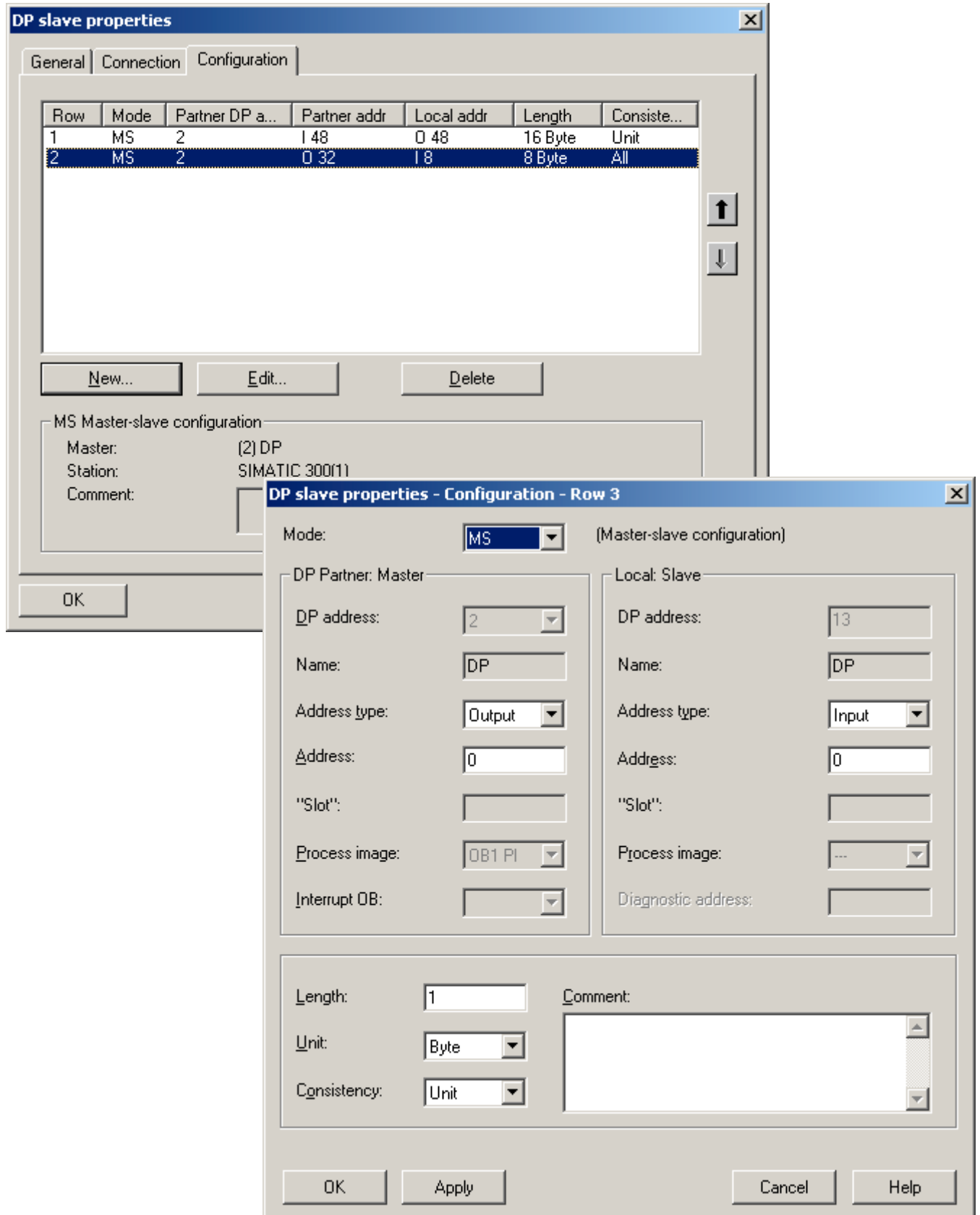
Когда добавляется CPU, раскрывается список свойств интерфейса PROFIBUS. Здесь вы должны назначить подсеть DP-интерфейсу, а также адрес. Если к этому моменту в проекте подсети PROFIBUS не существует, то вы можете создать новую, нажав кнопку «New» («Новый»). Это подсеть, к которой впоследствии будет подключен интеллектуальный ведомый.

Можно открыть список свойств интерфейса, выбрав DP-интерфейс и команду меню Edit → Object Properties (Правка → Свойства объекта), или дважды щелкнув на интерфейсе. На вкладке «Mode» («Режим») отметьте опцию «DP Slave» («DP-ведомый»). Теперь вы можете сконфигурировать интерфейс пользовательских данных на вкладке «Configuration» («Конфигурирование») с точки зрения DP-ведомого (рисунок 20.10); параграф 20.4.1 «Адресация распределенных входов/выходов» содержит информацию об интерфейсе пользовательских данных в подразделе «Память передачи в интеллектуальных DP-ведомых».

Размер и структура памяти передачи определяется версией CPU. В CPU 315-2DP, например, вы можете поделить всю память передачи максимально на 32 адресных области, к которым можно осуществлять отдельное обращение. Такие области адресов могут занимать до 32 байтов. Вся память передачи может иметь 122 адреса входов и 122 адреса выходов.

Адреса, определенные здесь, располагаются в адресном томе ведомого CPU. Эти адреса не должны перекрываться с адресами централизованных или распределенных модулей в станции DP-ведомого. Наименьший адрес области адресов является «стартовым адресом модуля».

Пользовательская программа в ведомом CPU получает диагностическую информацию от DP-мастера через диагностический адрес, заданный на этой вкладке.

**Рисунок 20.10**

Конфигурирование памяти передачи интеллектуального ведомого со встроенным DP-интерфейсом

Конфигурирование интеллектуального DP-ведомого завершается командой меню Station → Save and Compile (Станция → Сохранить и скомпилировать). Подключение интеллектуального DP-ведомого к системе DP-мастера описывается ниже.

Конфигурирование BM 147/CPU как интеллектуального DP-ведомого

Если вы хотите создать интеллектуальный DP-ведомый на базе ET 200X, сначала в SIMATIC-менеджере под проект поместите станцию SIMATIC 300 и откройте объект *Hardware*.

В утилите Hardware Configuration перетащите из каталога аппаратуры (Hardware Catalog) объект *MB147/CPU*, находящийся в нем под «PROFIBUS-DP» и «ET200X», в свободное окно или выберите его двойным щелчком. Затем в появившемся списке свойств DP-интерфейса укажите адрес узла и подсеть PROFIBUS (или создайте ее и назначьте DP-интерфейсу). Теперь у вас имеется конфигурационная таблица, как в случае станции SIMATIC 300. Отображаемый CPU представляет здесь интерфейсный модуль BM 147 с развитой логикой или станцию ET 200X. У этого CPU в таблице адресов нет MPI-адреса, так как он не оснащен MPI-интерфейсом (BM 147/CPU программируется через MPI-интерфейс главной станции).

Двойным щелчком на строке CPU можно открыть окно его свойств; двойной щелчок на DP-интерфейсе открывает окно свойств интерфейса. Здесь можно установить области адресов для интерфейса пользовательских данных с точки зрения DP-ведомого. У BM 147 стартовый адрес зафиксирован на 128; максимальный размер области пользовательских данных составляет 32 байта входов и 32 байта выходов. Вы можете поделить эту область на восемь подобластей с различной консистентностью данных. Диагностический адрес установлен в 127; программа ведомого получает диагностическую информацию от DP-мастера через этот адрес.

Дальнейшее конфигурирование станции ET 200X выполняется точно так же, как для станции S7-300 с фиксированной адресацией слота. Вы можете смонтировать только модули, имеющиеся в каталоге аппаратуры (Hardware Catalog) под «BM147/CPU».

Завершается конфигурирование интеллектуального DP-ведомого командой меню Station → Save and Compile (Станция → Сохранить и скомпилировать). Подключение интеллектуального DP-ведомого к системе DP-мастера рассматривается ниже.

Конфигурирование IM 151/CPU как интеллектуального DP-ведомого

Если вы хотите построить ET 200S в качестве интеллектуального DP-ведомого, сначала в SIMATIC-менеджере в проекте вставьте станцию SIMATIC 300 и откройте объект *Hardware*.

Теперь перетащите объект *IM151/CPU*, находящийся в каталоге аппаратуры (Hardware Catalog) под «PROFIBUS-DP2» и «ET200S», в свободное окно или выберите его, дважды щелкнув на нем левой кнопкой мыши. Появится список свойств DP-интерфейса, в котором выберите адрес узла и подсеть PROFIBUS (или сгенерируйте ее и назначьте DP-интерфейсу). Вы увидите конфигурационную таблицу, подобную

конфигурационной таблице станции SIMATIC 300. Отображенный CPU представляет интерфейсный модуль IM 151 с развитой логикой станции ET 200S. Этот CPU не имеет MPI-интерфейса (IM 151/CPU программируется через MPI-интерфейс главной станции).

Двойным щелчком на строке CPU открывается окно свойств CPU; двойной щелчок на DP-интерфейсе приведет к открытию окна свойств интерфейса. Вы можете в этом пункте установить диапазоны адресов для интерфейса пользовательских данных с точки зрения DP-ведомого. Для IM 151/CPU максимальный размер области пользовательских данных составляет 64 байта. Вы можете разбить эту область на восемь подобластей с различными консистентными данными. Ведомая программа получает диагностическую информацию от DP-мастера через диагностический адрес.

Дальнейшее конфигурирование станции ET 200S проводится точно так же, как для станции S7-300 с фиксированной адресацией слота. Вы можете смонтировать только модули, имеющиеся в каталоге аппаратуры (Hardware Catalog) под «IM151/CPU». Завершает конфигурирование интеллектуального DP-ведомого команда меню Station → Save and Compile (Станция → Сохранить и скомпилировать). Интегрирование интеллектуального DP-ведомого в систему DP-мастера описывается ниже.

Конфигурирование станции S7 с CP 342-5B3 как интеллектуального DP-ведомого

Если вы в SIMATIC-менеджере вставляете станцию S7-300, откройте объект *Hardware* и приступите к конфигурированию «нормальной» станции S7-300. Помимо прочего, скомпонуйте в конфигурационной таблице коммуникационный процессор CP 342-5DP.

При вставке станции появляется список свойств DP-интерфейса; подсеть, к которой позже будет подключен интеллектуальный DP-ведомый, должна быть здесь назначена DP-интерфейсу, а также вы должны назначить адрес узла.

Чтобы открыть окно свойств, отметьте CP 342-5DP и выберите команду меню Edit → Object Properties (Правка → Свойства объекта) или щелкните дважды на CP 342-5DP. На вкладке «Mode» («Режим») выберите опцию «DP Slave» («DP-ведомый»).

На вкладке «Addresses» («Адреса») отображен интерфейс пользовательских данных с точки зрения ведомого CPU (стартовый адрес и размером 16 байтов). Размер памяти передачи CP 342-5DP, используемого в качестве DP-ведомого, может составлять до 86 байтов, и вы можете разделить ее на различные области адресов после подключения к мастер-системе.

Команда Station → Save and Compile (Станция → Сохранить и скомпилировать) завершает конфигурирование интеллектуального DP-ведомого.

Подключение интеллектуального DP-ведомого к DP-мастеру

Вы должны располагать созданным проектом, сконфигурированной станцией DP-мастера и интеллектуальным DP-ведомым (в любом случае, по крайней мере, с DP-интерфейсом). DP-мастер и DP-ведомый должны быть сконфигурированы для одной и той же подсети PROFIBUS.

Откройте главную станцию (мастер-станцию); система DP-мастера (жирная штриховая черно-белая линия) должна существовать, если нет, создайте ее при помощи команды **Insert → DP Master System** (**Вставка → Система DP-мастера**). В каталоге аппаратуры (**Hardware Catalog**) под «PROFIBUS-DP» и «Configured Stations» («Сконфигурированные станции») вы найдете объекты, представляющие интеллектуальные ведомые: «CPU31x-2DP» представляет станции S7-300 со встроенным DP-ведомым, «X-VM147 / CPU» представляет станции ET 200X с VM147/CPU, «ET200S/CPU» представляет интеллектуальный DP-ведомый ET200S и «S7-300 CP342-5 DP» представляет станции S7-300 с CP342-5 в качестве интерфейсного модуля DP-ведомого. Выберите требуемый тип ведомого и перетащите его в систему DP-мастера.

CPU, ET200X или ET200S как DP-ведомые

Перенос DP-ведомого в систему DP-мастера или двойной щелчок на нем откроет список свойств. Ведомые, уже сконфигурированные для данной подсети PROFIBUS, приведены на вкладке «Connection» («Соединение»). Выберите требуемый ведомый и нажмите на кнопку «Connect» («Соединить»). Это приведет к проведению активации соединения в нижней части того же диалогового окна.

На вкладке «General» («Общие») вы можете установить диагностический адрес DP-ведомого с точки зрения главной станции.

На вкладке «Configuration» («Конфигурирование») вы можете теперь установить адреса интерфейса пользовательских данных с точки зрения DP-мастера. Адреса выходов мастера являются адресами входов ведомого и наоборот. В параграфе 20.4.1 «Адресация распределенных входов/выходов» в разделе «Память передачи интеллектуальных DP-ведомых» содержится более подробная информация об интерфейсе пользовательских данных.

CP 342-5DP как DP-ведомый

При перетаскивании DP-ведомого в систему DP-мастера и по двойному щелчку на нем открывается список свойств. Уже сконфигурированные для данной подсети PROFIBUS ведомые приведены на вкладке «Connection» («Соединение»). Выберите требуемый ведомый и нажмите на кнопку «Connect» («Соединить»). После этого в нижней части того же диалогового окна отобразится активное соединение.

Когда отмечен DP-ведомый, его конфигурационная таблица отображается в нижнем разделе окна станции. Теперь вы можете структурировать память передачи: из каталога аппаратуры (**Hardware Catalog**) (под используемым CP) выберите «Universal submodule» («Универсальный подмодуль»), перетащите его в строку конфигураци-

онной таблицы или отметьте строку и дважды щелкните на «Universal submodule» («Универсальный подмодуль»). Поместите по одному универсальному модулю для каждой отдельной (консистентной) области данных памяти передачи; максимальное количество – 32.

Чтобы открыть окно, в котором вы можете определить свойства адресной области, отметьте универсальный модуль и вызовите команду меню Edit → Object Properties (Правка → Свойства объекта) или дважды щелкните на строке таблицы: на пустом пространстве, области входов или выходов или обоих. Задайте стартовый адрес и размер области.

Определяемые здесь адреса расположены в области адресов главного CPU. Область может занимать до 64 байтов; максимальный общий размер области памяти передачи равен 86 байтам.

Если CP 342-5DP является DP-мастером, то память передачи можно не структурировать, потому что CP 342-5DP пересылает всю область передачи целиком.

При разделении памяти передачи области адресов группируются вместе, без промежутков, начиная с байта 0. Вы можете обратиться ко всей назначенной памяти передачи в ведомом CPU с использованием загружаемых блоков FC 1 DP_SEND и FC 2 DP_RECV (включенные в стандартную библиотеку *Standard Library* под программой коммуникационных блоков *Communication Blocks*).

Консистентность данных покрывает всю память передачи.

На вкладке «General» («Общие») вы можете установить диагностический адрес DP-ведомого с точки зрения главной станции. Диагностические данные считываются с помощью FC 3 DP_DIAG (в главной станции). Параграф 20.4.1 «Адресация распределенных входов/выходов» в разделе «Память передачи интеллектуальных DP-ведомых» содержит более подробную информацию об интерфейсе пользовательских данных.

Файлы GSE

Вы можете осуществить «пост-вставку» DP-ведомых, которые не включены в каталог модулей. Для этого вам потребуется типовой файл, специализированный для ведомого. (GSE-файл, файл базы данных устройств). В утилите Hardware Configuration выберите команду меню Options → Install New GSE (Опции → Инсталлировать новый GSE) и укажите директорию GSE-файла в открывшемся окне. Здесь вы можете задать пиктограмму, которую STEP 7 будет использовать для графического представления DP-ведомого. STEP 7 принимает GSE-файл и отображает ведомого в каталоге аппаратуры (Hardware Catalog) под «Additional Filed Devices» («Дополнительные устройства из файлов»).

Вы можете скопировать файлы GSE, которые уже имеются в другом S7-проекте, в текущий проект при помощи пункта меню Options → Import Station GSE (Опции → Импортировать станцию GSE).

STEP 7 сохраняет GSE-файлы в директории ... \Step7\S7data\gsd. GSE-файлы, удаленные при инсталлировании или импортировании в более позднее время, записываются в поддиректорию ... \gsd\bkpx. Они могут быть восстановлены отсюда командой Options → Install New GSE (Опции → Инсталлировать новый GSE).

Конфигурирование PROFIBUS PA

Для конфигурирования мастер-системы PROFIBUS PA и параметризации полевых устройства PA вам потребуется дополнительное программное обеспечение SIMATIC PDM. С помощью утилиты Hardware Configuration вы можете установить подключение к системе DP-мастера с использованием DP/PA-Link (модуля связи DP/PA): перетащите из каталога аппаратных средств (Hardware Catalog) в систему DP-мастера интерфейсный модуль IM 157. Одновременно с DP-ведомым создается система PA-мастера в своей собственной подсети PROFIBUS (45,45 Кбит/с); она обозначается жирной штриховой черно-белой линией.

DP/PA-интерфейс пересылает между шинными системами немодифицированные и неинтерпретированные данные; поэтому он не параметризуется. Полевые устройства PA адресуются DP-мастером. Они могут быть объединены как стандартные DP-ведомые в утилите STEP 7 Hardware Configuration через GSE-файл. Затем вы сможете найти полевые устройства PA в каталоге аппаратуры (Hardware Catalog) под «PROFIBUS-DP» и «Other Field Devices» («Другие полевые устройства»).

Конфигурирование DP/AS-i-Link

Конфигурирование DP/AS-Interface-Link (модуль связи DP/AS-интерфейса) осуществляется подобно настройке модульного DP-ведомого. Вы найдете модули, которые можно перенести в систему DP-мастера, такие как DP/AS-i-Link 20, в каталоге аппаратных средств (Hardware Catalog) под «PROFIBUS-DP» и «DP/AS-i». В открывающихся окнах можете установить сначала конфигурацию уставки (setpoint) (от 16 до 20 байтов), а затем адрес узла.

В DP/AS-i-Link 20 вы можете определить 16 байтов выходов/выходов в качестве конфигурации уставки с дополнительными 4 байтами для команд управления. В этом случае нижняя часть окна Hardware Configuration предоставляет 16 байтов пользовательских данных с адресами в образе процесса и 4 байта команд с адресами, например, от 512.

Выберите DP-ведомый и затем команду Edit → Object Properties (Правка → Свойства объекта) или дважды щелкните на DP-ведомом, чтобы открыть окно, в котором можно изменить адреса, предлагаемые утилитой Hardware Configuration, а также установить образ подпроцесса, предоставляемый соответствующим CPU.

Отметьте DP-ведомый и выберите команду Edit → Object Properties (Правка → Свойства объекта) или дважды щелкните на DP-ведомом, чтобы открыть окно свойств ведомого. На вкладке «Parameterize» («Параметризация») вы можете установить параметры AS-i-ведомых с 4 битами для каждого ведомого.

Система AS-i-мастера с AS-i-ведомыми утилитой Hardware Configuration не отображается.

Конфигурирование групп SYNC/FREEZE

Команда управления SYNC побуждает объединенных в группу DP-ведомых одновременно (синхронно) вывести их выходные состояния. Команда управления FREEZE «замораживает» текущие входные сигнальные состояния одновременно (синхронно) у всех DP-ведомых, объединенных в группу, чтобы затем позволить DP-мастеру циклически опросить их. Команды управления UNSYNC и UNFREEZE отменяют действие команд SYNC и FREEZE соответственно.

Необходимым условием является то, чтобы DP-мастер и DP-ведомые имели соответствующую функциональность. Из объектных свойств ведомого вы можете узнать, какие команды он поддерживает; выберите DP-ведомый, нажмите Edit → Object Properties (Правка → Свойства объекта), перейдите к пункту «SYNC/FREEZE Capabilities» («Возможности SYNC/FREEZE») и вкладке «General» («Общие»).

Для одной системы DP-мастера вы можете сформировать до 8 групп SYNC/FREEZE, предназначенных для выполнения команды SYNC или команды FREEZE, либо обе эти команды. Можно назначить в группу любой DP-ведомый; в специальной версии CP 342-5DP один DP-ведомый может быть представлен в восьми (максимум) группах.

Когда вы вызываете SFC 11 DPSYC_FR, вы заставляете пользовательскую программу выдать команду группе (см. параграф 20.4.3 «Системные функции для распределенных входов/выходов»). В этом случае DP-мастер посылает соответствующую команду одновременно всем DP-ведомым определенной группы.

Конфигурирование группы SYNC/FREEZE осуществляется после конфигурирования системы DP-мастера (все DP-ведомые должны существовать в системе DP-мастера). Отметьте систему DP-мастера (жирная штриховая черно-белая линия) и выберите команду меню Edit → Object Properties (Правка → Свойства объекта). В открывшемся окне на вкладке «Group Properties» («Свойства группы») вы должны сначала установить предназначенные для исполнения групповые команды, затем назначить отдельным группам DP-ведомые на вкладке «Group Assignment» («Назначение групп») (рисунок 20.11).

Здесь вы должны, выбирая в списке по порядку каждый DP-ведомый с его номером узла, указать в каждом случае группу, к которой он должен принадлежать. Если DP-ведомый не может выполнять определенную команду, например, FREEZE, то группы, которые содержат эту команду, не могут быть выбраны, например, все группы с командой FREEZE. Нажатие кнопки «ОК» завершит конфигурирование групп SYNC/FREEZE.

Обратите внимание на то, что при конфигурировании шинных циклов одинаковой длины (эквидистантных) группы 7 и 8 приобретают особое значение.

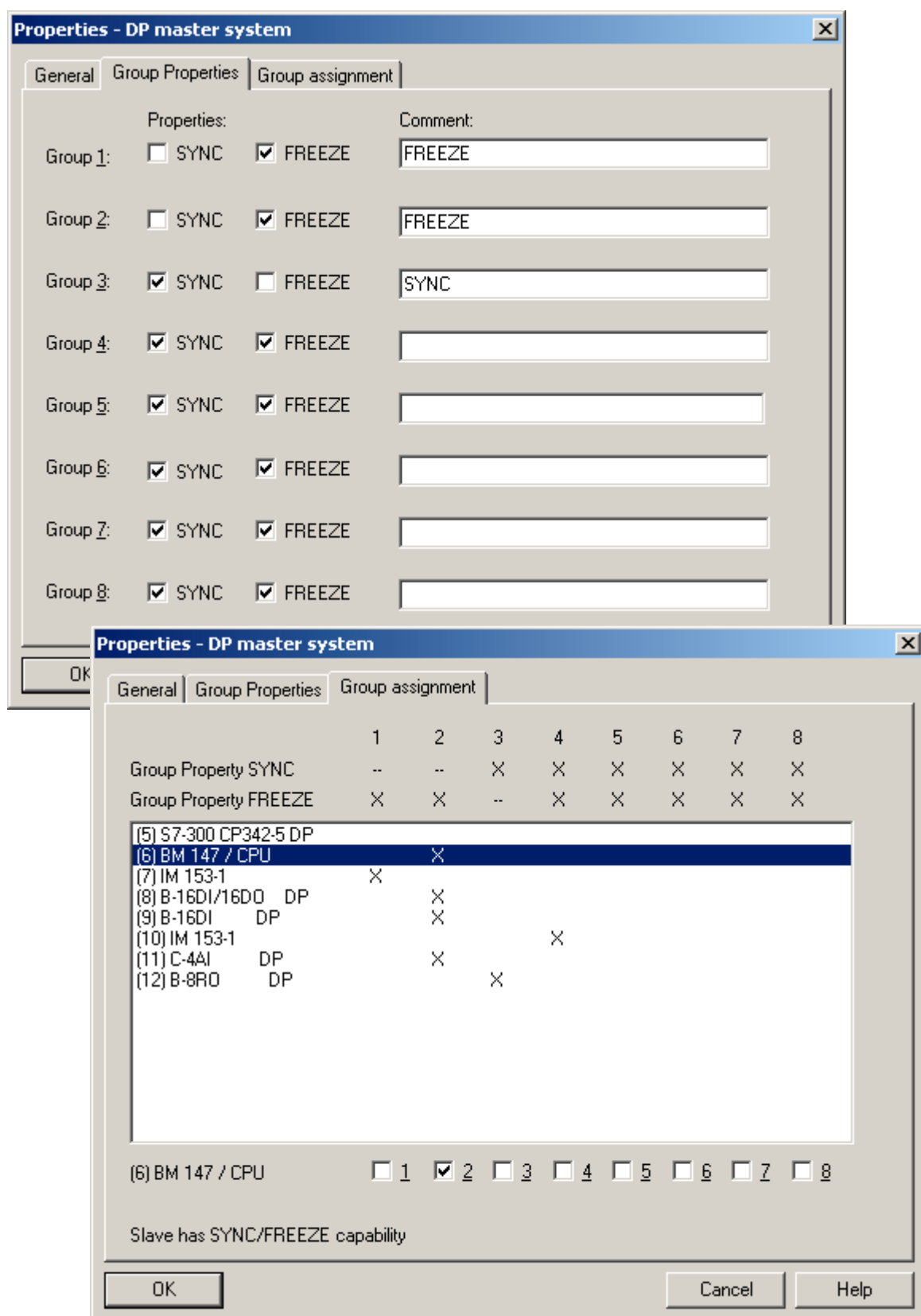


Рисунок 20.11 Конфигурирование групп SYNC и FREEZE

Конфигурирование постоянного времени цикла шины

Обычно DP-мастер управляет назначенными ему DP-ведомыми циклически, без пауз. Посредством коммуникации S7, например, когда программирующее устройство выполняет модификацию функций через подсеть PROFIBUS, это может привести к различиям во временных интервалах. Если, к примеру, выходы должны быть изменены с помощью распределенной периферии через одинаковый интервал времени, то вы можете установить постоянные шинные циклы с использованием соответствующим образом оснащенного DP-мастера. Для этого DP-мастер должен быть только мастером класса 1 в подсети PROFIBUS. Применение постоянного времени цикла шины возможно с профилями шины «DP» и «User-Defined» («Определенный пользователем»).

Вы можете открыть окно свойств подсети PROFIBUS, к примеру, отметив подсеть PROFIBUS и выбрав команду Edit → Object Properties (Правка → Свойства объекта) в утилите конфигурирования сети. На вкладке «Network Settings» («Сетевые установки») нажмите кнопку «Options» («Опции»). На вкладке «Constant Bus Cycle Time» («Постоянное время цикла шины») отметьте флажок (щелкните на квадратном окошке метки) «Activate constant bus cycle time / Recalculate constant bus cycle time» («Активировать постоянное время цикла шины / Пересчитать постоянное время цикла шины»). Вы можете изменить предложенное постоянное время цикла шины, но нельзя выйти за минимальное время, которое показано. Кнопка «Details» («Детали») отображает отдельные компоненты постоянного времени цикла шины. Заметьте, что постоянное время цикла шины увеличивается с ростом числа подключенных напрямую к подсети PROFIBUS устройств программирования и количества интеллектуальных DP-ведомых в системе DP-мастера.

Если вы кроме постоянного времени цикла шины конфигурируете группы SYNC/FREEZE, обратите внимание на следующее:

- для DP-ведомых в группе 7 DP-мастер автоматически инициирует команду SYNC/FREEZE в каждом шинном цикле; инициация через пользовательскую программу не допускается;
- группа 8 используется для сигнала постоянного времени цикла шины и отменена для DP-ведомых; вы не сможете сконфигурировать постоянное время цикла шины, если для группы 8 уже сконфигурированы ведомые.

Конфигурирование прямого обмена данными (горизонтальная коммуникация)

В своей системе DP-мастер эксклюзивно управляет назначенными ему DP-ведомыми. В станциях, оборудованных соответствующим образом, другой узел (мастер или интеллектуальный ведомый, называемый «приемником») может наблюдать подсеть PROFIBUS, чтобы узнать, какие входные данные DP-ведомый («передатчик») посылает «своему» мастеру. Этот прямой обмен данными также называется «горизонтальной коммуникацией». В принципе любые DP-ведомые определенных версий могут функционировать как передатчики в прямом обмене данными.

Прямой обмен данными вы можете сконфигурировать с использованием утилиты Hardware Configuration в окне свойств (Properties) DP-ведомого (приемника), когда все станции в подсети PROFIBUS подключены. Откройте принимающую станцию и отметьте DP-интерфейс, затем выберите команду меню **Edit → Object Properties** (**Правка → Свойства объекта**). Вкладка «Communication» («Коммуникация») содержит интерфейс передачи между DP-ведомым и DP-мастером. Установите здесь рабочий режим DX (direct data exchange – прямой обмен данными) в столбце «Mode» («Режим»). Передатчик, чьи сигналы требуется наблюдать, выберите в разделе «PROFIBUS DP Partner» («Партнер PROFIBUS DP») в столбце «Address» («Адрес»).

Также можно использовать прямой обмен данными между двумя системами DP-мастера на одной подсети PROFIBUS. Таким образом, мастер в системе 1 может наблюдать, например, за данными ведомого из системы 2.

20.4.3 Системные функции для распределенных входов/выходов

Вы можете использовать следующие SFC совместно с распределенными входами/выходами (I/O):

- SFC 7 DP_PRAL
Иницирует прерывание процесса
- SFC 11 DPSYN_FR
Посылает команды SYNC/FREEZE
- SFC 12 D_ACT_DP
Активирует/деактивирует DP-ведомого
- SFC 13 DPNRM_DG
Считывает диагностические данные из стандартного DP-ведомого
- SFC 14 DPRD_DAT
Считывает пользовательские данные из DP-ведомого
- SFC 15 DPWR_DAT
Записывает пользовательские данные в DP-ведомый

Таблица 20.7 содержит параметры этих SFC.

SFC 7 DP_PRAL

Инициация прерывания процесса

С помощью SFC 7 DP_PRAL вы можете инициировать прерывание процесса в DP-мастере, ассоциированном с интеллектуальным ведомым, из пользовательской программы этого ведомого.

Таблица 20.7

Параметры SFC, используемых для обращения к распределенным входам/выходам

SFC	Параметр	Объявление	Тип данных	Содержимое, описание
7	REQ	INPUT	BOOL	Запрос на инициацию при REQ = «1»
	IOID	INPUT	BYTE	B#16#54 = ID входа B#16#55 = ID выхода
	LADDR	INPUT	WORD	Стартовый адрес области адресов в памяти передачи
	AL_INFO	INPUT	DWORD	ID прерывания (передача стартовой информации блоку OB прерывания)
	RET_VAL	OUTPUT	INT	Информация об ошибке
	BUSY	OUTPUT	BOOL	Когда BUSY = «1», подтверждения от DP-мастера еще нет
11	REQ	INPUT	BOOL	Запрос на передачу при REQ = «1»
	LADDR	INPUT	WORD	Сконфигурированный диагностический адрес DP-мастера
	GROUP	INPUT	BYTE	Группа DP-ведомых (из Hardware Configuration)
	MODE	INPUT	BYTE	Команда (см. текст)
	RET_VAL	OUTPUT	INT	Информация об ошибке
	BUSY	OUTPUT	BOOL	Если BUSY = «1», задание еще выполняется
12	REQ	INPUT	BOOL	Запрос на активацию/деактивацию при REQ = «1»
	MODE	INPUT	BYTE	Режим функционирования 0 Проверка, активирован или деактивирован DP-ведомый 1 Активация DP-ведомого 2 Деактивация DP-ведомого 3 Прервать активирование/деактивирование
	LADDR	INPUT	WORD	Диагностический адрес или стартовый адрес модуля DP-ведомого
	RET_VAL	OUTPUT	INT	Результат считывания или информация об ошибке
	BUSY	OUTPUT	BOOL	Если BUSY = «1», задание еще выполняется
13	REQ	INPUT	BOOL	Запрос на чтение при REQ = «1»
	LADDR	INPUT	WORD	Сконфигурированный диагностический адрес DP-ведомого
	RET_VAL	OUTPUT	INT	Информация об ошибке
	RECORD	OUTPUT	ANY	Область назначения для операции чтения диагностических данных
	BUSY	OUTPUT	BOOL	Если BUSY = «1», чтение еще выполняется
14	LADDR	INPUT	WORD	Сконфигурированный стартовый адрес (из I-области)
	RET_VAL	OUTPUT	INT	Информация об ошибке
	RECORD	OUTPUT	ANY	Область назначения для операции чтения пользовательских данных
15	LADDR	INPUT	WORD	Сконфигурированный стартовый адрес (из Q-области)
	RECORD	INPUT	ANY	Область-источник для пользовательских данных, предназначенных для записи
	RET_VAL	OUTPUT	INT	Информация об ошибке

В параметре AL_INFO передается ID прерывания, определенный вами, который пересылается в стартовую информацию организационного блока (ОБ) прерывания, вызываемого в DP-мастере (переменная OBxx_POINT_ADDR). Запрос на прерывание инициируется при REQ = «1»; параметры RET_VAL и BUSY показывают состояние задания. Выполнение задания завершено, когда исполнен ОБ прерывания в DP-мастере.

Память передачи между DP-мастером и интеллектуальным DP-ведомым может быть разбита на отдельные адресные области, которые представляют отдельные модули с точки зрения главного CPU. Наименьший адрес области адресов резервируется под стартовый адрес модуля. Вы можете инициировать прерывание процесса в мастере для каждой из этих областей адресов («виртуальные» модули).

Вы можете определить область адресов при выполнении SFC 7 с помощью параметров IOID и LADDR с точки зрения ведомого CPU (ID входа/выхода и стартовый адрес со стороны ведомого). Стартовая информация ОБ прерывания в этом случае содержит адреса «модуля», инициирующего прерывание с точки зрения главного CPU.

SFC 11 DPSYN_FR

Передача команд SYNC/FREEZE

Используя SFC 11 DPSYN_FR, вы можете посылать команды SYNC, UNSYNC, FREEZE и UNFREEZE группам SYNC/FREEZE, сконфигурированным вами в утилите Hardware Configuration. Передача (SEND) инициируется при REQ = «1» и завершается, когда сигнализируется BUSY = «0».

В параметре GROUP каждая группа занимает 1 бит (от бита 0 для группы 1 до бита 7 для группы 8). Команды в параметре MODE также организованы на уровне битов:

- 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 в DP-ведомых сначала выключены. Входы DP-ведомых последовательно сканируются DP-мастером, выходы DP-ведомых модифицируются; DP-ведомые немедленно передают полученные выходные сигналы в выходные терминалы.

Если вы хотите «заморозить» входные сигналы нескольких DP-ведомых в определенное время, вы должны выдать соответствующей группе команду FREEZE. Затем входные сигналы, имеющие значения на момент «заморозки», последовательно считываются DP-мастером. Эти входные сигналы сохраняют свои значения до другой команды FREEZE, приводящей к проведению DP-ведомыми операции считывания и задержке текущих входных сигналов, или до переключения DP-ведомых обратно в «нормальный» режим командой UNFREEZE.

Если требуется синхронно вывести выходные сигналы нескольких DP-ведомых в определенное время, то сначала нужно выдать команду SYNC соответствующей группе. Тогда адресованные DP-ведомые зафиксируют текущие сигналы на выходных терминалах. Теперь вы можете переслать требуемые сигнальные состояния в DP-ведомые. После передачи вы должны снова выдать команду SYNC; это приведет к одновременному переключению DP-ведомыми полученных выходных сигналов на выходных терминалах. Затем DP-ведомые, к которым происходит обращение, задерживают сигналы на выходных терминалах до переключения на новые входные сигналы с помощью новой команды SYNC или до переключения DP-ведомых обратно в их «нормальный» режим командой UNSYNC.

SFC 12 D_ACT_DP

Активация/деактивация DP-ведомого

SFC 12 D_ACT_DP позволяет вам деактивировать сконфигурированный (и существующий) DP-ведомый, в результате чего DP-мастер не сможет больше к нему обращаться. Выходные терминалы деактивированного выходного ведомого имеют нулевое или заменяющее значение.

Деактивированный ведомый может быть снят с шины без сообщения об ошибке; сообщений о нем, как о сбойном или отсутствующем, не будет. Вызовы организационных блоков обработки асинхронных ошибок OB 85 (ошибки исполнения программы, когда пользовательские данные деактивированного ведомого располагаются в автоматически обновляемом образе процесса) и OB 86 (сбой станции) прекращаются. После деактивации вы не должны больше обращаться к DP-ведомому из программы, иначе возникнет ошибка доступа к входам/выходам.

При помощи SFC 12 D_ACT_DP вы можете вновь активировать выключенный DP-ведомый. Он конфигурируется и параметризуется DP-мастером таким же образом, как и при восстановлении станции. При активации OB 85 и 86 обработки асинхронных ошибок не запускаются. Если параметр BUSY имеет сигнальное состояние «0» после активации, то DP-ведомый может быть доступен из пользовательской программы.

SFC 13 DPNRM_DG

Чтение диагностических данных

SFC 13 DPNRM_DG считывает диагностические данные DP-ведомого. Процедура чтения инициируется при REQ = «1» и завершается, когда возвращается нулевое значение параметра BUSY. Значение функции RET_VAL в этом случае содержит количество прочитанных байтов. В зависимости от ведомого диагностические данные могут включать от 6 до 240 байтов. Если имеется более 240 байтов, то первые 240 байтов пересылаются, и устанавливается соответствующий бит переполнения данных.

Параметр RECORD записывается в область, в которой хранятся считываемые данные. В качестве фактических параметров допустимы переменные типов ARRAY и

STRUCT, а также указатель ANY типа данных BYTE (например, P#DBzDBXu.zBYTEnnn).

SFC 14 DPRD_DAT

Чтение пользовательских данных

SFC 14 DPRD_DAT считывает консистентные пользовательские данные длиной 3 байта или более 4 байтов из DP-ведомого. Указать размер консистентных данных вы можете при параметризации DP-ведомого.

Параметр LADDR получает стартовый адрес модуля DP-ведомого (область входов).

В параметр RECORD записывается область, в которой хранятся считываемые данные. В качестве фактических параметров допустимы переменные типов ARRAY и STRUCT, а также указатель ANY типа данных BYTE (например, P#DBzDBXu.zBYTEnnn).

SFC 15 DPWR_DAT

Запись пользовательских данных

SFC 15 DPWR_DAT записывает консистентные пользовательские данные длиной 3 байта или более 4 байтов в DP-ведомый. Размер консистентных данных вы можете задать при параметризации DP-ведомого.

Параметр LADDR получает стартовый адрес модуля DP-ведомого (область входов).

Параметр RECORD записывается в область, в которой хранятся записываемые данные. В качестве фактических параметров допустимы переменные типов ARRAY и STRUCT, а также указатель ANY типа данных BYTE (например, P#DBzDBXu.zBYTEnnn).

20.5 Коммуникация глобальных данных

20.5.1 Основы

Коммуникация глобальных данных (global data communication, GD-коммуникация) представляет собой службу коммуникации, встроенную в операционную систему CPU и используемую для обмена небольшими объемами данных, некритичных ко времени, через MPI-шину. Глобальные данные, которые можно передавать, включают

- Входы и выходы (образы процесса),
- Меркеры,
- Данные в блоках данных,
- Значения таймеров и счетчиков в качестве данных, предназначенных для передачи.

Требуется, чтобы CPU были объединены в сеть посредством MPI-интерфейса или соединены через K-шину (или C-шину) как в монтажной стойке S7-400. Для того, чтобы можно было сконфигурировать GD-коммуникацию, все CPU должны находиться в одном и том же проекте STEP 7.

Службе циклической GD-коммуникации операционная система не требуется: имеются системные функции S7-400, доступные для событийной GD-коммуникации.

Заметьте, что принимающий CPU не подтверждает (не квитирует) прием глобальных данных. Таким образом, передатчик не получает никакого отклика о том, получил ли приемник данные, и если так, то какие. Однако, вы можете отобразить состояние коммуникации между двумя CPU, а также общее состояние всех GD-циклов (GD cycles) CPU.

Отправление и получение глобальных данных управляется с помощью частоты или скорости сканирования. Она определяет количество циклов (пользовательской программы), после которых CPU посылает или получает данные. Передача и прием происходят синхронно между передатчиком и приемником каждый раз в контрольной точке цикла, то есть после циклического выполнения программы и перед началом нового программного цикла (как обновление образа процесса, например).

Обмен данным между CPU, сгруппированными в GD-циклы, происходит с использованием пакетов данных (GD-пакетов, GD packets).

GD-цикл

Центральные процессоры (CPU), которые обмениваются общими GD-пакетами, образуют GD-цикл.

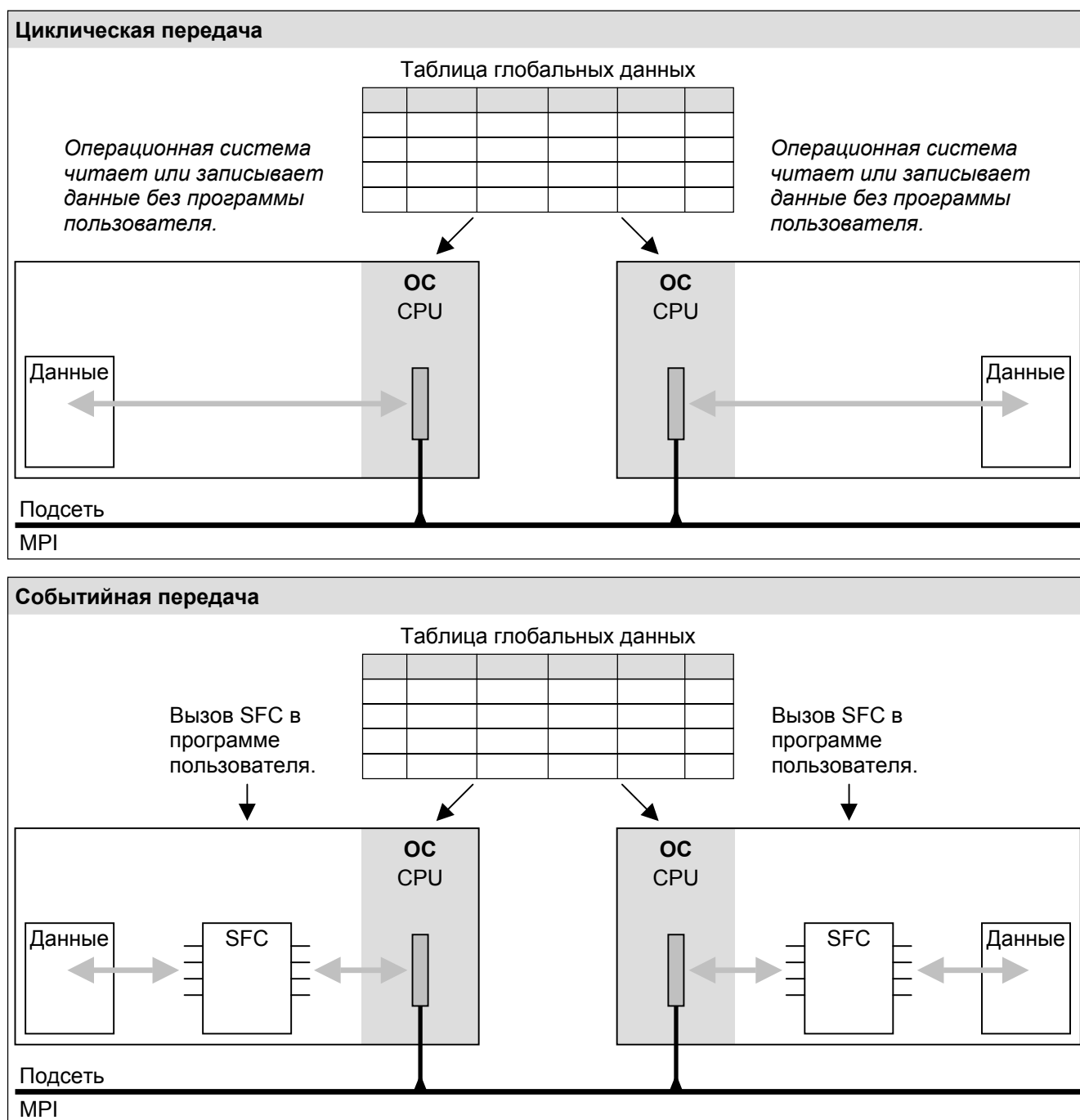


Рисунок 20.12 Коммуникация глобальных данных

GD-цикл может быть одним из следующих:

- Односторонним соединением CPU, который посылает GD-пакет другим нескольким CPU, получающим в этом случае данный пакет.
- Двусторонним соединением между двумя CPU, где каждый из этих CPU может посылать GD-пакет другому.

- Двусторонним соединением между тремя CPU, при котором каждый из этих трех CPU может отправлять один GD-пакет другим двум CPU (только CPU S7-400).

В одном GD-цикле обмениваться данными друг с другом могут до 15 CPU. Также один CPU может принадлежать нескольким GD-циклам.

В таблице 20.8 показаны ресурсы отдельных CPU.

Таблица 20.8 Ресурсы CPU для коммуникации глобальных данных

GD-ресурсы	CPU 312 CPU 313 CPU 314	CPU 315 CPU 316	CPU 318	CPU 412 CPU 413 CPU 414	CPU 416 CPU 417
Максимальное число GD-циклов для CPU	4	4	8	8	16
Получаемых GD-пакетов для CPU	4	4	16	16	32
Получаемых GD-пакетов для цикла	1	1	2	2	2
Отправляемых GD-пакетов для CPU	4	4	8	8	16
Отправляемых GD-пакетов для цикла	1	1	1	1	1
Максимальный размер GD-пакета	32 байта	32 байта	64 байта	64 байта	64 байта
Максимальная консистентность данных	8 байтов	8 байтов	32 байта	16 байтов	32 байта

GD-пакет

GD-пакет включает в свой состав заголовок пакета и один или более элементов глобальных данных (GD-элементов):

- Заголовок пакета (8 байтов),
- ID первого GD-элемента (2 байта),
- Пользовательские данные первого GD-элемента (x байтов),
- ID второго GD-элемента (2 байта),
- Пользовательские данные второго GD-элемента (x байтов),
- и так далее.

Каждый GD-элемент состоит из двух байтов описания и фактических сетевых данных. Для передачи байта памяти (или меркерной памяти) требуется 3 байта, для слова памяти требуется 4 байта, для двойного слова памяти необходимо 6 байтов.

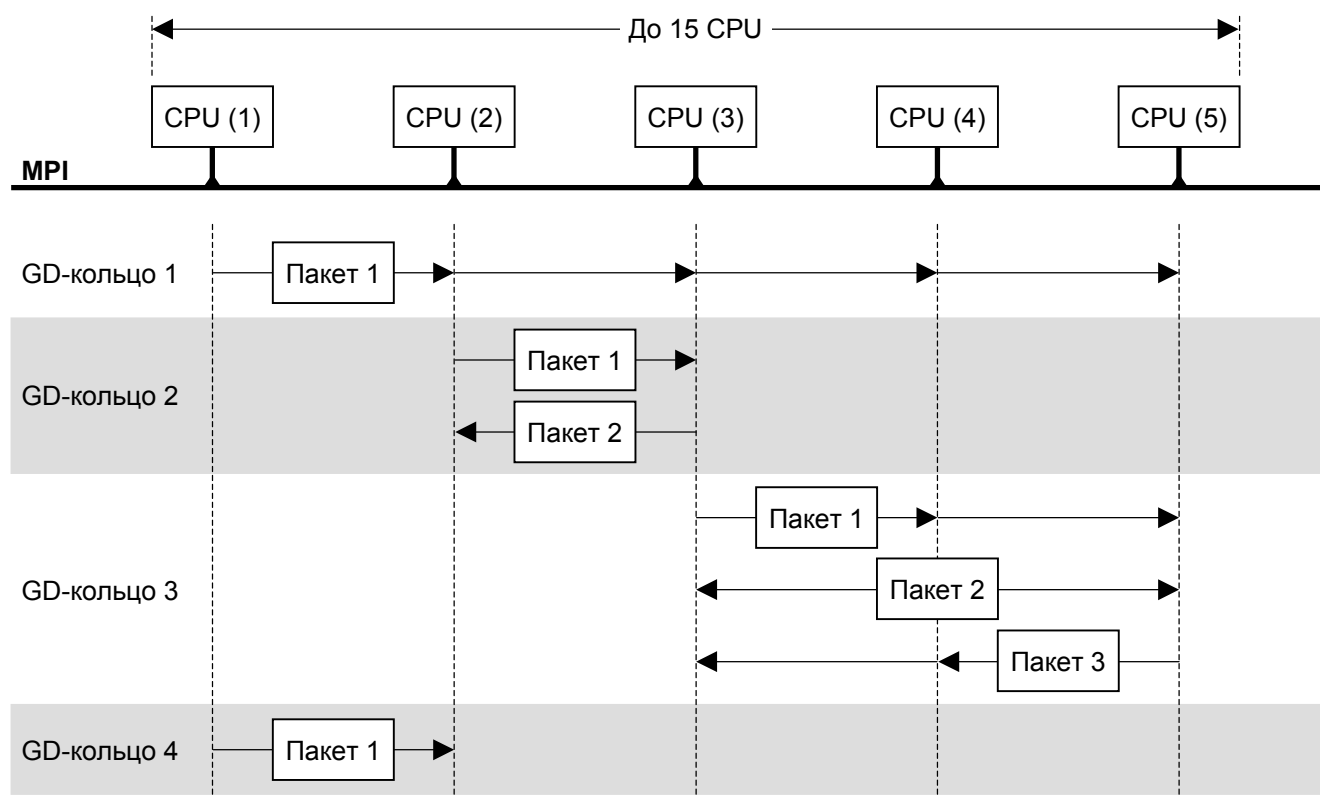


Рисунок 20.13 Пример GD-цикла

Булевская переменная занимает 1 байт сетевых данных; таким образом, ей требуется то же пространство, что и для переменной размером в байт. Двухбайтные значения таймера и счетчика занимают каждый 4 байта в GD-пакете.

GD-элемент также может быть областью адресов. Например, MB 0:15 представляет область от байта памяти MB 0 до MB 15, а DB20.DBW14:8 представляет область данных, расположенную в DB 20, и которая начинается со слова данных DBW 14 и включает 8 слов данных.

Максимальный размер GD-пакета составляет 32 байта в S7-300 и 64 байта для S7-400. Максимально число байтов сетевых данных в пакете достигается при передаче только одного GD-элемента, включающего до 22 байтов в S7-300 и до 54 байтов для S7-400.

Консистентность данных

Консистентность данных охватывает один GD-элемент. Если GD-элемент перезаписывает переменную, определяемую типом CPU, то применяются условия для области, указанные в таблице 20.8.

Если GD-элемент больше, чем размер консистентности данных, то создается блок с консистентными данными соответствующего размера, начиная с первого байта.

20.5.2 Конфигурирование GD-коммуникации

Требования

У вас должен быть созданный проект, и должны быть в наличии доступная MPI-подсеть и сконфигурированные S7-станции. В станциях должен быть, по крайней мере, CPU. Установить MPI-адрес и выбрать MPI-подсеть, с которой соединен CPU, вы можете, нажав кнопку «Properties» («Свойства») MPI-интерфейса на вкладке «General» («Общие») окна свойств CPU (способ вызова: двойной щелчок на строке CPU в утилите Hardware Configuration или на строке подмодуля MPI-интерфейса).

Таблица глобальных данных

Конфигурирование GD-коммуникации осуществляется путем заполнения таблицы. Отметив пиктограмму подсети MPI в SIMATIC-менеджере или утилите Hardware Configuration, при помощи команды меню Options → Define Global Data (Опции → Определение глобальных данных) вы можете вызвать пустую таблицу. Выберите столбец и затем нажмите Edit → CPU (Правка → CPU). В окне выбора проекта, которое после этого откроется, отметьте в его левой части станцию, а в правой части выберите CPU. Этот CPU нажатием кнопки «ОК» принимается в таблицу глобальных данных.

Повторите эти действия для других CPU, участвующих в GD-коммуникации. Таблица глобальных данных (GD-таблица) может включать в себя до 15 столбцов.

Чтобы сконфигурировать передачу данных между CPU, выберите первую строку под передающим CPU и укажите адрес, значение которого (находящееся по этому адресу) должно быть передано (завершается операция нажатием RETURN).

С помощью пункта меню Edit → Sender (Правка → Передатчик) вы можете определить это значение как предназначенное для передачи, оно будет отмечено знаком префикса «>» и затенено. В этой же строке под принимающим CPU вы можете ввести адрес, который должен принять значение (свойство «Receiver», «Приемник» установлено по умолчанию). Вы можете использовать функции таймера и счетчика только как передатчики; приемник должен быть адресом размеров в слово для каждой функции таймера или счетчика.

Строка может содержать несколько приемников, но только один передатчик (таблица 20.9). После ее заполнения выберите команду меню GD Table → Compile (GD-таблица → Компилировать).

После компиляции (фаза 1) созданных системных данных для GD-коммуникации достаточно. Если вы конфигурируете также GD-состояние (GD-status) (состояние GD-коммуникации) и частоты сканирования, вы должны после этого скомпилировать GD-таблицу во второй раз.

Таблица 20.9 Пример GD-таблицы с информацией о состоянии и частотах сканирования

GD-идентификатор	Станция 417 \ CPU417 (3)	Станция 414 \ CPU414 (4)	Станция 416 \ CPU416 (5)	Станция 315 ведомый \ CPU315 (7)	Станция 314CP \ CPU314 (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	>C10:10	DB3.DBW20:10			
GDS 3.1			MD96		
SR 3.1	0	0	44	8	8
GD 3.1.1			>MW98	DB10.DBW220	DB10.DBW220

GD ID

После компилирования без возникновения ошибок STEP 7 завершает создание столбца «GD ID» («GD-идентификатор»). GD ID показывает вам, как передаваемые данные структурированы при образовании GD-циклов, GD-пакетов и GD-элементов. Например, GD ID «GD 2.1.3» соответствует GD-циклу 2, GD-пакету 1, GD-элементу 3. В таком случае вы можете найти назначение ресурса (число GD-циклов) для CPU в столбце CPU таблицы глобальных данных.

GD-состояние

После компиляции вы можете ввести адрес для состояния коммуникации в таблицу глобальных данных с помощью пункта меню View → GD Status (Вид → GD-состояние). Общее состояние (GST) показывает состояние всех коммуникационных соединений в таблице. Состояние (GDS) отображает состояние коммуникационного соединения (передаваемый GD-пакет). В каждом случае состояние занимает двойное слово.

Частоты сканирования

Службе GD-коммуникации требуется значительная часть времени исполнения операционной системы CPU и время передачи по шине MPI. Чтобы эта «коммуникационная загрузка» была минимальна, возможно установить «частоту сканирования» («scan rate»). Частота сканирования определяет число программных циклов, после которых данные (или точнее GD-пакет) должны быть отосланы или приняты.

Так как данные не обновляются в каждом программном цикле с частотой сканирования, вы не должны использовать передачу критичных ко времени данных посредством этого вида коммуникации.

После первой (без ошибок) компиляции вы можете воспользоваться командой меню View → Scan Rates (Вид → Частоты сканирования), чтобы самостоятельно задать частоты сканирования (SR) для каждого GD-пакета и каждого CPU. Частота сканирования устанавливается в качестве стандартной таким образом, что в случае «пустого» CPU (без программы пользователя) GD-пакеты передаются и принимаются приблизительно каждые 10 мс. Если затем загружается пользовательская программа, временные интервалы увеличиваются.

Вы можете ввести частоты сканирования из области между 1 и 255. Обратите внимание на то, что по мере уменьшения частоты сканирования коммуникационная нагрузка в CPU увеличивается. Для того, чтобы коммуникационная нагрузка находилась в приемлемых границах, установите частоту сканирования в передающем CPU таким образом, при котором произведение частоты сканирования и времени цикла в S7-300 был больше 60 мс, а в S7-400 – больше 10 мс. Для принимающего CPU этот результат должен быть меньше, чем для передающего CPU, чтобы избежать потерю каких-либо GD-пакетов.

При частоте сканирования 0 обмен данными соответствующего пакета отключается, если вы хотите только передать или получить его в зависимости от события при помощи SFC.

После конфигурирования GD-состояния и частот сканирования вы должны скомпилировать GD-таблицу во второй раз. Затем STEP 7 введет скомпилированные данные в объект *System data* (*Системные данные*). GD-коммуникация начинает действовать, когда вы передадите GD-таблицу в подключенные CPU с помощью PLC → Download to Module (PLC → Загрузить в модуль).

GD-коммуникация также становится эффективной (действующей), когда объект *System data* (*Системные данные*), содержащий все аппаратные и параметрические установки, передается.

20.5.3 Системные функции для GD-коммуникации

В системах S7-400 вы также можете управлять GD-коммуникацией в вашей программе. Дополнительно или в качестве альтернативы циклической передаче глобальных данных вы можете послать или получить GD-пакет, используя следующие SFC:

- SFC 60 GD_SND
Передача GD-пакета
- SFC 61 GD_RCV
Получение GD-пакета

Параметры для этих SFC приведены в таблице 20.10. Необходимым условием для их применения является сконфигурированная таблица глобальных данных. После компиляции этой таблицы STEP 7 в столбце «GD Identifier» («GD-идентификатор») покажет вам номера GD-циклов и GD-пакетов, которые требуются вам для назначения параметров.

Таблица 20.10 Параметры SFC для GD-коммуникации

Параметр	Имеется в SFC		Объявление	Тип данных	Содержимое, описание
CIRCLE_ID	60	61	INPUT	BYTE	Номер GD-цикла
BLOCK_ID	4	61	INPUT	BYTE	Номер GD-пакета, предназначенного для передачи или получения
RET_VAL	60	61	OUTPUT	INT	Информация об ошибке

SFC 60 GD_SND переносит GD-пакет в системную память CPU и инициирует передачу; SFC 61 GD_RCV считывает GD-пакет из системной памяти. Если частота сканирования, определенная для GD-пакета в GD-таблице, больше нуля, то циклическая передача также имеет место.

Если вы хотите обеспечить консистентность данных для всего GD-пакета при передаче с помощью SFC 60 и 61, то во время обработки SFC 60 или SFC 61 необходимо отменить или задержать прерывания более высокого приоритета и асинхронные ошибки для обеих сторон – передатчика и приемника.

Не требуется SFC вызывать парами; «смешанная» операция также допустима. Например, вы можете использовать SFC 60 GD_SND для организации событийной передачи GD-пакетов, но прием в этом случае циклический.

20.6 SFC-коммуникация

20.6.1 SFC-коммуникация внутри станции

Основы

Внутренняя для станции SFC-коммуникация позволяет вам осуществлять обмен данными между программируемыми модулями в SIMATIC-станции. Коммуникационные функции, требующиеся при этом, являются SFC операционной системы CPU. Если необходимо, эти SFC сами устанавливают коммуникационные соединения. Поэтому эти внутростанционные соединения не конфигурируются посредством таблицы соединений (connection table) («Коммуникация через неконфигурированные соединения», «Communication via non-configured connections», базовая коммуникация).

Внутростанционная SFC-коммуникация может иметь место, к примеру, параллельно циклическому обмену данных через PROFIBUS-DP между главным CPU и ведомым CPU, при этом обмен данными является событийным (рисунок 20.14).

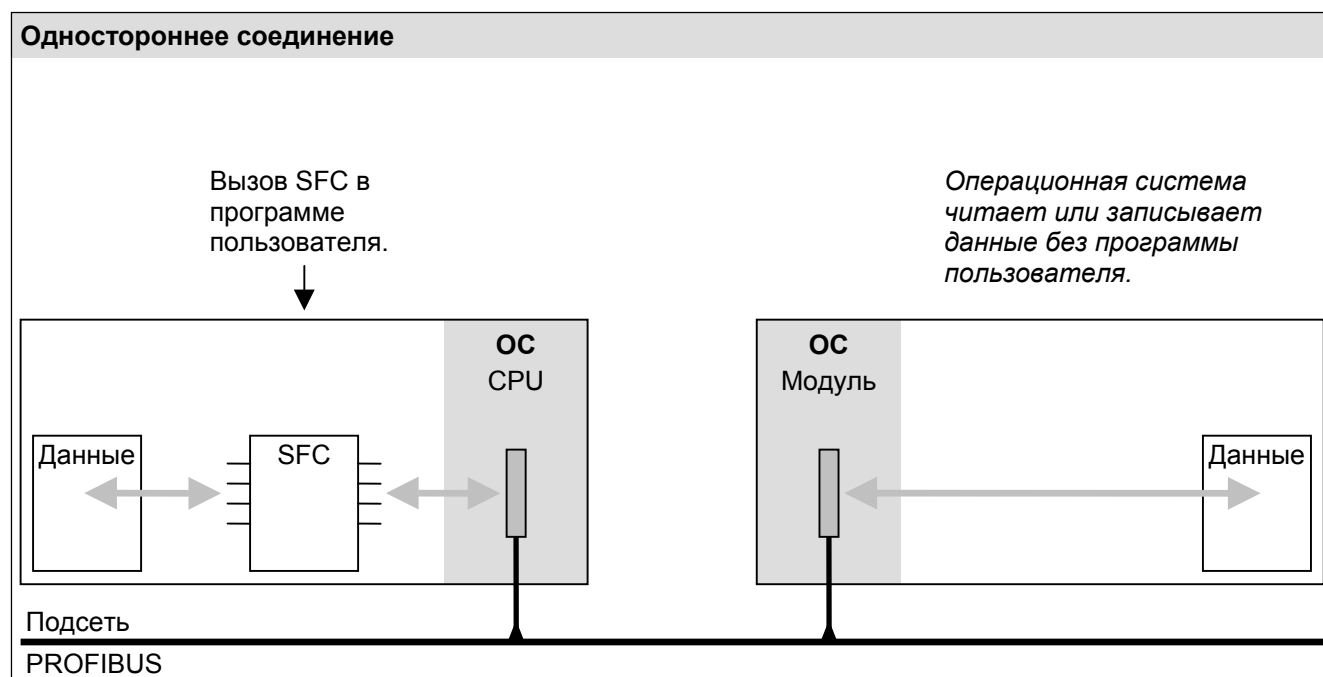


Рисунок 20.14 Внутростанционная SFC-коммуникация

Адресация узлов, соединения

Идентификация узла основывается на адресе входа/выхода (I/O-адресе): в параметре LADDR в должны указать стартовый адрес модуля, а в параметре IOID – где находится этот адрес, в области входов или области выходов.

Рассматриваемые системные функции динамически устанавливают необходимые коммуникационные соединения и снимают их после завершения задания (программируемые). Если построение соединения невозможно выполнить из-за недостатка ресурсов либо в передающем устройстве, либо в принимающем устройстве, то возникает сообщение «Temporary lack of resources» («Временная нехватка ресурсов»). Затем передача должна быть возобновлена. Между двумя коммуникационными партнерами в каждом направлении может быть только одно соединение.

Вы можете использовать одну системную функцию для различных коммуникационных соединений путем изменения параметров во время исполнения. SFC не может прерывать сама себя. Программный раздел, в котором одна из этих SFC используется, может модифицироваться только в режиме CPU STOP; после этого выполняется полный рестарт.

Пользовательские данные, консистентность данных

Эти SFC передают до 76 байтов пользовательских данных. Независимо от направления передачи операционная система CPU компонует пользовательские данные в блоки, внутри являющиеся консистентными. В S7-300 эти блоки имеют размер 8 байтов, в CPU 412/413 они занимают 16 байтов, а в CPU 414/416 – 32 байта. Если два CPU обмениваются данными, то решающее значение для консистентности данных имеет размер блока у «пассивного» CPU.

Конфигурирование внутростанционной SFC-коммуникации

SFC-коммуникация внутри станции является особым случаем, в котором не требуется ее конфигурирование, так как передача данных управляется через динамические соединения. Вы просто используете имеющуюся подсеть PROFIBUS или создаете ее либо в SIMATIC-менеджере, выбрав объект *Project (Проект)* и затем команду меню Insert → Subnetwork → PROFIBUS (Вставка → Подсеть → PROFIBUS), либо в утилите Network Configuration (см. параграф 2.4 «Конфигурирование сети»).

Пример: у вас имеются сконфигурированные распределенные входы/выходы с CPU 315-2DP в качестве мастера (главного CPU). Вы можете использовать другой CPU 315-2DP как «интеллектуальный» ведомый. Теперь можно воспользоваться внутростанционной SFC-коммуникацией для чтения и записи данных в обоих контроллерах.

20.6.2 Системные функции для обмена данными в станции

Следующие системные функции управляют передачей данных между двумя CPU в одной станции:

- SFC 72 I_GET
Чтение данных

- SFC 73 I_PUT
Запись данных
- SFC 74 I_ABORT
Разъединение

Параметры этих SFC показаны в таблице 20.11.

Таблица 20.11 Параметры SFC для коммуникации в станции

Параметр	Для SFC			Объявление	Тип данных	Содержимое, описание
	72	73	74			
REQ	72	73	74	INPUT	BOOL	Начинает работу при REQ = «1»
CONT	72	73	-	INPUT	BOOL	CONT = «1»: соединение сохраняется после завершения задания
IOID	72	73	74	INPUT	BYTE	V#16#54 = область входов V#16#55 = область выходов
LADDR	72	73	74	INPUT	WORD	Стартовый адрес модуля
VAR_ADDR	72	73	-	INPUT	ANY	Область данных в CPU-партнере
SD	-	73	-	INPUT	ANY	Область данных в собственном CPU, содержащем передаваемые данные
RET_VAL	72	73	74	OUTPUT	INT	Информация об ошибке
BUSY	72	73	74	OUTPUT	BOOL	Задание выполняется, когда BUSY = «1»
RD	72	-	-	OUTPUT	ANY	Область данных в собственном CPU, который примет данные (для получения)

SFC 72 I_GET Чтение данных

Задание инициируется при REQ = «1» и BUSY = «0» («первый вызов»). Пока задание обрабатывается, BUSY установлен в «1». Изменения параметра REQ в дальнейшем не учитываются. По завершении задания BUSY сбрасывается в «0». Если REQ = «1» по-прежнему, то задание немедленно перезапускается.

Когда процедура чтения инициирована, CPU-партнер собирает и отправляет запрашиваемые данные. Вызов SFC передает данные (для получения) в область назначения. В этом случае RET_VAL показывает количество переданных байтов.

Если CONT = «0», коммуникационная связь разорвана. Если CONT = «1», связь сохраняется. Чтение данных осуществляется также, когда коммуникационный партнер находится в режиме STOP.

Параметры RD и VAR_ADDR описывают область, из которой данные, предназначенные для передачи, должны быть прочитаны, или в которую записываются принимаемые данные. Фактические параметры могут быть адресами, переменными или

областями данных, адресованными указателем ANY. Передаваемые и принимаемые данные не проверяются на идентичность типов данных.

SFC 73 I_PUT

Запись данных

Задание инициируется при REQ = «1» и BUSY = «0» («первый вызов»). Пока задание выполняется, BUSY установлен в «1». Изменения параметра REQ в дальнейшем игнорируются. Когда задание завершается, BUSY сбрасывается в «0». Если REQ = «1» по-прежнему, то задание немедленно перезапускается.

Когда процедура записи инициирована, операционная система при первом вызове передает все данные из области-источника во внутренний буфер и отправляет их коммуникационному партнеру. Приемник записывает данные в область данных VAR_ADDR. BUSY в этом случае устанавливается в «0». Запись данных производится также, когда принимающий партнер находится в режиме STOP.

Параметры RD и VAR_ADDR описывают область, из которой данные, предназначенные для передачи, должны быть прочитаны, или в которую записываются принимаемые данные. Фактическими параметрами могут быть адреса, переменные или области данных, адресованные указателем ANY. Передаваемые и принимаемые данные не проверяются на идентичность типов данных.

SFC 74 I_ABORT

Разъединение

REQ = «1» разрывает соединение с определенным коммуникационным партнером. Используя I_ABORT, вы можете разъединить только соединения, установленные в той же станции с помощью I_GET и I_PUT.

Пока задание выполняется, BUSY установлен в «1». Изменения параметра REQ в дальнейшем влияния не оказывают. Когда задание завершается, BUSY сбрасывается в «0». Если REQ все еще установлен в «1», то задание немедленно перезапускается.

20.6.3 SFC-коммуникация вне станции

Основы

С использованием SFC-коммуникации вне станции вы можете организовать событийный обмен данными между станциями SIMATIC S7. Станции должны быть соединены между собой через подсеть MPI. Коммуникационные функции, требующиеся для этого, относятся к системным функциям (SFC) операционной системы CPU. При необходимости эти SFC сами устанавливают коммуникационные соединения. По этой причине эти внешние по отношению к станции соединения не конфигурируются через таблицу соединений («Коммуникация через неконфигуриро-

ванные соединения», «Communication via non-configured connections», базовая коммуникация).

Внешняя по отношению к станции SFC-коммуникация может выполнять событийную передачу данных, к примеру, параллельно с циклической передачей глобальных данных.

Адресация узлов, соединения

Эти функции адресуют узлы, которые находятся в одной подсети MPI. Идентификация узлов основывается на MPI-адресе (параметр `DEST_ID`).

Данные системные функции динамически устанавливают требуемые коммуникационные связи и, если задано, разрывают их, когда задание выполнено. Если соединение не может быть установлено по причине недостатка ресурсов либо в передатчике, либо в приемнике, то сообщается о «временной нехватке ресурсов» («Temporary lack of resources»). Должна быть осуществлена повторная передача. Между двумя коммуникационными партнерами может быть только одно соединение в каждом направлении.

При переходе из режима CPU RUN в состояние STOP все активные соединения (все SFC, кроме `X_RECV`) удаляются.

Изменяя параметры блоков во время исполнения, вы можете использовать одну системную функцию для различных коммуникационных связей. SFC не может прерывать сама себя. Вы можете модифицировать программный раздел, в котором используется одна из этих SFC, только в режиме CPU STOP; после этого должен быть выполнен полный рестарт.

Пользовательские данные, консистентность данных

Рассматриваемые SFC могут передавать максимум 76 байтов пользовательских данных. Операционная система комбинирует данные пользователя в блоки, консистентные внутри, независимо от направления передачи. В системах S7-300 эти блоки имеют размер 8 байтов, в системах с CPU 412/413 длина их достигает 16 байтов, а в системах с CPU 414/416 их размер составляет 32 байта.

Если два CPU обмениваются данными через `X_GET` или `X_PUT`, то решающее значение для консистентности передаваемых данных имеет размер блока «пассивного» CPU. В случае соединения SEND/RECEIVE все данные вызова являются консистентными.

Конфигурирование SFC-коммуникации вне станции

Внешняя по отношению к станции SFC-коммуникация является особым случаем, при котором не требуется ее конфигурировать, так как передача данных управляется

посредством динамических соединений. Вы можете просто использовать существующую подсеть PROFIBUS или создать ее.

Пример: у вас может быть разделенная монтажная стойка (CR2) S7-400 с одним CPU 416 в каждой секции. Кроме этого, станция S7-300 с CPU 314 через кабель MPI соединена с одной из станций S7-400. Вы можете сконфигурировать все три CPU в утилите Hardware Configuration, например, как «объединенные в сеть» через подсеть MPI. Теперь вы можете воспользоваться внешней по отношению к станции SFC-коммуникацией всех трех контроллеров, чтобы осуществить обмен данными.

20.6.4 Системные функции для SFC-коммуникации вне станции

Передачей данных между партнерами в различных станциях управляются следующие системные функции:

- SFC 65 X_SEND
Отправка данных
- SFC 66 X_RCV
Прием данных
- SFC 67 X_GET
Чтение данных
- SFC 68 X_PUT
Запись данных
- SFC 69 X_ABORT
Разъединение

Параметры для SFC приведены в таблице 20.12.

SFC 65 X_SEND

Отправка данных

Задание инициируется при REQ = «1» и BUSY = «0» («первый вызов»). Пока задание выполняется, BUSY установлен в «1»; изменения параметра REQ не учитываются. Когда задание завершено, BUSY устанавливается обратно в «0». Если по-прежнему REQ = «1», то задание немедленно перезапускается.

При первом вызове операционная система передает все данные из области источника во внутренний буфер, затем пересылает данные в партнер-CPU.

При выполнении процедуры отправки BUSY установлен в «1». Если партнер про-сигнализировал о получении данных, BUSY переходит обратно в «0», и задание от-правки завершается.

Если CONT = «0», то соединение разрывается, и соответствующие ресурсы CPU становятся доступными для других коммуникационных связей. Если CONT = «1», соединение сохраняется. Параметр REQ_ID позволяет вам назначить ID (идентификатор) отправляемым данным, который вы можете получить при помощи SFC X_RCV.

Параметр SD описывает область, из которой должны быть прочитаны данные, предназначенные для отправки. Фактическими параметрами могут быть адреса, переменные или области данных, адресованные указателем ANY. Отправляемые и получаемые данные не проверяются на соответствие типам данных.

Таблица 20.12 Параметры SFC для коммуникаций вне станции

Параметр	Для SFC					Объявление	Тип данных	Содержимое, описание
	65	-	67	68	69			
REQ	65	-	67	68	69	INPUT	BOOL	Инициация задания при REQ = «1»
CONT	65	-	67	68	-	INPUT	BOOL	CONT = «1»: соединение сохраняется после завершения задания
DEST_ID	65	-	67	68	69	INPUT	WORD	Идентификация узла партнера (MPI-адрес)
REQ_ID	65	-	-	-	-	INPUT	DWORD	Идентификация задания
VAR_ADDR	-	-	67	68	-	INPUT	ANY	Область данных в CPU-партнере
SD	65	-	-	68	-	INPUT	ANY	Область данных в собственном CPU, которая содержит отправляемые данные
EN_DT	-	66	-	-	-	INPUT	BOOL	Если «1»: принимает данные (для получения)
RET_VAL	65	66	67	68	69	OUTPUT	INT	Информация об ошибке
BUSY	65	-	67	68	69	OUTPUT	BOOL	Задание выполняется, когда BUSY = «1»
REQ_ID	-	66	-	-	-	OUTPUT	DWORD	Идентификация задания
NDA	-	66	-	-	-	OUTPUT	BOOL	Когда «1»: данные приняты
RD	-	66	67	-	-	OUTPUT	ANY	Область данных в собственном CPU, которая примет данные (для получения)

SFC 66 X_RCV

Прием данных

Принимаемые данные помещены во внутренний буфер. Несколько пакетов могут быть помещены в очередь в хронологическом порядке их поступления.

Можно использовать EN_DT = «0» чтобы проверить, данные были получены или нет; если это так (данные получены), то NDA устанавливается в «1», RET_VAL показывает число байтов принятых данных, а REQ_ID имеет то же значение, как и соответствующий параметр в SFC 65 X_SEND. При EN_DT = «1» SFC передает первый (старший) пакет в область назначения; NDA в таком случае равен «1», а RET_VAL отображает количество переданных байтов. Если EN_DT установлен в «1», но во внутренней очереди данных нет, то NDA равен «0». В случае полного рестарта все пакеты данных в очереди отбрасываются.

При разрыве соединения или рестарте старший записанный элемент в очереди, если уже «запрошен» с EN_DT = «0», сохраняется; иначе он теряется, как и другие элементы очереди.

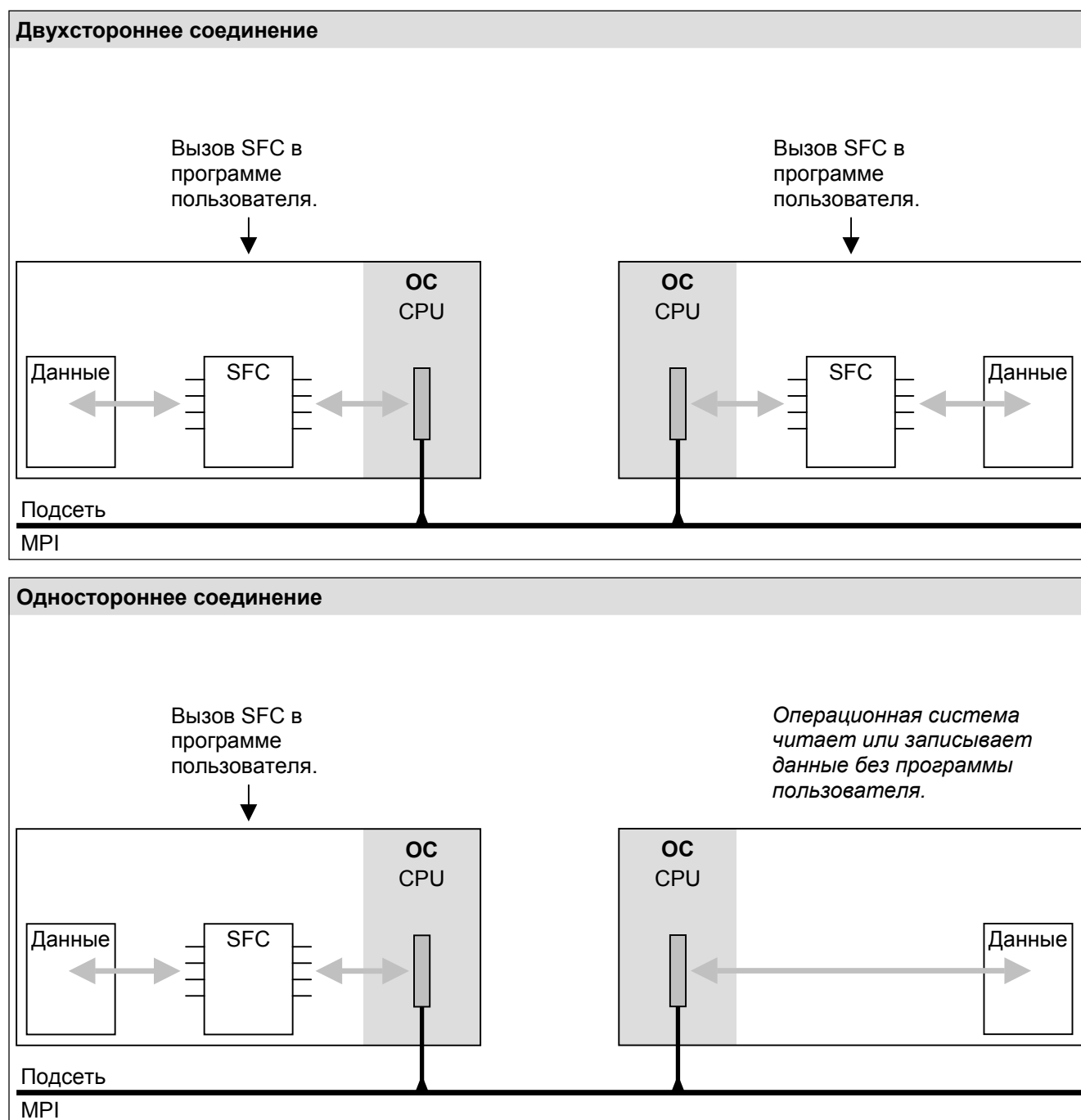


Рисунок 20.15 Внешняя по отношению к станции SFC-коммуникация

Параметр RD описывает область, в которую получаемые данные должны быть записаны. Формальными параметрами могут быть адреса, переменные или области данных, адресованные указателем ANY.

Передаваемые и получаемые данные не проверяются на соответствие типов данных. Когда получаемые данные нерелевантные, то в качестве параметра RD в X_RCV допускается «пустой» указатель ANY (указатель NIL).

SFC 67 X_GET

Чтение данных

Задание инициируется при REQ = «1» и BUSY = «0» («первый вызов»). Пока задание выполняется, BUSY установлен в «1»; изменения параметра REQ в дальнейшем игнорируются.

Когда задание завершается, BUSY устанавливается обратно в «0». Если REQ все еще равен «1», задание немедленно перезапускается.

Когда процедура чтения инициирована, операционная система в CPU-партнере собирает и отправляет требуемые данные, используя VAR_ADDR. При вызове SFC принимаемые данные вводятся в область назначения, определенную в параметре RD. RET_VAL в этом случае показывает число переданных байтов.

Если CONT = «0», коммуникационная связь разрывается. Если CONT = «1», то соединение сохраняется. Данные при этом считываются, даже если коммуникационный партнер находится в режиме STOP.

Параметры RD и VAR_ADDR описывают область, из которой считываются данные, предназначенные для пересылки, или в которую принимаемые данные должны быть записаны. Формальными параметрами могут быть адреса, переменные или адресованные указателем ANY области данных. Отправляемые и принимаемые данные не проверяются на соответствие типов данных.

SFC 68 X_PUT

Запись данных

Задание инициируется при REQ = «1» и BUSY = «0» («первый вызов»). Пока задание выполняется, BUSY равен «1»; изменения параметра REQ в дальнейшем не учитываются.

Когда задание завершается, BUSY сбрасывается обратно в «0». Если REQ все еще равен «1», задание немедленно перезапускается.

Когда процедура записи инициирована, операционная система пересылает все данные из области-источника, определенной в параметре SD, во внутренний буфер при первом вызове, затем отправляет данные CPU-партнеру. Теперь операционная система CPU-партнера записывает принимаемые данные в область данных, определенную в параметре VAR_ADDR. BUSY устанавливается в этом случае в «0».

Параметры RD и VAR_ADDR описывают область, из которой считываются данные, предназначенные для пересылки, или в которую принимаемые данные должны быть записаны. Формальными параметрами могут быть адреса, переменные или адресо-

ванные указателем ANY области данных. Отправляемые и принимаемые данные не проверяются на соответствие типов данных.

SFC 69 X_ABORT

Разъединение

REQ = «1» разрывает существующее соединение с определенным коммуникационным партнером. SFC X_ABORT может быть использована для снятия соединений, установленных в собственной станции CPU с помощью SFC X_SEND, X_GET или X_PUT.

20.7 SFB-коммуникация

20.7.1 Основы

Используя SFB-коммуникацию, вы можете передавать большие объемы данных между станциями SIMATIC S7. Станции соединены друг с другом посредством подсети; это может быть подсеть MPI, подсеть PROFIBUS или подсеть Ethernet. Коммуникационные соединения статические; они конфигурируются в таблице соединений («Коммуникация через сконфигурированные соединения», «Communication via configured connections», расширенная коммуникация).

Коммуникационные функции выполняются системными функциональными блоками SFB, встроенными в операционную систему CPU S7-400. Связанные с ними экземпляры блоков данных располагаются в пользовательской памяти (user memory). Если вы хотите воспользоваться SFB-коммуникацией, скопируйте описание интерфейса SFB из *Standard Library* (Стандартной библиотеки), раздела *System Function Blocks* (Системные функциональные блоки) в контейнер *Blocks* (Блоки), сгенерируйте для каждого вызова экземплярный блок данных и вызовите SFB с соответствующим экземпляром блока данных. При пошаговом вводе вы также можете выбрать SFB из каталога программных элементов (Program Element Catalog), блок данных в таком случае сгенерируется автоматически.

Конфигурирование SFB-коммуникации

Обязательным условием для коммуникации через системные функциональные блоки является сконфигурированная таблица соединений с определенными в ней коммуникационными связями.

Коммуникационная связь определяется идентификатором (ID) соединения для каждого коммуникационного партнера. STEP 7 назначает идентификаторы соединений при компиляции таблицы соединений. Для инициализации SFB в локальном или «собственном» модуле используйте «локальный ID», а для инициализации SFB в модуле-партнере – «удаленный ID».

Одно и то же логическое соединение может быть использовано для различных запросов отправки/получения. Чтобы их различать, вы должны добавить ID задания к ID соединения, и тем самым определить отношения между блоком отправления и блоком приема.

Инициализация

SFB-коммуникация должна быть инициализирована на рестарте, с тем чтобы соединение с коммуникационным партнером могло быть установлено. Инициализация происходит в CPU, который в таблице соединений получает атрибут «Active connection buildup = Yes» («Построение активного соединения = Да»). Вызываются коммуникационные SFB, используемые в циклической операции, в ОВ рестарта, а параметры инициализируются (обеспечивая их доступность) в следующем виде:

- REQ = FALSE
- ID = ID локального соединения из таблицы соединений (тип данных WORD W#16#xxxx)
- PI_NAME = переменная, содержащая 'P_PROGRAM' в ASCII-коде (например, ARRAY[1..9] OF CHAR).

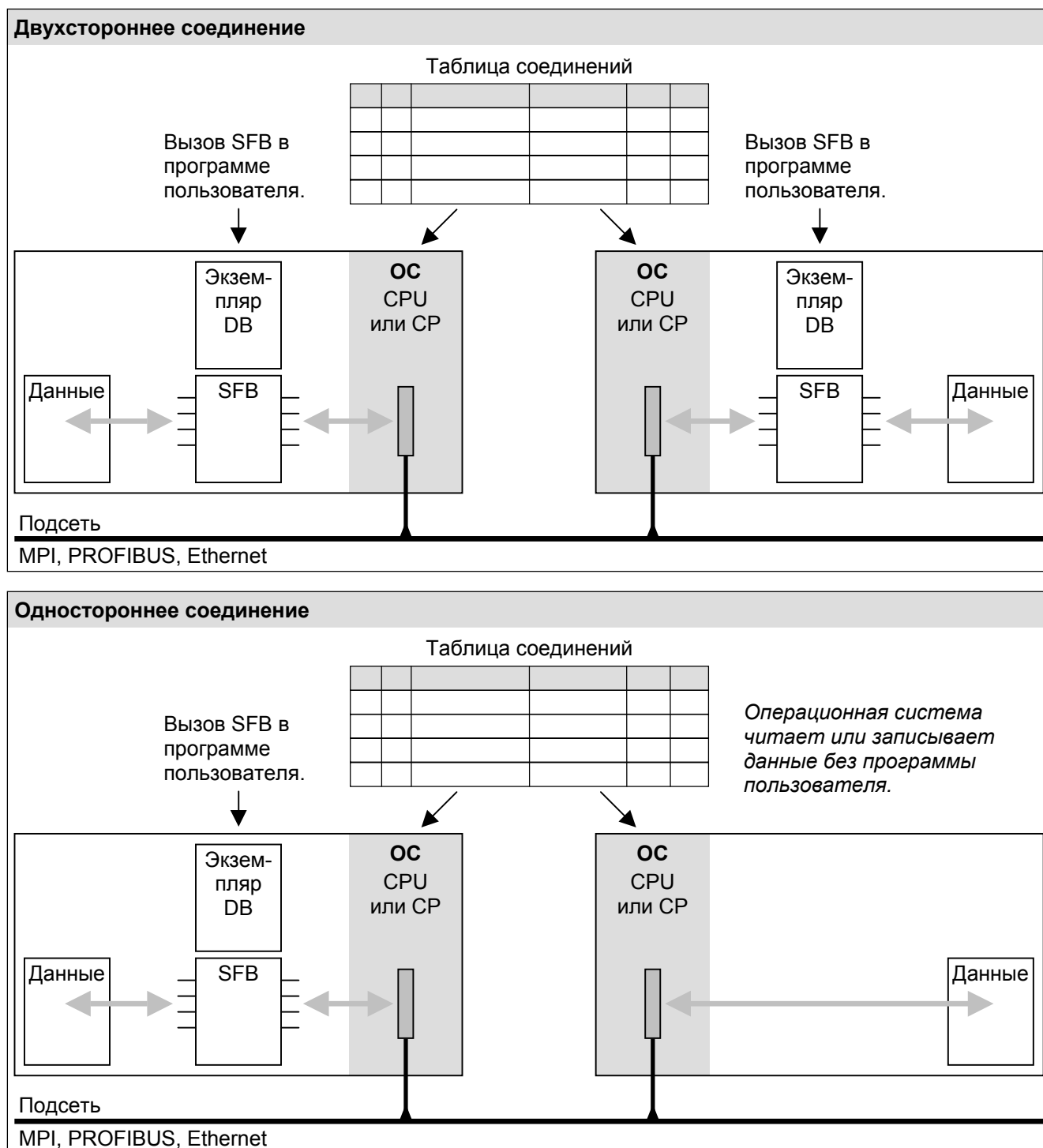


Рисунок 20.16 SFB-коммуникация

Вызовы SFB в программном цикле должны продолжаться, пока параметр DONE не приобретет сигнальное состояние «1». Параметры ERROR и STATUS предоставляют информацию о возникших ошибках и состоянии задания. Вам не нужно при рестарте переключать области данных (касается параметров ADDR_x, RD_x и SD_x).

В циклической операции коммуникационные SFB вызываются абсолютно, и передачей данных вы можете управлять через параметры REQ и EN_R.

20.7.2 Двухсторонний обмен данными

Для двухстороннего обмена данными вам потребуются один блок отправки (SEND) и один блок приема (RECEIVE) на концах соединений. Оба блока имеют идентификаторы соединения, расположенные в одной строке таблицы соединений. Вы так же можете использовать несколько «пар блоков», которые в этом случае различаются между собой идентификатором задания.

Для двухстороннего обмена данными доступны следующие SFB:

- SFB 8 USEND
Некоординированное отправление пакета данных размером, определяемым CPU;
- SFB 9 URCV
Некоординированное получение пакета данных размером, определяемым CPU;
- SFB 12 BSEND
Отправление блока данных размером до 64 Кб;
- SFB 13 BRCV
Получение блока данных размером до 64 Кб.

SFB 8 и SFB 9 или SFB 12 и SFB 13 должны всегда использоваться парами.

Параметры этих SFB приведены в таблице 20.13.

SFB 8 USEND и SFB 9 URCV **Некоординированные отправление и получение**

Параметры SD_x и RD_x используются для определения переменной или области, которую вы хотите передать. Отсылаемая область SD_x должна соответствовать соответствующей принимаемой области RD_x. Параметры используйте без промежутков, начиная с 1. Значения для ненужных параметров определять не требуется (как и у FB, значения могут быть присвоены не всем SFB-параметрам).

При первом вызове SFB 9 генерируется принимающий почтовый ящик; во все последующие вызовы получаемый элемент должен точно соответствовать этому почтовому ящику.

Таблица 20.13 Параметры SFB для отправления и получения данных

Параметр	Для SFB				Объявление	Тип данных	Содержимое, описание
	8	9	12	13			
REQ	8	-	12	-	INPUT	BOOL	Запуск обмена данными
EN_R	-	9	-	13	INPUT	BOOL	Прием готов
R	-	-	12	-	INPUT	BOOL	Останов обмена данными
ID	8	9	12	13	INPUT	WORD	ID соединения
R_ID	8	9	12	13	INPUT	DWORD	ID задания
DONE	8	-	12	-	OUTPUT	BOOL	Задание завершено
NDR	-	9	-	13	OUTPUT	BOOL	Считаны новые данные
ERROR	8	9	12	13	OUTPUT	BOOL	Возникла ошибка
STATUS	8	9	12	13	OUTPUT	WORD	Состояние задания
SD_1	8	-	12	-	IN_OUT	ANY	Первая передаваемая область
SD_2	8	-	-	-	IN_OUT	ANY	Вторая передаваемая область
SD_3	8	-	-	-	IN_OUT	ANY	Третья передаваемая область
SD_4	8	-	-	-	IN_OUT	ANY	Четвертая передаваемая область
RD_1	-	9	-	13	IN_OUT	ANY	Первая область для приема
RD_2	-	9	-	-	IN_OUT	ANY	Вторая область для приема
RD_3	-	9	-	-	IN_OUT	ANY	Третья область для приема
RD_4	-	9	-	-	IN_OUT	ANY	Четвертая область для приема
LEN	-	-	12	13	IN_OUT	WORD	Размер блока данных в байтах

Положительный фронт в параметре REQ (request, запрос) запускает обмен данными, положительный фронт в параметре R (reset, сброс) останавливает его. «1» в параметре EN_R (enable receive, разрешение приема) сигнализирует о готовности партнера принять данные.

Инициализируйте параметр ID, используя ID соединения, который STEP 7 вводит в таблицу соединений для обоих, локального и партнера (эти два идентификатора могут быть различными). R_DI позволяет вам выбрать определяемый, но уникальный ID задания, который должен совпадать для блоков отправки и приема. Это дает возможность нескольким парам блоков отправки и приема поделить единственное логическое соединение (так как каждый имеет уникальный ID).

Блок при первом вызове передает фактические значения параметров ID и R_ID своему экземпляру блока данных. Первый вызов устанавливает коммуникационное отношение (для этого экземпляра) до следующего полного рестарта.

Используя сигнальное состояние «1» параметров DONE и NDR, блок сообщает, что задание завершено без ошибок. Ошибка, если она возникает, отмечается в параметре ERROR. Значение параметра STATUS, отличное от нуля, индицирует либо предупреждение (ERROR = «0»), либо ошибку (ERROR = «1»). Вы должны оценивать параметры DONE, NDR, ERROR и STATUS после *каждого* вызова блока.

SFB 12 BSEND и SFB 13 BRCV**Ориентированные на блоки отправление и получение**

Вы можете ввести указатель на первый байт области данных в параметры SD_1 или RD_1 (размер не вычисляется); число байтов отправляемых и получаемых данных указывается в параметре LEN.

Может быть передано до 64 Кб; данные передаются в блоках (иногда называемых фреймами), сама передача асинхронна по отношению к сканированию программы пользователя.

Положительный фронт в параметре REQ (request, запрос) запускает обмен данными, положительный фронт в параметре R (reset, сброс) останавливает его. «1» в параметре EN_R (enable receive, разрешение приема) сигнализирует о готовности партнера принять данные. Инициализируйте параметр ID, используя ID соединения, который STEP 7 вводит в таблицу соединений для обоих, локального и партнера (эти два идентификатора могут быть различными).

R_DI позволяет вам выбрать определяемый, но уникальный ID задания, который должен совпадать для блоков отправки и приема. Это дает возможность нескольким парам блоков отправки и приема поделить единственное логическое соединение (так как каждый имеет уникальный ID).

При первом вызове блок передает фактические значения параметров ID и R_ID своему экземпляру блока данных. Первый вызов устанавливает коммуникационное отношение (для этого экземпляра) до следующего полного рестарта.

Используя сигнальное состояние «1» параметров DONE и NDR, блок сообщает, что задание завершено без ошибок. Ошибка, если она возникает, отмечается в параметре ERROR. Значение параметра STATUS, отличное от нуля, индицирует либо предупреждение (ERROR = «0»), либо ошибку (ERROR = «1»). Вы должны оценивать параметры DONE, NDR, ERROR и STATUS после *каждого* вызова блока.

20.7.3 Односторонний обмен данными

В случае одностороннего обмена данными вызов коммуникационного SFB расположен только в одном CPU. В CPU-партнере операционная система управляет необходимыми коммуникационными функциями.

Для одностороннего обмена данными доступны следующие SFB:

- SFB 14 GET
Чтение данных, максимальный размер которых определяется CPU;
- SFB 15 PUT
Запись данных, максимальный размер которых определяется CPU.

Таблица 20.14 содержит список параметров для этих SFB.

Таблица 20.14 Параметры SFB для чтения и записи данных

Параметр	Для SFB		Объявление	Тип данных	Содержимое, описание
REQ	14	15	INPUT	BOOL	Запуск обмена данными
ID	14	15	INPUT	WORD	ID соединения
NDR	14	-	OUTPUT	BOOL	Считаны новые данные
DONE	-	15	OUTPUT	BOOL	Задание завершено
ERROR	14	15	OUTPUT	BOOL	Возникла ошибка
STATUS	14	15	OUTPUT	WORD	Состояние задания
ADDR_1	14	15	IN_OUT	ANY	Первая область данных в CPU-партнере
ADDR_2	14	15	IN_OUT	ANY	Вторая область данных в CPU-партнере
ADDR_3	14	15	IN_OUT	ANY	Третья область данных в CPU-партнере
ADDR_4	14	15	IN_OUT	ANY	Четвертая область данных в CPU-партнере
RD_1	14	-	IN_OUT	ANY	Первая область для приема
RD_2	14	-	IN_OUT	ANY	Вторая область для приема
RD_3	14	-	IN_OUT	ANY	Третья область для приема
RD_4	14	-	IN_OUT	ANY	Четвертая область для приема
SD_1	-	15	IN_OUT	ANY	Первая передаваемая область
SD_2	-	15	IN_OUT	ANY	Вторая передаваемая область
SD_3	-	15	IN_OUT	ANY	Третья передаваемая область
SD_4	-	15	IN_OUT	ANY	Четвертая передаваемая область

Операционная система CPU-партнера собирает данные, прочитанные с помощью SFB 14; операционная система CPU-партнера распределяет данные, записанные при помощи SFB 15. Отправляющая или принимающая (пользовательская) программа в CPU-партнере не требуется.

Положительный фронт в параметре REQ (request, запрос) запускает обмен данными. Установите параметр ID для идентификатора (ID) соединения, введенного STEP 7 в таблице соединений.

С помощью значения «1» параметров DONE или NDR блок сигнализирует о завершении задания без возникновения ошибок. Ошибка, если она возникла, отмечается значением «1» в параметре ERROR.

Значение параметра STATUS, отличное от нуля, индицирует либо предупреждение (ERROR = «0»), либо ошибку (ERROR = «1»). Вы должны оценивать параметры DONE, NDR, ERROR и STATUS после *каждого* вызова блока.

Параметр ADDR_n используется для определения переменной или области в CPU-партнере, из которой вы хотите считать или в которую хотите послать данные. Области в ADDR_n должны совпадать с областями, определенными в SD_n или RD_n. Параметры используйте без промежутков, начиная с 1. Ненужные параметры определять не требуется (как в FB, SFB может не иметь значений для всех параметров).

20.7.4 Передача данных для печати

SFB 16 PRINT позволяет вам передавать описание формата на принтер через коммуникационный процессор CP 441. В таблице 20.15 показаны параметры для этого SFB.

Таблица 20.15 Параметры SFB 16 PRINT

Параметр	Объявление	Тип данных	Содержимое, описание
REQ	INPUT	BOOL	Запуск обмена данными
ID	INPUT	WORD	ID соединения
DONE	OUTPUT	BOOL	Задание завершено
ERROR	OUTPUT	BOOL	Обнаружена ошибка
STATUS	OUTPUT	WORD	Состояние задания
PRN_NR	IN_OUT	BYTE	Номер принтера
FORMAT	IN_OUT	STRING	Описание формата
SD_1	IN_OUT	ANY	Первая переменная
SD_2	IN_OUT	ANY	Вторая переменная
SD_3	IN_OUT	ANY	Третья переменная
SD_4	IN_OUT	ANY	Четвертая переменная

Положительный фронт в параметре REQ запускает обмен данными с принтером, заданным параметрами ID и PRN_NR. Блок сигнализирует о безошибочной передаче установкой DONE в «1». Любая возникающая ошибка отмечается значением «1» в параметре ERROR. Отличное от нуля значение параметра STATUS показывает либо предупреждение (ERROR = «0»), либо ошибку (ERROR = «1»). Вы должны оценивать параметры DONE, ERROR и STATUS после *каждого* вызова блока.

Литеры для вывода на печать вводите в параметр FORMAT в формате STRING. Вы можете в эту строку включить до четырех описаний форматов для переменных, определенных в параметрах SD_1, ..., SD_4. Параметры используйте без промежутков, начиная с 1; не определяйте значения для ненужных параметров. Можно передать до 420 байтов (сумма FORMAT и всех переменных) на запрос принтера.

20.7.5 Функции управления

Для управления коммуникационным партнером доступны следующие SFB:

- SFB 19 START
Выполняет полный рестарт контроллера-партнера;
- SFB 20 STOP
Переключает контроллер-партнер в режим STOP;

- **SFB 21 RESUME**
Выполняет «теплый» рестарт контроллера-партнера.

Эти SFB предназначены для одностороннего обмена данными; для этой цели в устройстве-партнере не требуется пользовательской программы. Параметры указанных SFB показаны в таблице 20.16.

Таблица 20.16 Параметры SFB для управления контроллером-партнером

Параметр	Для SFC			Объявление	Тип данных	Содержимое, описание
	19	20	21			
REQ	19	20	21	INPUT	BOOL	Запуск обмена данными
ID	19	20	21	INPUT	WORD	ID соединения
DONE	19	20	21	OUTPUT	BOOL	Задание завершено
ERROR	19	20	21	OUTPUT	BOOL	Обнаружена ошибка
STATUS	19	20	21	OUTPUT	WORD	Состояние задания
PI_NAME	19	20	21	IN_OUT	ANY	Имя программы (P_PROGRAM)
ARG	19	-	21	IN_OUT	ANY	Несущественен
IO_STATE	19	20	21	IN_OUT	BYTE	Несущественен

Положительный фронт в параметре REQ запускает обмен данными. Ведите в качестве параметра ID идентификатор (ID) соединения, который STEP 7 ввел в таблицу соединений.

С помощью значения «1» параметра DONE блок сигнализирует о том, что задание завершено без ошибок. Ошибка, если она возникла, отмечается значением «1» в параметре ERROR. Значение параметра STATUS, отличное от нуля, отображает либо предупреждение (ERROR = «0»), либо ошибку (ERROR = «1»). Вы должны оценивать параметры DONE, ERROR и STATUS после *каждого* вызова блока.

В качестве PI_NAME укажите массив, содержащий «P_PROGRAM» (ARRAY [1..9] OF CHAR). Параметры ARG и IO_STATE в настоящий момент несущественны, и значение им присваивать не требуется.

SFB 19 START выполняет полный рестарт CPU-партнера. Необходимые условия: CPU-партнер должен быть в режиме STOP, переключатель режимов находится в положении RUN или RUN-P.

SFB 20 STOP переводит CPU-партнер в режим STOP. Необходимое условие безошибочного выполнения запроса на это задание: CPU-партнер не должен находиться в режиме STOP при возникновении запроса.

SFB 21 RESUME выполняет «теплый» рестарт CPU-партнера. Необходимые условия: режим STOP CPU-партнера, переключатель режимов находится в положении RUN или RUN-P, «теплый» рестарт допустим в данный момент.

20.7.6 Функции наблюдения

Следующие системные блоки доступны для функций наблюдения:

- SFB 22 STATUS
Проверка состояния партнера;
- SFB 23 USTATUS
Получение состояния партнера;
- SFC 62 CONTROL
Проверка состояния экземпляра SFB.

Таблица 20.17 содержит список параметров для SFB, параметры для SFC 62 показаны в таблице 20.18.

Для этих системных блоков применимо следующее: ошибка отображается значением «1» параметра ERROR. Если параметр STATUS имеет значение, не равное нулю, то оно индицирует либо предупреждение (ERROR = «0»), либо ошибку (ERROR = «1»).

Таблица 20.17 Параметры SFB для опроса состояния

Параметр	Для SFB		Объявление	Тип данных	Содержимое, описание
REQ	22	-	INPUT	BOOL	Запуск обмена данными
EN_R	-	23	INPUT	BOOL	Готов к приему
ID	22	23	INPUT	WORD	ID соединения
NDR	22	23	OUTPUT	BOOL	Считаны новые данные
ERROR	22	23	OUTPUT	BOOL	Возникла ошибка
STATUS	22	23	OUTPUT	WORD	Состояние задания
PHYS	22	23	IN_OUT	ANY	Физическое состояние
LOG	22	23	IN_OUT	ANY	Логическое состояние
LOCAL	22	23	IN_OUT	ANY	Состояние CPU S7 как партнера

SFB 22 STATUS

Проверка состояния устройства-партнера

SFB 22 STATUS считывает состояние CPU-партнера и отображает его в параметрах PHYS (физическое состояние), LOG (логическое состояние) и LOCAL (рабочее состояние, если партнером является CPU S7).

Положительный фронт в параметре REQ (request, запрос) запускает запрос. Введите в качестве параметра ID идентификатор (ID) соединения, который STEP 7 ввел в таблицу соединений.

Используя значение «1» параметра NDR, блок сигнализирует о завершении задания и отсутствии ошибок. Вы должны оценивать параметры NDR, ERROR и STATUS после *каждого* вызова блока.

SFB 23 USTATUS

Получение состояния устройства-партнера

SFB 23 USTATUS поучает состояние партнера, которое он посылает без запроса в случае изменения. Состояние устройства отображается в параметрах PHYS, LOG и LOCAL.

«1» в параметре EN_R (enable receive, разрешение получения) сигнализирует о готовности партнера принимать данные. Инициализируйте параметр ID, используя ID соединения, который STEP 7 ввел в таблицу соединений.

С помощью значения «1» параметра NDR блок сообщает об окончании задания и отсутствии ошибок. Вы должны оценивать параметры NDR, ERROR и STATUS после *каждого* вызова блока.

Таблица 20.18 Параметры для SFC 62 CONTROL

Параметр	Объявление	Тип данных	Содержимое, описание
EN_R	INPUT	BOOL	Запуск обмена данными
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_TYP	OUTPUT	BYTE	Идентификатор типа блока
I_STATE	OUTPUT	BYTE	Идентификатор текущего состояния
I_CONN	OUTPUT	BOOL	Состояние соединения («1» = соединение есть)
I_STATUS	OUTPUT	WORD	Параметр STATUS для экземпляра SFB

SFC 62 CONTROL

Опрос состояния экземпляра SFB

SFC 62 CONTROL определяет состояние экземпляра SFB и соответствующего соединения в локальном контроллере. Передайте экземплярный блок данных SFB в параметр I_DB. Если SFB вызывается как локальный экземпляр, укажите номер локального экземпляра в параметре OFFSET (ноль, когда локального экземпляра нет, 1 для первого локального экземпляра, 2 для второго и так далее).

«1» в параметре EN_R (enable receive, допущение приема) сообщает о готовности партнера принимать данные. Инициализируйте параметр ID, используя ID соединения, который STEP 7 вводит в таблицу соединений.

С помощью значения «1» в параметре NDR блок сигнализирует о завершении задания при отсутствии ошибок. Вы должны оценивать параметры NDR, ERROR и STATUS после *каждого* вызова блока.

Параметры I_TYP, I_STATE, I_CONN и I_STATUS предоставляют информацию о состоянии локального экземпляра SFB.

Содержание главы 21

21	<u>Обработка прерываний</u>	4
21.1	<u>Общие замечания</u>	4
21.2	<u>Аппаратные прерывания</u>	7
21.2.1	<u>Генерирование аппаратного прерывания</u>	7
21.2.2	<u>Обслуживание аппаратных прерываний</u>	8
21.2.3	<u>Конфигурирование аппаратных прерываний с помощью STEP 7</u>	9
21.3	<u>Циклические прерывания</u>	10
21.3.1	<u>Обработка циклических прерываний</u>	10
21.3.2	<u>Конфигурирование циклических прерываний с помощью STEP 7</u>	12
21.4	<u>Прерывания по времени суток</u>	13
21.4.1	<u>Обработка прерываний по времени суток</u>	13
21.4.2	<u>Конфигурирование прерываний по времени суток с помощью STEP 7</u>	15
21.4.3	<u>Системные функции для прерываний по времени суток</u>	15
21.5	<u>Прерывания задержки времени</u>	18
21.5.1	<u>Обработка прерываний задержки времени</u>	18
21.5.2	<u>Конфигурирование прерываний задержки времени с помощью STEP 7</u>	19
21.5.3	<u>Системные функции для прерываний задержки времени</u>	19
21.6	<u>Мультипроцессорное прерывание</u>	22
21.7	<u>Обработка прерываний</u>	24

21 Обработка прерываний

Обработка прерываний всегда находится в зависимости от событий. При возникновении события операционная система прерывает сканирование главной программы и вызывает процедуру, назначенную этому конкретному событию. Когда эта процедура выполнена, операционная система продолжает сканирование главной программы с точки прерывания. Подобное прерывание может иметь место после каждой операции (оператора).

Возникающие события могут быть прерываниями и ошибками. Порядок, в котором виртуально одновременные события прерываний обрабатываются, регламентируется планировщиком приоритетов. Каждое событие имеет определенный приоритет обслуживания. Несколько событий прерываний могут объединяться в приоритетные классы.

Любая процедура, назначенная с событием прерывания, пишется в организационном блоке, в котором могут быть вызваны дополнительные блоки. Событие с более высоким приоритетом прерывает выполнение процедуры в организационном блоке с низким приоритетом. Вы можете вызвать прерывание программы событиями с высоким приоритетом, используя системные функции.

21.1 Общие замечания

В SIMATIC S7 действуют следующие события прерываний (прерывания):

- **Аппаратное прерывание (Hardware interrupt)**
Прерывание от модуля либо через вход, произошедший из сигнала процесса, либо сгенерированное в самом модуле;
- **Циклическое прерывание (Watchdog interrupt)**
Прерывание, периодически генерируемое операционной системой;
- **Прерывание по времени суток (Time-of-day interrupt)**
Прерывание, генерируемое операционной системой в определенное время суток либо один раз, либо периодически;
- **Прерывание задержки времени (Time-delay interrupt)**
Прерывание, генерируемое через определенный промежуток времени; вызов системной функции (SFC) определяет момент, в который этот период времени начинается;
- **Мультипроцессорное прерывание (Multiprocessor interrupt)**
Прерывание, генерируемое другим CPU в мультипроцессорной сети.

Другие события прерываний – это синхронные ошибки, которые могут возникнуть при сканировании программы, и асинхронные ошибки, такие как диагностические прерывания. Обработка этих прерываний обсуждается в главе 23 «Обработка ошибок».

Приоритеты

Событие с более высоким приоритетом прерывает обрабатываемую программу с более низким приоритетом другого события. Главная программа (main program) имеет низший приоритет (приоритетный класс 1), асинхронные ошибки – высший (приоритетный класс 26), не считая процедуру запуска. Остальные события находятся в промежуточных приоритетных классах. В системах S7-300 приоритетные классы фиксированные, в системах S7-400 вы можете изменять приоритеты при соответствующей параметризации CPU.

Обзор всех приоритетных классов совместно с заданными по умолчанию для каждого организационными блоками представлен параграфе 3.1.2 «Приоритетные классы».

Запрет прерываний

Организационные блоки зависящего от событий программного сканирования могут быть заблокированы или разблокированы с помощью системных функций SFC 39 DIS_IRT и SFC 40 EN_IRT и

задержаны и разрешены при помощи SFC 41 DIS_AIRT и SFC 42 EN_AIRT (см. параграф 21.7 «Обработка прерываний»).

Текущие сигнальные состояния

В процедуре обработки прерывания одним из обязательных условий является то, что вы работаете с текущими сигнальными состояниями модулей входов/выходов (а не с сигнальными состояниями входов, которые были обновлены при запуске главной программы) и записываете полученные сигнальные состояния непосредственно во входы/выходы (без ожидания обновления выходной таблицы образа процесса в конце главной программы).

В случае нескольких входов и выходов процедуры обработки прерывания достаточно обратиться напрямую к модулям входов/выходов с помощью операций загрузки и передачи или при помощи блочного элемента MOVE. Рекомендуется выполнить строгое разделение между главной программой и процедурой обработки прерывания в отношении входных/выходных сигналов.

Если требуется обработать много входных и выходных сигналов в процедуре обработки прерывания, то решением этой задачи в CPU серии S7-400 является использование образов подпроцессов. При определении адресов вы назначаете каждый модуль образу подпроцесса. С помощью SFC 26 UPDAT_PI и SFC 27 UPDAT_PO вы можете обновлять образы подпроцессов в пользовательской программе (см. также параграф 20.2.1 «Обновление образа процесса»).

В новых CPU S7-400 вы можете назначить образ входного и выходного подпроцесса любому организационному блоку прерывания (любому приоритетному классу пре-

рываний) и таким образом организовать автоматическое обновление образов процесса при возникновении прерывания.

Стартовая информация, временные локальные данные

В таблице 21.1 представлен обзор стартовой информации для событийных организационных блоков. В системах S7-300 доступные временные локальные данные имеют фиксированный размер – 256 байтов. В системах S7-400 вы можете определять размер временных локальных данных (локального стека) для приоритетного класса путем соответствующей параметризации CPU (параметр блока «local data», «локальные данные»), при этом общий размер не должен превышать максимум, задаваемый версией CPU. Обратите внимание, что число байтов временных локальных данных, используемых для приоритетного класса, должно быть не менее 20, чтобы вмещать стартовую информацию. Для неиспользуемых приоритетных классов указывайте ноль.

Таблица 21.1 Стартовая информация для организационных блоков прерываний

Байт	Мультипроцессорное прерывание	Аппаратные прерывания	Циклические прерывания	Прерывания задержки времени	Прерывания по времени суток
	ОВ 60	ОВ 40 – ОВ 47	ОВ 30 – ОВ 38	ОВ 20 – ОВ 23	ОВ 10 – ОВ 17
0	Класс события	Класс события	Класс события	Класс события	Класс события
1	Запуск события	Запуск события	Запуск события	Запуск события	Запуск события
2	Приоритетный класс	Приоритетный класс	Приоритетный класс	Приоритетный класс	Приоритетный класс
3	Номер ОВ	Номер ОВ	Номер ОВ	Номер ОВ	Номер ОВ
4	-	-	-	-	-
5	-	Идентификатор адреса	-	-	-
6..7	Идентификатор задания (INT)	Стартовый адрес модуля (WORD)	Сдвиг фазы в мс (WORD)	Идентификатор задания (INT)	Интервал (WORD)
8..9	-	Информация аппаратного прерывания (DWORD)	-	Истекшая задержка (TIME)	-
10..11	-		Время цикла в мс (INT)		-
12..19	Момент события (реализация) (DT)	Момент события (реализация) (DT)	Момент события (реализация) (DT)	Момент события (реализация) (DT)	Момент события (реализация) (DT)

21.2 Аппаратные прерывания

Аппаратные прерывания (hardware interrupts) используются для возможности немедленного обнаружения в пользовательской программе событий в управляемом процессе, позволяя выполнить ответную процедуру обработки прерывания. STEP 7 предоставляет организационные блоки с OB 40 по OB 47 для обслуживания аппаратных прерываний. Каждый из этих восьми блоков является фактически доступными, но это зависит от CPU.

Обработка аппаратных прерываний программируется в конфигурационных данных аппаратных средств. При помощи системных функций SFC 55 WR_PARM, SFC 56 WR_DPARM и SFC 57 PRAM_MOD вы можете (заново) параметризовать модули с возможностью аппаратного прерывания даже в режиме CPU RUN.

21.2.1 Генерирование аппаратного прерывания

Аппаратное прерывание генерируется в модулях с такой возможностью. Это могут быть модуль цифрового входа, который обнаруживает сигнал от процесса, или функциональный модуль, генерирующий аппаратное прерывание из-за процесса, проявляющегося в модуле.

По умолчанию аппаратные прерывания заблокированы. Для разрешения обслуживания аппаратного прерывания используется параметр (статический параметр), и вы можете определить, для приходящего, уходящего события или обоих (динамический параметр) должно быть сгенерировано аппаратное прерывание. Динамические параметры – это параметры, которые можно изменять во время исполнения программы с помощью SFC.

В оборудованных для этой цели интеллектуальных DP-ведомых вы можете инициировать прерывание процесса в главном CPU с помощью SFC 7 DP_PRAL.

Аппаратное прерывание распознается (подтверждается) в модуле, когда организационный блок, содержащий процедуру обслуживания этого прерывания, закончил выполнение.

Решение в S7-300

Если возникло событие во время исполнения блока OB аппаратного прерывания, которое само может вызвать генерирование того же аппаратного прерывания, то это прерывание будет потеряно в отсутствие события, вызвавшего аппаратное прерывание, после подтверждения.

Не делается различий между тем, приходит ли событие из модуля, чье аппаратное прерывание обрабатывается в текущий момент, или из другого модуля.

Пока обслуживается аппаратное прерывание, может быть сгенерировано диагностическое прерывание. Если в том же канале возникает другое аппаратное прерывание

между моментом генерирования первого аппаратного прерывания и моментом подтверждения этого прерывания, то через диагностическое прерывание в системную диагностику будет сообщено о потере более позднего прерывания.

Решение в S7-400

Если во время исполнения ОВ аппаратного прерывания возникает событие для аппаратного прерывания в том же канале, в том же модуле, который вызвал прерывание, то это прерывание будет потеряно. Если событие возникло в другом канале, в этом же или другом модуле, операционная система снова запустит ОВ, как только он завершит выполнение текущей обработки прерывания.

21.2.2 Обслуживание аппаратных прерываний

Запрос информации прерывания

Стартовый адрес модуля, который вызвал аппаратное прерывание, находится в байтах 6 и 7 стартовой информации блока ОВ аппаратного прерывания. Если этот адрес является адресом входа, то байт 5 стартовой информации содержит В#16#54; иначе он содержит В#16#55. Если модуль в запросе представляет собой модуль цифрового входа, то байты с 8 по 11 содержат данные о состоянии входов; в случае любого другого типа модуля эти байты содержат состояние прерывания модуля.

Обработка прерывания в процедуре запуска

В процедуре запуска модули не генерируют аппаратные прерывания. Обработка прерывания начинается с перехода CPU в режим RUN. Любые аппаратные прерывания во время перехода теряются.

Обработка ошибок

Если генерируется аппаратное прерывание, для которого в программе пользователя не имеется ОВ аппаратного прерывания, то операционная система вызывает ОВ 85 (ошибка исполнения программы).

Аппаратное прерывание подтверждается. Если ОВ 85 не запрограммирован, CPU переходит в состояние STOP.

Неуказанные при параметризации CPU аппаратные прерывания не обслуживаются, даже когда блоки ОВ для этих прерываний запрограммированы. CPU переходит в режим STOP.

Запрещение, задержка, разрешение

Вызов ОВ аппаратных прерываний может быть заблокирован и разблокирован с помощью системных функций SFC 39 DIS_IRT и SFC 40 EN_IRT, а также задержан и разрешен при помощи SFC 41 DIS_AIRT и SFC 42 EN_AIRT.

21.2.3 Конфигурирование аппаратных прерываний с помощью STEP 7

Аппаратные прерывания программируются в конфигурационных данных аппаратных средств. Откройте выбранный CPU с помощью команды меню Edit → Object Properties (Правка → Свойства объекта) и в диалоговом окне выберите вкладку «Interrupts» («Прерывания»).

В системах S7-300 (кроме CPU 318) приоритет по умолчанию для ОВ 40 равен 16 и не может быть изменен. В системах S7-400 и CPU 318 для любого возможного ОВ вы можете выбрать приоритет между 2 и 24 (в зависимости от версии CPU); приоритет 0 отменяет выполнение ОВ. Не следует назначать один приоритет дважды, потому что прерывания могут быть потеряны, когда одновременно возникает более 12 прерываний с одинаковым приоритетом.

Вы также должны разрешить вызов аппаратных прерываний в соответствующих модулях. Для этого данные модули параметризуются во многом так же, как CPU.

При сохранении конфигурации аппаратного обеспечения STEP 7 записывает скомпилированные данные в объект *System Data* (*Системные данные*) автономной пользовательской программы *Blocks* (*Блоки*). Отсюда вы можете загрузить данные параметризации в CPU, пока он находится в состоянии STOP. Данные параметризации для CPU вступают в силу немедленно после загрузки; данные, назначенные параметрам для модулей, оказывают воздействие после следующего запуска.

21.3 Циклические прерывания

Циклическое прерывание (watchdog interrupt) – это прерывание, которое генерируется через периодические интервалы времени, и которые инициируют исполнение блока ОВ циклического прерывания. Циклическое прерывание позволяет исполнять периодически определенные программы независимо от времени обработки циклической программы.

В STEP 7 для циклических прерываний задействованы организационные блоки с ОВ 30 по ОВ 38; эти девять организационных блоков фактически доступны в зависимости от версии используемого CPU.

Обработка циклических прерываний устанавливается в конфигурационных данных аппаратных средств при параметризации CPU.

21.3.1 Обработка циклических прерываний

Вызов циклических прерываний в S7-300

В S7-300 организационный блок для обслуживания циклических прерываний – ОВ 35, который имеет приоритет 12. При соответствующей параметризации CPU вы можете установить интервал в области от 1 миллисекунды до 1 минуты с шагом 1 миллисекунда.

Вызов циклических прерываний в S7-400

Циклическое прерывание определяется при параметризации CPU. Это прерывание имеет три параметра: интервал, сдвиг фазы и приоритет. Всех их можно задать. Задаваемые значения для интервала и сдвига фазы берутся из области от 1 миллисекунды до 1 минуты с шагом 1 миллисекунда; значение приоритета может быть выбрано из отрезка от 2 до 24 или ноль в зависимости от CPU (ноль означает, что циклическое прерывание неактивно).

STEP 7 предоставляет организационные блоки, приведенные в таблице 21.2, в их максимальных конфигурациях.

Сдвиг фазы

Сдвиг фазы может быть использован для разноса выполнения процедур обработки циклических прерываний, несмотря на занесение нескольких процедур их в один интервал. Использование сдвига фазы позволяет достичь более высокой интервальной точности.

Начальное время временного интервала и сдвиг фазы являются моментом перехода от режима START-UP (Запуск) к состоянию RUN центрального процессора. Таким

образом, момент вызова ОВ циклического прерывания складывается из временного интервала и сдвига фазы.

Таблица 21.2 Значения по умолчанию для циклических прерываний

ОВ	Временной интервал	Фаза	Приоритет
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

На рисунке 21.1 показан пример, комментирующий этот факт. Сдвига фазы для временного интервала 1 не установлено, временной интервал 2 в два раза длиннее временного интервала 1. Из-за сдвига фазы временного интервала 2 блоки ОВ для временного интервала 2 и блоки ОВ для временного интервала 1 одновременно не вызываются. Таким образом, блокам ОВ с низким приоритетом не надо ожидать, и они могут точно выдерживать свой временной интервал.

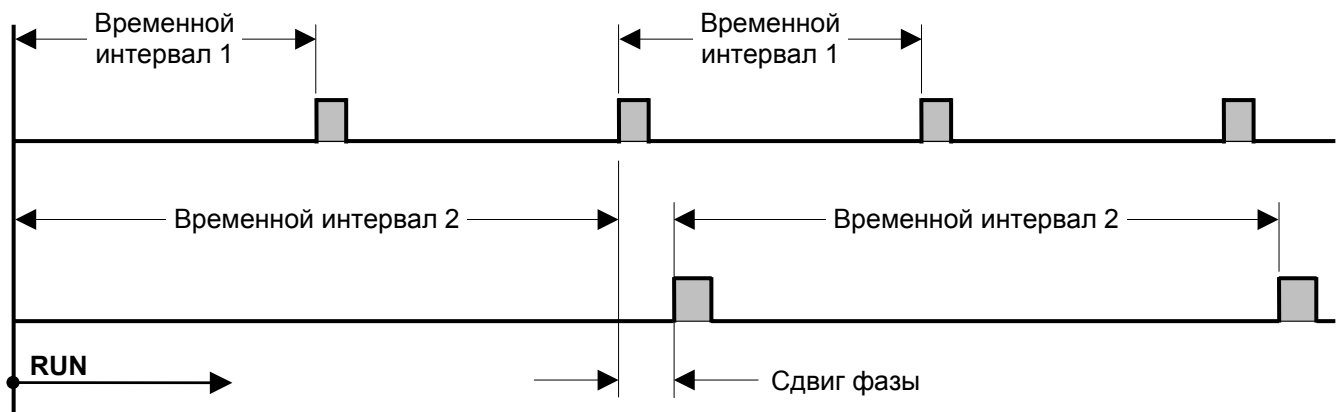


Рисунок 21.1 Пример сдвига фазы для циклического прерывания

Рабочие характеристики во время запуска

Циклические прерывания не могут обслуживаться в ОВ запуска. Временные интервалы не начинаются до перехода CPU в режим RUN.

Рабочие характеристики при ошибке

Если во время исполнения ОВ обработки соответствующего циклического прерывания снова генерируется это же прерывание, операционная система вызывает ОВ 80 (временная ошибка). Если ОВ не запрограммирован, то CPU переходит в режим STOP.

Операционная система сохраняет циклическое прерывание, которое не было обработано, откладывая его обработку до следующей возможности. Для приоритетного класса сохраняется только одно необслуженное циклическое прерывание, несмотря на то, сколько накапливается необслуженных циклических прерываний.

Циклические прерывания, не выбранные при параметризации CPU, не могут быть обработаны, даже если доступен соответствующий ОВ. В таком случае CPU переходит в состояние STOP.

Запрещение, задержка, разрешение

Вызов ОВ циклических прерываний может быть заблокирован и разблокирован с помощью системных функций SFC 39 DIS_IRT и SFC 40 EN_IRT, а также задержан и разрешен при помощи SFC 41 DIS_AIRT и SFC 42 EN_AIRT.

21.3.2 Конфигурирование циклических прерываний с помощью STEP 7

Циклические прерывания настраиваются посредством конфигурационных данных аппаратных средств. Просто откройте выбранный CPU с помощью команды меню Edit → Object Properties (Правка → Свойства объекта) и в диалоговом окне выберите вкладку «Cyclic Interrupt» («Циклическое прерывание»).

В контроллерах S7-300 (кроме CPU 318) приоритет обработки постоянно установлен в значение 12. В контроллерах S7-400 и CPU 318 вы можете выбрать приоритет между 2 и 24 (в зависимости от версии CPU) для любого возможного ОВ; приоритет 0 отменяет ОВ, которому он присвоен. Не следует назначать приоритет более одного раза, так как прерывания могут быть потеряны, если одновременно возникает более 12 прерываний с одинаковым приоритетом.

Интервал для каждого ОВ выбирается в пункте «Execution» («Выполнение»), момент отложенного вызова – в «Phase Offset» («Сдвиг фазы»).

При сохранении конфигурации аппаратного обеспечения STEP 7 записывает скомпилированные данные в объект *System Data* (*Системные данные*) автономной пользовательской программы *Blocks* (*Блоки*). Отсюда вы можете загрузить данные параметризации в CPU, пока он находится в состоянии STOP. Данные параметризации для CPU вступают в силу немедленно после загрузки.

21.4 Прерывания по времени суток

Прерывания по времени суток (time-of-day interrupts) применяются, когда требуется выполнить программу в определенное время, либо один раз, либо периодически, например, ежедневно. В STEP 7 организационные блоки с OB 10 по OB 17 предоставлены для обслуживания прерываний по времени суток; эти восемь организационных блоков фактически доступны в зависимости от используемого CPU.

Сконфигурировать прерывания по времени суток вы можете в конфигурационных данных аппаратного обеспечения, управление ими можно осуществлять во время исполнения программно с использованием системных функций. Необходимым условием надлежащей обработки прерываний по времени суток является корректная установка таймера реального времени CPU.

21.4.1 Обработка прерываний по времени суток

Общие замечания

Чтобы запустить прерывание по времени суток, вы должны сначала установить время запуска, затем активировать прерывание. Вы можете выполнять два действия отдельно посредством конфигурационных данных аппаратных средств или с помощью SFC. Заметьте, что при активировании через конфигурационные данные аппаратных средств прерывание по времени суток запускается автоматически после параметризации CPU.

Запуск прерывания по времени суток можно осуществить двумя способами:

- один раз: соответствующий OB вызывается только один раз в определенное время или
- периодически: в зависимости от назначений параметров соответствующий OB запускается каждую минуту, каждый час, ежедневно, еженедельно, ежемесячно или ежегодно.

После однократного вызова OB прерывания по времени суток это прерывание отменяется (аннулируется). Вы также можете отменить (cancel) прерывание по времени суток с помощью SFC 29 CAN_TINT.

Если вы хотите еще раз использовать аннулированное прерывание по времени суток, то следует снова установить время запуска, затем повторно активировать прерывание.

Вы можете запросить информацию о состоянии прерывания по времени суток, применив SFC 31 QRY_TINT.

Рабочие характеристики во время запуска

При «холодном» рестарте или полном рестарте операционная система очищает все установки, определенные с помощью SFC. Установки, сделанные через конфигурационные данные аппаратных средств, сохраняются. В случае «теплого» рестарта CPU продолжает обслуживание прерываний по времени суток в первом полном цикле сканирования главной программы.

Вы можете запросить информацию о состоянии прерываний по времени суток в ОВ запуска путем вызова SFC 31 и впоследствии отменить или переустановить и повторно активировать прерывания. Прерывания по времени суток обслуживаются только в режиме RUN CPU.

Рабочие характеристики при возникновении ошибки

Если ОВ прерывания по времени суток было вызвано, но не запрограммировано, то операционная система вызывает ОВ 85 (ошибка исполнения программы). Если ОВ 85 не запрограммировано, то CPU перейдет в состояние STOP.

Неотмеченные при параметризации CPU прерывания по времени суток не обслуживаются, даже если доступен соответствующий ОВ. CPU переходит в режим STOP.

Если вы однократно активируете прерывание по времени суток, и если время запуска уже прошло (с точки зрения таймера реального времени), то операционная система вызывает ОВ 80 (временная ошибка). Если ОВ 80 недоступен, CPU переходит в состояние STOP.

Если прерывание по времени суток активируется периодически, и если время запуска уже прошло (с точки зрения таймера реального времени), то ОВ прерывания по времени суток будет выполнен в следующий раз, когда должен наступить заданный период времени.

Если вы переводите таймер реального времени вперед для коррекции или синхронизации, тем самым перескакивая время запуска прерывания по времени суток, то операционная система вызывает ОВ 80 (временная ошибка). В таком случае ОВ прерывания по времени суток выполняется ровно один раз.

Если вы переводите таймер реального времени назад с целью коррекции или синхронизации, то активированный ОВ прерывания по времени суток не будет исполняться в моменты, которые уже прошли.

Если ОВ прерывания по времени суток еще исполняется, когда происходит следующий (периодический) вызов, операционная система активирует ОВ 80 (временная ошибка). Когда ОВ 80 и ОВ прерывания по времени суток выполнены, ОВ прерывания по времени суток запускается вновь.

Запрещение, задержка, разрешение

Вызовы ОВ прерываний по времени суток могут быть заблокированы и разблокированы с помощью системных функций SFC 39 DIS_IRT и SFC 40 EN_IRT, а также задержаны и разрешены при помощи SFC 41 DIS_AIRT и SFC 42 EN_AIRT.

21.4.2 Конфигурирование прерываний по времени суток с помощью STEP 7

Прерывания по времени суток конфигурируются посредством конфигурационных данных аппаратных средств. Откройте отмеченный CPU с помощью команды меню Edit → Object Properties (Правка → Свойства объекта) и в диалоговом окне выберите вкладку «Time-of-day» («Время суток»).

В контроллерах S7-300 (кроме CPU 318) приоритет обработки постоянно установлен в значение 2. В контроллерах серии S7-400 и CPU 318 вы можете установить приоритет между 2 и 24 (в зависимости от версии CPU) для любого возможного ОВ; приоритет 0 отменяет ОВ. Не следует назначать приоритет более одного раза, так как прерывания могут быть потеряны, если одновременно возникает более 12 прерываний с одинаковым приоритетом.

Опция «Active» («Активно») активирует автоматический запуск прерывания по времени суток. Опция «Execution» («Выполнение») отображает список, позволяющий выбрать, либо ОВ будет исполняться однократно, либо в определенные интервалы. Последним параметром является время запуска (дата и время).

При сохранении конфигурации аппаратного обеспечения STEP 7 записывает компилированные данные в объект *System Data* (*Системные данные*) автономной пользовательской программы *Blocks* (*Блоки*). Отсюда вы можете загрузить данные параметризации в CPU, пока он находится в режиме STOP. Данные параметризации вступают в силу немедленно.

21.4.3 Системные функции для прерываний по времени суток

Для управления прерыванием по времени суток могут использоваться следующие системные функции:

- SFC 28 SET_TINT
Установка прерывания по времени суток;
- SFC 29 CAN_TINT
Отмена прерывания по времени суток;
- SFC 30 ACT_TINT
Активирование прерывания по времени суток;
- SFC 31 QRY_TINT
Опрос прерывания по времени суток.

Параметры этих системных функций приведены в таблице 21.3.

Таблица 21.3 Параметры SFC для прерываний по времени суток

SFC	Параметр	Объявление	Тип данных	Содержимое, описание
28	OB_NR	INPUT	INT	Номер ОБ, предназначенного для однократного или периодического вызова в определенное время
	SDT	INPUT	DT	Дата и время запуска в формате DATE_AND_TIME
	PERIOD	INPUT	WORD	Период, на котором основывается время запуска: W#16#0000 = Однократно W#16#0201 = Каждую минуту W#16#0401 = Каждый час W#16#1001 = Ежедневно W#16#1201 = Еженедельно W#16#1401 = Ежемесячно W#16#2001 = Последний в месяце (412) W#16#1801 = Ежегодно
	RET_VAL	OUTPUT	INT	Информация об ошибке
	OB_NR	INPUT	INT	Номер ОБ, время запуска которого должно быть удалено
29	RET_VAL	OUTPUT	INT	Информация об ошибке
	OB_NR	INPUT	INT	Номер ОБ, который должен быть активирован
30	RET_VAL	OUTPUT	INT	Информация об ошибке
	OB_NR	INPUT	INT	Номер ОБ, состояние которого будет опрошено
31	RET_VAL	OUTPUT	INT	Информация об ошибке
	STATUS	OUTPUT	WORD	Состояние прерывания по времени суток

SFC 28 SET_TINT

Установка прерывания по времени суток

Задать время запуска для прерывания по времени суток вы можете, вызвав системную функцию SFC 28 SET_TINT. SFC 28 устанавливает только время запуска; чтобы запустить ОБ прерывания по времени суток, вы должны активировать прерывание по времени суток с помощью SFC 30 ACT_TINT. Время запуска указывается в параметре SDT в формате DATE_AND_TIME, например, DT#1997-06-03-08:30. Операционная система игнорирует секунды и миллисекунды и обнуляет эти значения. Установка времени запуска перезаписывает старое значение времени запуска, если оно было.

Активное прерывание по времени суток отменяется, то есть оно должно быть вновь активировано.

SFC 30 ACT_TINT**Активирование прерывания по времени суток**

Прерывание по времени суток активируется путем вызова системной функции SFC 30 ACT_TINT. Когда прерывание по времени суток активировано, принимается, что время для прерывания задано. Если, в случае однократного прерывания, время запуска уже прошло, то SFC 30 сообщает об ошибке. В случае периодического запуска операционная система вызывает соответствующий ОВ в следующий подходящий момент времени. Однажды обработанное однократное прерывание по времени суток для всех практических целей аннулируется. Если требуется, вы можете переустановить и повторно активировать его (с другим временем запуска).

SFC 29 CAN_TINT**Отмена прерывания по времени суток**

С помощью системной функции SFC 29 CAN_TINT вы можете удалить время запуска, деактивировав таким образом прерывание по времени суток. Соответствующий ОВ больше не вызовется. Если вы хотите снова использовать это же прерывание по времени суток, необходимо сначала установить время запуска, затем активировать прерывание.

SFC 31 QRY_TINT**Опрос прерывания по времени суток**

Вы можете запросить данные о состоянии прерывания по времени суток путем вызова системной функции SFC 31 QRY_TINT. Требуемая информация возвращается в параметре STATUS.

Когда биты установлены в сигнальное состояние «1», они имеют следующие значения:

- 0 Прерывание по времени суток отключено операционной системой;
- 1 Новое прерывание по времени суток отклонено;
- 2 Прерывание по времени суток не активировано и не окончено;
- 3 (Зарезервирован)
- 4 ОВ прерывания по времени суток загружен;
- 5 Нет запрета;
- 6 (И следующие - зарезервированы).

21.5 Прерывания задержки времени

Прерывание задержки времени (time-delay interrupt) позволяет вам реализовать таймер задержки независимо от стандартных таймеров. В STEP 7 организационные блоки с OB 20 по OB 23 предназначены для прерываний задержки времени. Эти четыре организационных блока фактически доступны в зависимости от версии CPU.

Приоритеты для прерывания задержки времени программируются в конфигурационных данных аппаратных средств; для управления используются системные функции.

21.5.1 Обработка прерываний задержки времени

Общие замечания

Прерывание задержки времени запускается по вызову SFC 32 SRT_DINT; эта системная функция также передает операционной системе интервал задержки и номер выбранного организационного блока. Когда интервал истекает, вызывается OB.

Вы можете отменить обслуживание прерывания задержки времени, в таком случае связанный OB больше не будет исполняться.

Можно запросить состояние прерывания задержки времени с помощью SFC 34 QRY_DINT.

Рабочие характеристики при запуске

При «холодном» рестарте или полном перезапуске операционная система удаляет все запрограммированные установки для прерываний задержки времени. В случае «теплого» рестарта установки сохраняются до обработки в режиме RUN CPU, посредством чего «оставшийся цикл» считается частью процедуры запуска.

Вы можете запустить прерывание задержки времени путем вызова SFC 32. Когда интервал задержки заканчивается, CPU должен быть в режиме RUN, чтобы иметь возможность выполнить соответствующий организационный блок. В другом случае CPU задерживает вызов организационного блока до завершения процедуры запуска, затем вызывает OB прерывания задержки времени до первой сети в главной программе.

Рабочие характеристики в случае ошибки

Если OB прерывания задержки времени не был запрограммирован, операционная система вызывает OB 85 (ошибка выполнения программы). Если в программе пользователя OB 85 отсутствует, то CPU переходит в состояние STOP.

Если интервал задержки истек, а соответствующий ОВ еще выполняется, то операционная система вызывает ОВ 80 (временная ошибка) или переходит в режим STOP, если в пользовательской программе нет ОВ 80.

Прерывания задержки времени, которые не выбраны во время параметризации CPU, не могут быть обслужены, даже если запрограммирован соответствующий ОВ. CPU переходит в состояние STOP.

Запрещение, задержка, разрешение

Блоки ОВ прерываний задержки времени могут быть заблокированы и разблокированы с помощью системных функций SFC 39 DIS_IRT и SFC 40 EN_IRT, а также их вызовы могут быть задержаны и разрешены при помощи SFC 41 DIS_AIRT и SFC 42 EN_AIRT.

21.5.2 Конфигурирование прерываний задержки времени с помощью STEP 7

Прерывания задержки времени конфигурируются в конфигурационных данных аппаратных средств. Просто откройте отмеченный CPU с помощью команды меню Edit → Object Properties (Правка → Свойства объекта) и в диалоговом окне выберите вкладку «Interrupts» («Прерывания»).

В контроллерах S7-300 (кроме CPU 318) приоритет постоянно установлен в значение 3. В контроллерах серии S7-400 и CPU 318 вы можете выбрать приоритет между 2 и 24 (в зависимости от версии CPU) для любого возможного ОВ; выбор приоритета 0 отменяет ОВ. Не следует назначать приоритет более одного раза, так как прерывания могут быть потеряны, если одновременно возникает более 12 прерываний с одинаковым приоритетом.

При сохранении конфигурации аппаратного обеспечения STEP 7 записывает скомпилированные данные в объект *System Data* (*Системные данные*) автономной пользовательской программы *Blocks* (*Блоки*). Отсюда вы можете загрузить данные параметризации в CPU, пока он находится в режиме STOP. Данные параметризации вступают в силу немедленно.

21.5.3 Системные функции для прерываний задержки времени

Управление прерыванием задержки времени может осуществляться с использованием следующих системных функций:

- SFC 32 SRT_DINT
Запуск прерывания задержки времени;
- SFC 33 CAN_DINT
Отмена прерывания задержки времени;

- SFC 34 QRY_DINT
Опрос прерывания задержки времени.

Параметры этих системных функций показаны в таблице 21.4.

Таблица 21.4 Параметры SFC для прерываний задержки времени

SFC	Параметр	Объявление	Тип данных	Содержимое, описание
32	OB_NR	INPUT	INT	Номер ОБ, предназначенного для вызова, когда истечет интервал задержки
	DTIME	INPUT	TIME	Интервал задержки; допустимы: T#1 мс,..., T#1 м
	SIGN	INPUT	WORD	Идентификация задания в стартовой информации соответствующего ОБ, когда ОБ вызван (произвольные литеры)
	RET_VAL	OUTPUT	INT	Информация об ошибке
33	OB_NR	INPUT	INT	Номер отменяемого ОБ
	RET_VAL	OUTPUT	INT	Информация об ошибке
34	OB_NR	INPUT	INT	Номер ОБ, состояние которого опрашивается
	RET_VAL	OUTPUT	INT	Информация об ошибке
	STATUS	OUTPUT	WORD	Состояние прерывания задержки времени

SFC 32 SRT_DINT

Запуск прерывания задержки времени

Прерывание задержки времени запускается путем вызова системной функции SFC 32 SRT_DINT. Вызов SFC также является моментом запуска запрограммированного интервала задержки. По истечении интервала задержки CPU вызывает запрограммированный ОБ и передает в стартовую информацию для этого ОБ значение времени задержки и идентификатор задания. Идентификатор задания указывается в параметре SIGN для SFC 32; вы можете прочитать это же значение в байтах 6 и 7 стартовой информации соответствующего ОБ прерывания задержки времени. Приращение для задержки времени составляет 1 мс. Точность задержки времени также 1 мс. Обратите внимание на то, что исполнение ОБ прерывания задержки времени само может быть отложено, когда при вызванном ОБ прерывания задержки времени обрабатываются организационные блоки с более высоким приоритетом. Вы можете перезаписать значение задержки времени новым путем повторного вызова SFC 32. Новая задержка времени вступает в силу с вызовом SFC.

SFC 33 CAN_DINT

Отмена прерывания задержки времени

Чтобы аннулировать прерывание задержки времени, вы можете вызвать системную функцию SFC 33 CAN_DINT. В этом случае запрограммированный организационный блок не вызывается.

SFC 34 QRY_DINT**Опрос прерывания задержки времени**

Системная функция SFC 34 QRY_DINT информирует вас о состоянии прерывания задержки времени. Вы можете выбрать прерывание задержки времени через номер ОВ, и информация о состоянии будет возвращена в параметре STATUS.

Когда биты имеют сигнальное состояние «1», они обозначают следующее:

- 0 Прерывание задержки времени отключено операционной системой;
- 1 Новое прерывание задержки времени отклонено;
- 2 Прерывание задержки времени не активировано и не окончено;
- 3 (Зарезервирован)
- 4 ОВ прерывания задержки времени загружен;
- 5 Нет запрета;
- 6 (И следующие - зарезервированы).

21.6 Мультипроцессорное прерывание

Мультипроцессорное прерывание (multiprocessor interrupt) позволяет осуществить синхронный отклик на событие во всех CPU в мультипроцессорном режиме. Мультипроцессорное прерывание вызывается с помощью SFC 35 MP_ALM. Для обслуживания мультипроцессорного прерывания используется организационный блок OB 60, имеющий фиксированный приоритет 25.

Общие замечания

Вызов SFC 35 MP_ALM инициирует выполнение OB мультипроцессорного прерывания. Если CPU работает в однопроцессорном режиме, то OB 60 запускается немедленно. В мультипроцессорном режиме OB 60 запускается одновременно на всех задействованных CPU, то есть даже CPU, в котором была вызвана SFC 35, ожидает до того, как вызвать OB 60, пока остальные CPU не сообщат о своей готовности.

Мультипроцессорное прерывание не программируется в конфигурационных данных аппаратуры; оно уже присутствует в любом CPU с возможностью обработки данных в многопроцессорной системе. Однако, несмотря на этот факт, во вкладке «Local Data» («Локальные данные») CPU под приоритетным классом 25 должно быть зарезервировано достаточное число байтов (по крайней мере, 20).

Рабочие характеристики при запуске

Мультипроцессорное прерывание вызывается только в режиме центрального процессора RUN. Вызов SFC 35 в процедуре запуска завершается после возвращения в качестве значения функции ошибки 32929 (W#16#80A1).

Рабочие характеристики при ошибке

Если OB 60 еще находится в обработке, когда повторно был вызван SFC 35, то операционная система возвращает в качестве значения функции код ошибки 32928 (W#16#80A0). OB 60 запускается не во всех CPU.

Недоступность OB 60 в одном из CPU в момент его вызова или блокировка или задержка системными функциями его исполнения воздействия не оказывает, и SFC 35 об ошибке не сообщает.

Запрещение, задержка, разрешение

Мультипроцессорные OB могут быть заблокированы и разблокированы с помощью системных функций SFC 39 DIS_IRT и SFC 40 EN_IRT, а также задержаны и разрешены при помощи SFC 41 DIS_AIRT и SFC 42 EN_AIRT.

SFC 35 MP_ALM**Мультипроцессорное прерывание**

Мультипроцессорное прерывание вызывается при помощи системной функции SFC 35 MP_ALM. Ее параметры приведены в таблице 21.5.

Параметр JOB позволяет вам передать идентификатор задания. То же значение находится в байтах 6 и 7 стартовой информации OB 60 во всех CPU.

Таблица 21.4 Параметры для SFC 35 MP_ALM

Параметр	Объявление	Тип данных	Содержимое, описание
JOB	INPUT	BYTE	Идентификация задания из области значений от B#16#00 до B#16#0F
RET_VAL	OUTPUT	INT	Информация об ошибке

21.7 Обработка прерываний

Для обработки прерываний и асинхронных ошибок доступны следующие системные функции:

- SFC 39 DIS_IRT
Блокировка прерываний;
- SFC 40 EN_IRT
Разрешение прерываний;
- SFC 41 DIS_AIRT
Задержка прерываний;
- SFC 40 EN_AIRT
Разрешение задержанных прерываний.

В таблице 21.6 показаны параметры для этих системных функций.

Эти системные функции воздействуют на все прерывания и все асинхронные ошибки. Системные функции с SFC 36 по SFC 38 предоставляются для обработки синхронных ошибок.

Таблица 21.6 Параметры SFC для обработки прерываний

SFC	Параметр	Объявление	Тип данных	Содержимое, описание
39	MODE	INPUT	BYTE	Режим запрета (см. текст)
	OB_NR	INPUT	INT	Номер ОБ (см. текст)
	RET_VAL	OUTPUT	INT	Информация об ошибке
40	MODE	INPUT	BYTE	Режим разрешения (см. текст)
	OB_NR	INPUT	INT	Номер ОБ (см. текст)
	RET_VAL	OUTPUT	INT	Информация об ошибке
41	RET_VAL	OUTPUT	INT	(Новые) количество задержек
42	RET_VAL	OUTPUT	INT	Количество оставшихся задержек

SFC 39 DIS_IRT

Блокировка прерываний

Системная функция SFC 39 DIS_IRT запрещает обработку новых прерываний и асинхронных ошибок. Все новые прерывания и асинхронные ошибки отклоняются. Если прерывание или асинхронная ошибка возникают после включения блокировки, то организационный блок не выполняется; если ОБ не существует, то CPU переходит в состояние STOP.

Запрет остается в силе для всех приоритетных классов до его отмены с использованием SFC 40 EN_IRT. После полного рестарта допускаются все прерывания и синхронные ошибки.

Параметры MODE и OB_NR служат для определения, какие прерывания и асинхронные ошибки следует запретить. MODE = B#16#00 запрещает все прерывания и асинхронные ошибки. MODE = B#16#01 блокирует класс прерываний, номер первого OB которого указывается в параметре OB_NR.

Например, MODE = B#16#01 и OB_NR = 40 запрещают все аппаратные прерывания; OB = 80 блокирует все асинхронные ошибки. MODE = B#16#02 отменяет прерывание или асинхронную ошибку, номер OB которых вводится в параметр OB_NR.

Независимо от запрета операционная система помещает каждое новое прерывание или асинхронную ошибку в диагностический буфер.

SFC 40 EN_IRT

Разрешение прерываний

Системная функция SFC 40 EN_IRT разрешает прерывания и асинхронные ошибки, заблокированные функцией SFC 39 DIS_IRT. Прерывание или асинхронная ошибка, возникающие после разблокировки, будут обслуживаться соответствующим организационным блоком; если в пользовательской программе организационный блок отсутствует, то CPU переходит в состояние STOP (за исключением случая OB 81, организационного блока для сбоя электропитания).

Параметры MODE и OB_NR определяют, какие прерывания и асинхронные ошибки разблокировать. MODE = B#16#00 разрешает все прерывания и асинхронные ошибки. MODE = B#16#01 разблокирует класс прерываний, номер первого OB которого указывается в параметре OB_NR. MODE = B#16#02 снимает запрет с прерывания или асинхронной ошибки, номер OB которых вводится в параметр OB_NR.

SFC 41 DIS_AIRT

Задержка прерываний

Системная функция SFC 41 DIS_AIRT вводит задержку обслуживания новых прерываний и асинхронных ошибок более высокого приоритета. Задержка означает, что операционная система сохраняет прерывания и асинхронные ошибки, возникшие во время задержки, и обрабатывает их по завершению интервала задержки. Если была вызвана SFC 41, программа в текущем организационном блоке (в текущем приоритетном классе) не будет остановлена прерыванием более высокого приоритета; прерывания или асинхронные ошибки не теряются.

Задержка остается в силе до завершения исполнения текущего OB или вызова SFC 42 EN_AIRT.

Вы можете вызывать SFC 41 последовательно несколько раз. Параметр RET_VAL показывает количество вызовов. Вызвать SFC 42 вы должны ровно столько раз,

сколько было вызовов SFC 41, чтобы вновь разрешить прерывания и асинхронные ошибки.

SFC 42 EN_AIRT

Разрешение задержанных прерываний

Системная функция SFC 42 EN_AIRT осуществляет разблокировку прерываний и асинхронных ошибок, для которых функцией SFC 41 была введена задержка. Вы должны вызвать SFC 42 ровно столько раз, сколько было вызовов SFC 41 (в текущем ОВ). Параметр RET_VAL показывает число остающихся в силе задержек; если RET_VAL = 0, то прерывания и асинхронные ошибки разблокированы. Если вы вызовете SFC 42 без предварительного вызова SFC 41, то RET_VAL будет содержать значение 32896 (W#16#8080).

Содержание главы 22

<u>22</u>	<u>Особенности рестарта</u>	4
<u>22.1</u>	<u>Общие замечания</u>	4
<u>22.1.1</u>	<u>Рабочие режимы</u>	4
<u>22.1.2</u>	<u>Режим HOLD</u>	5
<u>22.1.3</u>	<u>Блокировка модулей выходов</u>	6
<u>22.1.4</u>	<u>Организационные блоки рестарта</u>	6
<u>22.2</u>	<u>Включение электропитания</u>	8
<u>22.2.1</u>	<u>Режим STOP</u>	8
<u>22.2.2</u>	<u>Сброс памяти</u>	8
<u>22.2.3</u>	<u>Реманентность</u>	9
<u>22.2.4</u>	<u>Параметризация рестарта</u>	10
<u>22.3</u>	<u>Типы рестарта</u>	11
<u>22.3.1</u>	<u>Режим STURT-UP (Запуск)</u>	11
<u>22.3.2</u>	<u>«Холодный» рестарт</u>	11
<u>22.3.3</u>	<u>Полный рестарт</u>	13
<u>22.3.4</u>	<u>«Теплый» рестарт</u>	14
<u>22.4</u>	<u>Определение адреса модуля</u>	16
<u>22.5</u>	<u>Параметризация модулей</u>	19

22 Особенности рестарта

22.1 Общие замечания

22.1.1 Рабочие режимы

Перед тем, как CPU начнет обработку главной программы, следующую за включением питания, он выполняет процедуру рестарта (повторного запуска). START-UP (Запуск) является одним из рабочих режимов CPU наряду с режимами STOP или RUN. В этой главе рассказывается о действиях CPU при переходе из режима START-UP и в него, а также в ходе самой процедуры рестарта.

Обратимся к рисунку 22.1. После включения электропитания ① CPU находится в режиме STOP. Если переключатель на передней панели CPU находится в положении RUN или RUN-P, то CPU переключается в режим START-UP ②, затем в режим RUN ③. Если во время работы CPU в режимах START-UP или RUN возникает «неисправимая» ошибка, или если вы переводите переключатель в положение STOP, то CPU возвращается в состояние STOP ④ ⑤.

Программа пользователя может тестироваться с использованием точек останова (breakpoints) в режиме пошагового исполнения операций, CPU при этом находится в состоянии HOLD (Задержка). Вы можете переключиться в этот режим как из RUN, так и из START-UP, и вернуться в исходный режим, когда тестирование прерывается ⑥ ⑦. Также можно из состояния HOLD перевести CPU в режим STOP ⑧.

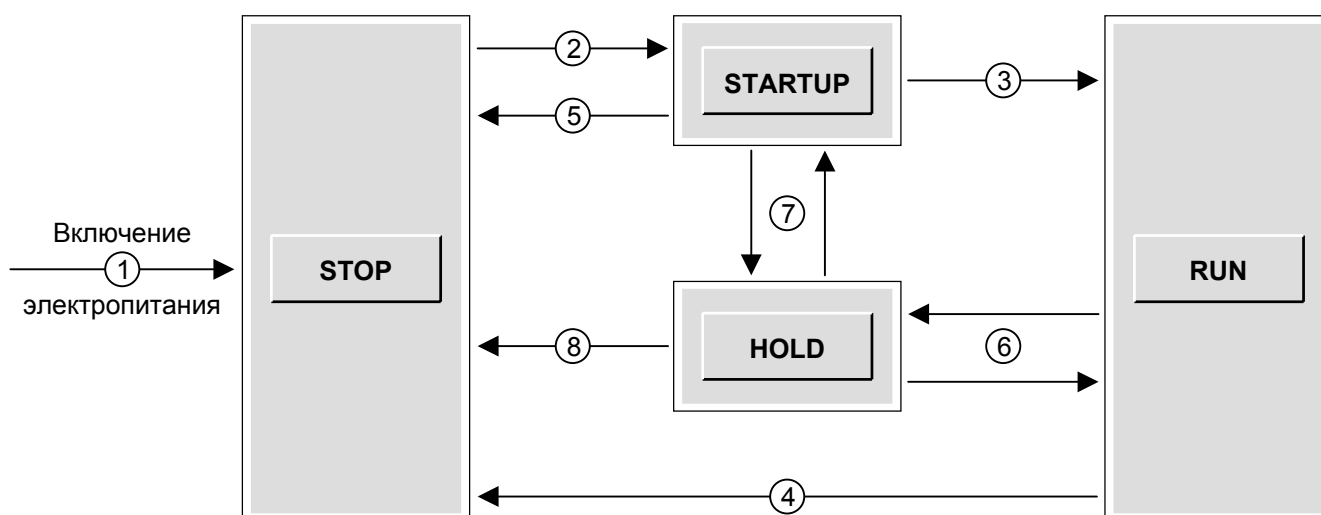


Рисунок 22.1 Рабочие режимы CPU

При параметризации CPU вы можете на вкладке «Restart» («Рестарт») определить такие характеристики рестарта, как максимально допустимый промежуток времени для сигналов Ready (Готов) модулей после включения питания или, должен ли CPU

запускаться, когда конфигурационные данные не соответствуют фактической конфигурации, или в каком режиме должен быть рестарт CPU.

SIMATIC S7 имеет три режима рестарта, которые называются «холодный» *рестарт* (*cold restart*), *полный рестарт* (*complete restart*) и «теплый» рестарт (*warm restart*). При «холодном» рестарте или полном рестарте главная программа всегда обрабатывается с начала. «Теплый» рестарт позволяет продолжить исполнение программы с точки прерывания и «закончить» цикл.

CPU S7, поставляемые до октября 1998 г., оснащены механизмами полного рестарта и «теплого» рестарта. Полный рестарт соответствует по функциональности «теплому» рестарту.

Вы можете провести однократное сканирование программы в режиме START-UP. STEP 7 предоставляет организационные блоки OB 102 («холодный» рестарт), OB 100 (полный рестарт или «теплый» рестарт) и OB 101 («теплый» рестарт) специально для этой цели. Примеры приложений: параметризация модулей, если она еще не проведена CPU, и программирование значений по умолчанию для вашей главной программы.

22.1.2 Режим HOLD

CPU переходит в режим HOLD, когда вы тестируете программу с точками останова (в «пошаговом» режиме). Светодиод STOP при этом загорается, а светодиод RUN мигает.

В режиме HOLD модули выходов заблокированы. Запись в модули производит действия с памятью модулей, но не переключает сигнальные состояния «out» выходов модулей. Блокировка модулей снимается после выхода из режима HOLD.

В режиме HOLD все, что имеет дело с расчетом времени, прерывается. Сюда относятся, к примеру, обработка таймеров, тактового меркера и таймеров времени исполнения, наблюдение времени цикла и минимального времени цикла сканирования, а также обслуживание прерываний времени суток и задержки времени. Исключение: таймер реального времени продолжает функционировать в нормальном режиме.

Каждый раз, когда в режиме тестирования осуществляется переход к следующему оператору, таймеры длительности прогона программы в своем счете продвигаются немного вперед, моделируя таким образом динамичную систему, схожую с «нормальным» сканированием программы.

В состоянии HOLD CPU допускает пассивную коммуникацию, то есть он может принимать глобальные данные или принимать участие в одностороннем обмене данными.

Если происходит сбой в электропитании при работе CPU в режиме HOLD, то CPU, оснащенный резервными батареями, переходит после восстановления питания в состояние STOP. CPU без резервных батарей выполняют автоматический полный рестарт.

22.1.3 Блокировка модулей выходов

В режимах STOP и HOLD модули заблокированы (сигнал OD – output disable, блокировка выходов). Заблокированные модули выходов выдают нулевой сигнал или, если он имеет возможность, заменяющее значение. Посредством таблицы переменных (variable table) вы можете управлять выходами модулей с помощью функции «Isolate PQ» («Изолирование PQ») даже в режиме STOP.

Во время рестарта модули выходов остаются заблокированными. Модули активируются, только когда начинается циклическое сканирование. И сигнальные состояния, находящиеся в памяти модулей (не образ процесса!), направляются в выходы.

Пока модули выходов заблокированы, память модулей может быть установлена либо с помощью прямого доступа (операторов передачи или блочных элементов MOVE с областью адресов PQ), либо посредством передачи выходной таблицы образа процесса. Если CPU снимает сигнал Disable (Блокировка), то сигнальные состояния в памяти модулей передаются во внешние выходы.

При «холодном» рестарте (ОВ 102) и полном рестарте (ОВ 100) образы процесса и память модулей очищаются. Если вы хотите сканировать входы в ОВ 102 или ОВ 100, вы должны загрузить сигнальные состояния из модуля, используя прямой доступ. Тогда вы сможете установить входы (передать их, например, с помощью операторов загрузки или блочного элемента MOVE из области адресов PI в область адресов I), затем работать с входами. Если вы хотите, чтобы при переходе от полного рестарта к циклической программе (до вызова ОВ 1) были установлены определенные выходы, то вы должны использовать прямой доступ для адресации модулей выходов. Недостаточно просто установить выходы (в образе процесса), так как выходная таблица образа процесса не передается в конце процедуры полного рестарта.

При «теплом» рестарте «старые» входная и выходная таблицы образа процесса, используемые до выключения питания или перехода в режим STOP, применяются в ОВ 101 и в оставшейся части цикла. В конце этого цикла выходная таблица образа процесса пересылается в память модулей (но еще не применена к внешним выходам, так как модули выходов все еще заблокированы).

Теперь у вас есть возможность задать при параметризации CPU очистку выходной таблицы образа процесса и памяти модулей в завершении «теплого» рестарта. Перед переключением к ОВ 1 CPU отменяет сигнал блокировки (Disable), так что сигнальные состояния из памяти модулей передаются во внешние выходы.

22.1.4 Организационные блоки рестарта

В случае «холодного» рестарта CPU вызывает организационный блок ОВ 102; при полном рестарте он вызывает организационный блок ОВ 100. В отсутствие ОВ 100 или ОВ 102 CPU начинает исполнение циклической программы немедленно.

При «теплом» рестарте CPU однократно вызывает организационный блок ОВ 101 перед обработкой главной программы. Если ОВ 101 нет, то CPU начинает сканирование с точки прерывания.

Стартовая информация во временных локальных данных имеет тот же формат для организационных блоков рестарта.

Таблица 22.1 содержит стартовую информацию для ОВ 100. Причина рестарта показана в запросе рестарта (байт 1):

- В#16#81 Полный рестарт вручную (ОВ 100);
- В#16#82 Автоматический полный рестарт (ОВ 100);
- В#16#83 «Теплый» рестарт вручную (ОВ 101);
- В#16#84 Автоматический «теплый» рестарт (ОВ 101);
- В#16#85 «Холодный» рестарт вручную (ОВ 102);
- В#16#86 Автоматический «холодный» рестарт (ОВ 102).

Номер события останова и дополнительная информация определяют рестарт более точно (сообщает вам, например, был ли полный рестарт вручную инициирован посредством переключателя режимов). Располагая этой информацией, вы можете создать соответствующую событийную процедуру рестарта.

Таблица 22.1 Стартовая информация для блоков ОВ рестарта

Байт	Наименование	Тип данных	Описание
0	OB100_EV_CLASS	BYTE	Класс событий
1	OB100_STRTUP	BYTE	Запрос рестарта (см. текст)
2	OB100_PRIORITY	BYTE	Приоритетный класс
3	OB100_OB_NUMBER	BYTE	Номер ОВ
4	OB100_RESERVED_1	BYTE	Зарезервировано
5	OB100_RESERVED_2	BYTE	Зарезервировано
6..7	OB100_STOP	WORD	Номер события останова
8..11	OB100_STRT_INFO	DWORD	Дополнительная информация по текущему рестарту
12..19	OB100_DATE_TIME	DT	Дата и время возникшего события

22.2 Включение электропитания

22.2.1 Режим STOP

CPU переходит в режим STOP в следующих случаях:

- При включении CPU;
- Когда переключатель режимов перемещен из положения RUN в положение STOP;
- При возникновении «неисправимой» ошибки во время сканирования программы;
- Когда выполняется системная функция SFC 46 STP;
- Когда останов запрашивается коммуникационной функцией (запрос останова от устройства программирования или через блоки коммуникационных функций от другого CPU).

CPU помещает причину перехода в режим STOP в диагностический буфер. В этом режиме вы также можете считать информацию CPU с использованием устройства программирования, чтобы локализовать проблему.

В состоянии STOP пользовательская программа не сканируется. CPU восстанавливает установки – либо значения, введенные вами в конфигурационные данные аппаратных средств при параметризации CPU, либо значения по умолчанию – и устанавливает модули в определенное начальное состояние.

В режиме STOP CPU может принимать глобальные данные через GD-коммуникацию и выполнять пассивные односторонние коммуникационные функции. Таймер реального времени продолжает работать.

Вы можете в режиме STOP параметризовать CPU, также можно, например, установить MPI-адрес, переслать или модифицировать пользовательскую программу и выполнить сброс памяти CPU.

22.2.2 Сброс памяти

Сброс памяти переводит CPU в «начальное состояние». Вы можете инициировать сброс памяти только с помощью программирующего устройства в режиме STOP или при помощи переключателя режимов: задержите переключатель в позиции MRES по крайней мере в течение 3 секунд, затем отпустите и через максимум 3 секунды снова переведите его в положение MRES на 3 секунды минимум.

CPU стирает всю пользовательскую программу и в рабочей памяти, и в загрузочной памяти RAM. Системная память (к примеру, меркерная память, таймеры и счетчики) также очищается независимо от сохраняющих установок.

CPU устанавливает параметры всех модулей, включая свои собственные, в их значения по умолчанию. Исключением являются MPI-параметры. Они не изменяются, поэтому CPU, чья память была сброшена, все еще может быть адресован на шине MPI. Сброс памяти также не оказывает влияния на диагностический буфер, таймер реального времени и таймеры времени исполнения.

Если вставлена флэш-карта памяти EPROM, CPU копирует пользовательскую программу с карты памяти в рабочую память. CPU также копирует все обнаруженные на карте памяти конфигурационные данные.

22.2.3 Реманентность

Область памяти является реманентной или сохраняемой (обладает способностью к запоминанию), когда ее содержимое сохраняется, даже когда отключено электропитание, а также при переходе CPU из состояния STOP к режиму RUN после включения питания (при «теплом» рестарте и «холодном» рестарте).

Сохраняемыми областями памяти могут быть меркерная память, таймеры, счетчики и в S7-300 области данных. Количество сохраняемых данных в таких областях зависит от версии CPU. Определить сохраняемые области вы можете на вкладке «Retentivity» в ходе параметризации CPU.

Установки для сохранения располагаются в системных блоках данных (system data blocks, SDB) в загрузочной памяти, то есть на карте памяти. Если карта памяти является RAM-картой, то программируемый контроллер должен быть оснащен резервной батареей, чтобы постоянно хранить сохраняемые установки.

При использовании резервной батареи определенные сохраняемыми сигнальные состояния меркеров, таймеров и счетчиков запоминаются. Программа пользователя и пользовательские данные остаются без изменений. Разница в том, используется карта памяти RAM или флэш-карта EPROM, отсутствует.

Если в качестве карты памяти используется флэш-карта EPROM и *резервной батареи нет*, то S7-300 и S7-400 реагируют по-разному. В контроллерах S7-300 сигнальные состояния сохраняемых меркеров, таймеров и счетчиков запоминаются, в контроллерах S7-400 такого не происходит.

В контроллерах S7-300 содержимое сохраняемых блоков также остается неизменным. Обратите внимание на то, что в S7-300 содержимое сохраняемых областей данных запоминается в CPU, а не на карте памяти.

Остальные блоки данных в контроллере S7-300 и все блоки данных в контроллере S7-400 копируются с карты памяти в рабочую память, как кодовые блоки. На карте памяти находятся те блоки данных, чье содержимое обладает способностью к запоминанию. Блоки данных, генерируемые с помощью системной функции SFC 22 CREAT_DB, не являются сохраняемыми. После рестарта блоки данных имеют содержимое, как на карте памяти, то есть содержимое, с которым они программировались.

22.2.4 Параметризация рестарта

На вкладке «Restart» («Рестарт») CPU вы можете настроить рестарт с помощью следующих установок:

- Рестарт при несовпадении установленной и фактической конфигураций
Рестарт выполняется, даже если конфигурация параметризованного аппаратного обеспечения не соответствует фактической конфигурации.
- Тестирование аппаратного обеспечения при полном рестарте («теплом» рестарте)
CPU S7-300 выполняют тест аппаратуры при включении электропитания.
- Удаление PIQ при «теплом» рестарте
CPU S7-400 удаляют все выходные таблицы образа процесса в случае «теплого» рестарта.
- Запрет «теплого» рестарта при ручном рестарте
«Теплый» рестарт вручную не допускается.
- Рестарт после POWER UP (Включения питания)
Определение типа рестарта после включения питания.
- Время наблюдения сигналов готовности модулей
Если время наблюдения модулей истекает, CPU остается в режиме STOP. Результат вводится в диагностический буфер (важно для включения питания в стойках расширения).
- Время наблюдения передачи параметров модулям
Если время наблюдения заканчивается, CPU остается в режиме STOP. Событие заносится в диагностический буфер. (В случае этой ошибки вы можете только параметризовать CPU с более продолжительным временем наблюдения – без сброса памяти – если вы пересылаете системные данные «пустого» проекта, в которых указано новое значение времени наблюдения, так что параметризация модуля завершается в течение «старого» времени наблюдения.)
- Время наблюдения «теплого» рестарта
Если время между отключением и включением питания или между режимами STOP и RUN больше времени наблюдения, CPU остается в режиме STOP. Если указать 0 мс, то наблюдение отключается.

22.3 Типы рестарта

22.3.1 Режим STURT-UP (Запуск)

CPU выполняет рестарт в следующих случаях:

- когда включается электропитание;
- когда переключатель режимов переводится из положения STOP в положения RUN или RUN-P;
- при запросе от коммуникационной функции (иницируется устройством программирования или через блоки коммуникационных функций другим CPU).

Ручной рестарт инициируется через переключатель или коммуникационную функцию, *автоматический* рестарт – при включении электропитания.

Процедура рестарта может продолжаться, сколько требуется, ограничения времени ее исполнения не накладывается; монитор цикла сканирования не активен.

Во время исполнения процедуры рестарта прерывания не обрабатываются. Исключениями являются ошибки, обрабатываемые как в режиме RUN (вызов соответствующих организационных блоков обслуживания ошибок).

В процедуре рестарта CPU обновляет таймеры, таймеры времени исполнения и таймер реального времени. Во время рестарта модули выходов заблокированы, то есть выходные сигналы не могут передаваться. Запрет выходов отменяется только в конце рестарта и до начала циклической программы.

Процедура рестарта может быть прервана, например, когда переключатель режимов поменял положение, или при сбое электропитания. Прерванная процедура перезапуска при включении питания выполняется с начала. Если прерван полный рестарт, он вновь должен быть выполнен. Если прерван «теплый» рестарт, допустимы все типы перезапуска.

Рисунок 22.2 показывает действия, выполняемые CPU во время рестарта.

22.3.2 «Холодный» рестарт

При «холодном» рестарте CPU переводит и самого себя, и модули в запрограммированное исходное состояние, удаляет все данные из системной памяти (включая сохраняемые данные), вызывает ОВ 102 и затем выполняет с начала главную программу в ОВ 1.

Текущая программа и текущие данные в рабочей памяти удаляются и вместе с ними также блоки данных, созданные с помощью SFC; программа, находящаяся в загрузочной памяти, перегружается. (В отличие от сброса памяти загрузочная память RAM не стирается).

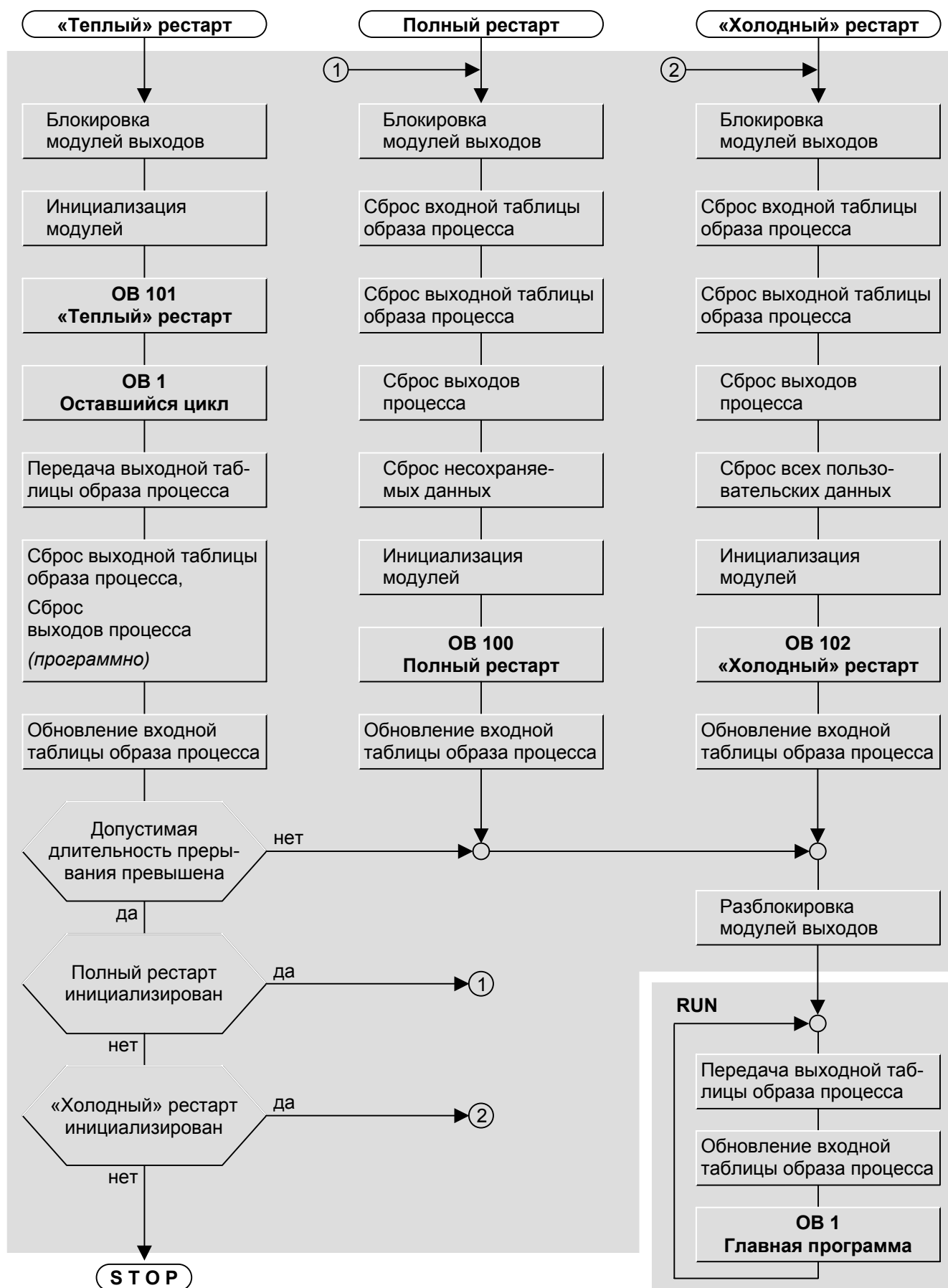


Рисунок 22.2 Действия CPU во время рестарта

Ручной «холодный» рестарт

«Холодный» рестарт реализуется вручную следующими способами:

- Посредством переключателя режимов на CPU, если переключатель был задержан в положении MRES в течение, по крайней мере, 3 секунд при переводе из режима STOP в режим RUN или RUN-P.
- Через коммуникационную функцию от PG или с использованием SFB от другого CPU; переключатель режимов должен быть в позиции RUN или RUN-P.

Ручной «холодный» рестарт всегда может быть инициирован, если CPU не запрашивает сброс памяти.

Автоматический «холодный» рестарт

Автоматический «холодный» рестарт инициируется при включении электропитания. «Холодный» рестарт выполняется, если

- CPU не находился в состоянии STOP, когда питание было выключено;
- переключатель режимов находится в положении RUN или RUN-P;
- CPU был прерван перебоем в электропитании при выполнении «холодного» рестарта;
- задана опция параметризации «Automatic cold restart on power up» («Автоматический «холодный» рестарт при включении питания»).

Когда работа производится без резервной батареи, CPU выполняет автоматический несохраняющий полный рестарт. CPU автоматически запускает сброс памяти, затем копирует пользовательскую программу с карты памяти в рабочую память. Карта памяти должна быть флэш-EPROM.

22.3.3 Полный рестарт

При полном рестарте CPU переводит самого себя и модули в запрограммированное исходное состояние, стирает все несохраняемые данные в системной памяти, вызывает OB 100 и затем выполняет главную программу в OB 1 с начала.

Полный ручной рестарт

Вручную полный рестарт реализуется следующими способами:

- Посредством переключателя режимов на CPU при переводе из режима STOP в режим RUN или RUN-P (в CPU S7-400 с переключателем типов рестарта выби-

рается положение CRST).

- Через коммуникационную функцию от PG или с использованием SFB от другого CPU; переключатель режимов должен быть в положении RUN или RUN-P.

Ручной полный рестарт всегда может быть инициирован, если CPU не запрашивает сброс памяти.

Полный автоматический рестарт

Автоматический полный рестарт инициируется при включении электропитания. Рестарт выполняется, если

- CPU не находился в состоянии STOP, когда питание было отключено;
- переключатель режимов находится в позиции RUN или RUN-P;
- CPU был прерван перебоем в электропитании при выполнении полного рестарта;
- задана опция параметризации «Automatic complete restart on power up» («Автоматический полный рестарт при включении питания»).

Когда работа производится без резервной батареи, CPU выполняет автоматический несохраняющий полный рестарт. CPU запускает сброс памяти автоматически, затем копирует пользовательскую программу с карты памяти в рабочую память. Карта памяти должна быть флэш-EPROM.

22.3.4 «Теплый» рестарт

«Теплый» рестарт возможен только на S7-400.

В режиме STOP или при отключении питания CPU сохраняет все прерывания, а также внутренние регистры CPU, являющиеся важными для обработки пользовательской программы. Таким образом, при «теплом» рестарте он может продолжить работу с точки, в которой произошло прерывание программы. Это может быть главная программа, процедура обработки прерывания или ошибки. Все («старые») прерывания сохраняются и будут обслужены.

Так называемый «оставшийся цикл», который продолжается с точки возобновления обработки программы CPU после «теплого» рестарта до конца главной программы, рассматривается как часть рестарта. Прерывания (новые) не обслуживаются. Модули выходов заблокированы и находятся в своих начальных состояниях.

«Теплый» рестарт допускается только тогда, когда не произошло изменений в пользовательской программе, к примеру, модификации блока, пока CPU находился в состоянии STOP.

При соответствующей параметризации CPU вы можете указать длительность прерывания для CPU с возможностью совершить «теплый» рестарт (от 100 миллисекунд до 1 часа). Если прерывание продолжительнее, то допустим только полный рестарт. Длительностью прерывания является количество времени между выходом из режима RUN (режим STOP или отключение питания) и повторным входом в режим RUN (после выполнения OB 101 и оставшегося цикла).

«Теплый» рестарт вручную

«Теплый» рестарт инициируется

- Если переключатель режимов был в положении RUN или RUN-P, когда CPU был включен переводом переключателя режимов из состояния STOP в один из этих режимов, при переключателе рестарта, находящемся в положении WRST (возможно только на CPU с переключателем типов рестарта);
- Через коммуникационную функцию от PG или с использованием SFB о другого CPU; переключатель режимов должен быть в позиции RUN или RUN-P.

Ручной «теплый» рестарт возможен, только если в ходе параметризации CPU на вкладке «Restart» («Рестарт») запрет «теплого» рестарта был отменен. Переход в состояние STOP должно производиться вручную либо посредством переключателя режимов, либо через коммуникационную функцию; только тогда «теплый» рестарт может быть выполнен вручную, пока CPU находится в режиме STOP.

Автоматический «теплый» рестарт

Автоматический «теплый» рестарт инициируется включением электропитания. CPU выполняет автоматический «теплый» рестарт только в следующих случаях:

- Если он при выключении не находился в режиме STOP;
- Если при выключении CPU переключатель режимов был в положении RUN или RUN-P;
- Если задана опция параметризации «Automatic warm restart on power up» («Автоматический «теплый» рестарт при включении питания»);
- Если вставлена резервная батарея, и она исправна.

Позиция переключателя рестарта не имеет значения для автоматического «теплого» рестарта.

22.4 Определение адреса модуля

Вы можете вычислить адрес модуля, используя следующие SFC:

- SFC 5 GADR_LGC
Определение логического адреса канала модуля;
- SFC 50 RD_LGADR
Определение всех логических адресов модуля;
- SFC 49 LGC_GADR
Определение адреса слота модуля.

В таблице 22.2 показаны параметры этих SFC.

SFC имеют общие параметры IOID и LADDR для логического адреса (= адресу в области входов/выходов). IOID равен либо B#16#54, что обозначает периферийные входы (PI), либо B#16#55, что указывает на периферийные выходы (PQ). LADDR содержит адрес входа/выхода (I/O-адрес) в области PI или PQ, который соответствует определенному каналу. Если канал 0, то это стартовый адрес модуля.

Конфигурационные данные аппаратных средств должны определять местоположение между логическим адресом (стартовым адресом модуля) и адресом слота (положение модуля в стойке или станции для распределенного входа/выхода) для адресов, определенных с помощью этих SFC.

SFC 5 GADR_LGC

Определение логического адреса канала модуля

Системная функция SFC 5 GADR_LGC возвращает логический адрес канала, когда вы указываете адрес слота («географический» адрес). Введите в параметр SUBNETID номер подсети, если модуль принадлежит распределенному входу/выходу, или B#16#00, если модуль вставлен в стойку. Параметр RACK определяет номер стойки или, в случае распределенного входа/выхода, номер станции. Если у модуля нет подмодульного слота, введите в параметр SUBSLOT B#16#00. SUBADDR содержит смещение адреса в пользовательских данных модуля (W#16#0000, например, обозначает стартовый адрес модуля).

SFC 49 LGC_GADR

Определение адреса слота модуля

SFC 49 LGC_GADR возвращает адрес слота модуля, когда вы указываете произвольный логический адрес модуля. Вычитание смещения адреса (параметр SUBADDR) из определенного адреса пользовательских данных дает вам стартовый адрес модуля. Значение в параметре AREA определяет систему, в которой модуль обрабатывается (таблица 22.3).

Таблица 22.2 Параметры SFC, используемых для определения адреса модуля

SFC	Параметр	Объявление	Тип данных	Содержимое, описание
5	SUBNETID	INPUT	BYTE	Идентификатор области
	RACK	INPUT	WORD	Номер стойки
	SLOT	INPUT	WORD	Номер слота
	SUBSLOT	INPUT	BYTE	Номер подмодуля
	SUBADDR	INPUT	WORD	Смещение в области адресов пользовательских данных модуля
	RET_VAL	OUTPUT	INT	Информация об ошибке
	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 ошибке
	PEADDR	OUTPUT	ANY	Поле WORD для PI-адресов
	PECOUNT	OUTPUT	INT	Количество возвращенных PI-адресов
	PAADDR	OUTPUT	ANY	Поле WORD для PQ-адресов
	PACOUNT	OUTPUT	INT	Количество возвращенных PQ-адресов
49	IOID	INPUT	BYTE	Идентификатор области
	LADDR	INPUT	WORD	Логический адрес модуля
	RET_VAL	OUTPUT	INT	Информация об ошибке
	AREA	OUTPUT	BYTE	Идентификатор области
	RACK	OUTPUT	WORD	Номер стойки
	SLOT	OUTPUT	WORD	Номер слота
	SUBADDR	OUTPUT	WORD	Смещение в области адресов пользовательских данных модуля

SFC 50 RD_LGADR**Определение всех логических адресов модуля**

В S7-400 вы можете назначить адреса для не являющихся смежными байтов пользовательских данных модуля (при создании).

SFC 50 RD_LGADR возвращает все логические адреса для модуля, когда вы указываете произвольный адрес из области пользовательских данных.

Параметры PEADDR и PAADDR предназначены для определения области WORD-компонентов (указателя ANY на базе слова, например, P#DBzDBXu..x WORD nnn).

SFC 50 в таком случае показывает количество возвращаемых вхождений в эти области в параметрах PECOUNT и PACOUNT.

Таблица 22.3 Описание выходных параметров SFC 49 LGC_GADR

AREA	Система	Значение RACK, SLOT и SUBADDR
0	S7-400	RACK = Номер стойки SLOT = Номер слота SUBADDR = Смещение адреса к стартовому адресу
1	S7-300	
2	Распределенный вход/выход (I/O)	
3	Область S5 P	RACK, SLOT и SUBADDR несущественны
4	Область S5 Q	RACK = Номер стойки SLOT = Номер слота кожуха адаптера SUBADDR = Адрес в области S5
5	Область S5 IM3	
6	Область S5 IM4	

22.5 Параметризация модулей

Для параметризации модулей доступны следующие системные функции:

- SFC 54 RD_DPARM
Чтение предопределенных параметров;
- SFC 55 WR_PARM
Запись динамических параметров;
- SFC 56 WR_DPARM
Запись предопределенных параметров;
- SFC 57 PARM_MOD
Параметризация модуля;
- SFC 58 WR_REC
Запись записи (элемента, data record) данных;
- SFC 59 RD_REC
Чтение записи (элемента, data record) данных.

Параметры этих и некоторых других системных функций показаны в таблице 22.4.

Таблица 22.4 Параметры системных функций, используемых для передачи данных

Параметр SFC						Параметр	Объявление	Тип данных	Содержимое, описание
54	55	56	57	58	59	REQ	INPUT	BOOL	«1» = Запрос (операции) записи
54	55	56	57	58	59	IOID	INPUT	BYTE	V#16#54 = Периферийные входы (PI) V#16#55 = Периферийные выходы (PQ)
54	55	56	57	58	59	LADDR	INPUT	WORD	Стартовый адрес модуля
54	55	56	-	58	59	RECNUM	INPUT	BYTE	Номер записи (элемента) данных
54	55	-	-	58	-	RECORD	INPUT	ANY	Запись (элемент) данных
54	55	56	57	58	59	RET_VAL	OUTPUT	INT	Информация об ошибке
54	55	56	57	58	59	BUSY	OUTPUT	BOOL	Если «1», то передача еще выполняется
-	-	-	-	-	59	RECORD	OUTPUT	ANY	Запись (элемент) данных

С помощью вышеупомянутых системных функций могут быть переданы следующие записи данных:

Номер записи	Содержимое для чтения	Содержимое для записи
0	Диагностические данные	Параметры
1	Диагностические данные	Параметры
от 2 до 127	Пользовательские данные	Пользовательские данные
от 128 до 255	Диагностические данные	Параметры

Общие замечания по параметризации модулей

Некоторые S7-модули могут быть параметризованы, то есть в модуле можно установить значения, отличные от значений по умолчанию. Чтобы задать параметры, откройте модуль в утилите конфигурирования аппаратных средств и заполните вкладки диалогового окна. Когда вы передаете объект *System Data* (*Системные данные*) в контейнере *Blocks* (*Блоки*) в PLC, вы также пересылаете параметры модуля.

CPU автоматически передает в модуль его параметры в следующих случаях:

- При рестарте;
- Когда модуль был вставлен в конфигурированный слот (S7-400);
- После «ответа» («return») стойки или станции распределенного входа/выхода.

Параметры модулей делятся на статические и динамические параметры. Параметры обоих типов могут устанавливаться автономно в утилите Hardware Configuration. Вы также можете модифицировать динамические параметры во время исполнения программы с помощью вызовов SFC. В процедуре запуска параметры, установленные в модулях при помощи SFC, переписываются параметрами, установленными (и сохраненными в CPU) посредством Hardware Configuration.

Параметры для сигнальных модулей находятся в двух записях данных: статические параметры в записи данных 0, динамические параметры в записи данных 1. Вы можете передать в модуль обе записи данных с помощью SFC 57 PARM_MOD, запись данных 0 или 1 с использованием SFC 56 WR_DPARM и только запись данных 1 при помощи SFC 55 WR_PARM. Записи данных должны находиться в системных блоках данных в CPU.

После параметризации модуля S7-400 заданные значения не вступают в силу до тех пор, пока бит 2 («Operating mode», «Рабочий режим») в байте 2 записи диагностических данных 0 не примет значение «RUN» (может быть прочитан с помощью SFC 59 RD_REC).

Поскольку рассматривается адресация для передачи данных, используйте *наименьший* стартовый адрес модуля (параметр LADDR) совместно с идентификатором, показывающим, определен этот адрес как вход или как выход (параметр IOID). Если вы назначили одинаковый стартовый адрес для областей входов и выходов, то используйте идентификатор для входа. Используйте идентификатор входа/выхода вне зависимости от того, какую операцию вы хотите выполнить, чтение (Read) или запись (Write).

Параметр RECORD типа данных ANY применяется для определения области компонентов BYTE. Это может быть переменная типа ARRAY, STRUCT или UDT (тип, определяемый пользователем), или ANY-указатель типа BYTE (например, P#DBzDBXy.x BYTE nnn). Если вы используете переменную, то она должна быть «полностью» переменной; отдельные компоненты массива или структуры недопустимы.

SFC 54 RD_DPARM

Чтение предопределенных параметров

Системная функция SFC 54 RD_DPARM передает запись данных с номером, указанным в параметре RECNUM, из соответствующего системного блока данных SDB в область назначения, определенную в параметре RECORD.

Теперь вы можете, к примеру, модифицировать эту запись данных и записать ее в модуль при помощи функции SFC 58 WR_REC.

SFC 55 WR_PARM

Запись динамических параметров

Системная функция SFC 55 WR_PARM пересылает запись данных, адресованную параметром RECORD, в модуль, определенный параметрами IOID и LADDR. В параметре RECNUM указывается номер записи данных. Необходимо, чтобы запись данных находилась в корректном системном блоке данных SDB, и она должна содержать только динамические параметры.

При инициировании задания SFC считывает всю запись данных; передача может быть распределена на несколько циклов сканирования программы. Во время передачи параметр BUSY установлен в «1».

SFC 56 WR_DPARM

Запись предопределенных параметров

Системная функция SFC 56 WR_DPARM передает запись данных с номером, указанным в параметре RECNUM, из соответствующего системного блока данных SDB в модуль, идентифицированный параметрами IOID и LADDR.

Передача может быть распределена на несколько циклов сканирования программы; во время передачи параметр BUSY установлен в «1».

SFC 57 PARM_MOD

Параметризация модуля

Системная функция SFC 57 PARM_MOD пересылает все записи данных, запрограммированные при параметризации модуля в Hardware Configuration.

Передача может быть распределена на несколько циклов сканирования программы; во время передачи параметр BUSY установлен в «1».

SFC 58 WR_REC

Запись записи данных

SFC 58 WR_REC передает запись данных, адресованную параметром RECORD и номером RECNUM, в модуль, задаваемый параметрами IOID и LADDR. «1» в параметре REQ запускает пересылку. При инициировании операции SFC считывает запись данных полностью.

Передача может быть осуществлена за несколько циклов сканирования программы; во время передачи параметр BUSY установлен в «1».

SFC 59 RD_REC

Чтение записи данных

Когда параметр REQ равен «1», SFC 59 RD_REC считывает запись данных, адресованную параметром RECNUM, из модуля и помещает ее в область назначения RECORD. Область назначения должна быть больше или, по крайней мере, равной по размеру записи данных. Если передача завершена без ошибок, параметр RET_VAL содержит число переданных байтов.

Передача может быть распределена на несколько циклов сканирования программы; во время передачи параметр BUSY установлен в «1».

S7-300, выпущенные до февраля 1997 г: SFC считывает из определенной записи данных столько данных, сколько область назначения может вместить. Размер области назначения не может превышать размер записи данных.

Содержание главы 23

<u>23</u>	<u>Обработка ошибок</u>	4
<u>23.1</u>	<u>Синхронные ошибки</u>	4
<u>23.2</u>	<u>Обработка синхронных ошибок</u>	7
<u>23.2.1</u>	<u>Фильтры ошибок</u>	7
<u>23.2.2</u>	<u>Маскирование синхронных ошибок</u>	9
<u>23.2.3</u>	<u>Демаскирование синхронных ошибок</u>	10
<u>23.2.4</u>	<u>Чтение регистра ошибок</u>	10
<u>23.2.5</u>	<u>Ввод заменяющего значения</u>	10
<u>23.3</u>	<u>Асинхронные ошибки</u>	12
<u>23.4</u>	<u>Системная диагностика</u>	17
<u>23.4.1</u>	<u>Диагностические события и диагностический буфер</u>	17
<u>23.4.2</u>	<u>Запись пользовательского ввода в диагностический буфер</u>	18
<u>23.4.3</u>	<u>Оценка диагностических прерываний</u>	19
<u>23.4.4</u>	<u>Чтение списка состояний системы</u>	19

23 Обработка ошибок

CPU сообщает об ошибках или сбоях, обнаруженных модулями или самим CPU, различными способами:

- В случае ошибок в арифметических операциях (переполнение, недействительное число REAL) устанавливаются биты состояния (к примеру, бит состояния OV служит для индикации числового переполнения);
- При возникновении ошибок во время исполнения программы пользователя (синхронных ошибок) вызываются организационные блоки OB 121 и OB 122;
- Ошибки в программируемом контроллере, которые не влияют на программное сканирование, (асинхронные ошибки) приводят к вызову организационных блоков с OB 80 по OB 87.

CPU сигнализирует о возникновении ошибки или сбое и в некоторых случаях о причине, включая светодиоды на передней панели. В случае критических (неисправимых) ошибок (таких как неверный код OP) CPU сразу переходит в состояние STOP.

В то время как CPU находится в режиме STOP, вы можете использовать программирующее устройство и функции, работающие с информацией CPU, чтобы прочитать содержимое стека блоков (B-стек), стека прерываний (I-стека) и стека локальных данных (L-стека) и затем составить заключение о причине ошибки.

Системная диагностика может обнаруживать ошибки/сбои в модулях и помещать сообщения об этих ошибках в диагностический буфер. Информация о смене режимов CPU (например, причины перехода в режим STOP) также вводится в диагностический буфер.

Содержимое этого буфера сохраняется при переходе в состояние STOP, сбросе памяти и сбое в электропитании и после его восстановления и выполнения процедуры запуска может быть считано с использованием устройства программирования.

В новых CPU вы можете использовать параметризацию CPU для установки количества сохраняемых записей диагностического буфера.

23.1 Синхронные ошибки

Операционная система CPU генерирует синхронную ошибку, когда возникает ошибка, непосредственно связанная со сканированием программы. Если OB синхронной ошибки не запрограммирован, то CPU переходит в режим STOP. Различаются два типа ошибок:

- **Программная ошибка**, вызывается OB 121 и
- **Ошибка доступа**, вызывается OB 122.

В таблице представлена стартовая информация для обоих организационных блоков обработки синхронных ошибок.

Таблица 23.1 Стартовая информация для блоков ОВ синхронных ошибок

Имя переменной	Тип данных	Описание, содержимое																
OB12x_EV_CLASS	BYTE	V#16#25 = Вызов ОВ 121 программной ошибки V#16#29 = Вызов ОВ 122 ошибки доступа																
OB12x_SW_FLT	BYTE	Код ошибки (см. параграф 23.2.1 «Фильтры ошибок»)																
OB12x_PRIORITY	BYTE	Приоритетный класс, в котором возникла ошибка																
OB12x_OB_NUMBER	BYTE	Номер ОВ (V#16#79 или V#16#80)																
OB12x_BLK_TYPE	BYTE	Тип прерванного блока (только в S7-400) OB: V#16#88, FB: V#16#8E, FC: V#16#8C																
OB121_RESERVED_1 OB122_MEM_AREA	BYTE	Назначения байтов (V#15#xy): 7... (x) ...4 3... (y) ...0																
		<table border="0"> <tr> <td>0</td> <td>Область входа/выхода (I/O) PI или PQ</td> </tr> <tr> <td>1</td> <td>Входная таблица образа процесса I</td> </tr> <tr> <td>2</td> <td>Выходная таблица образа процесса Q</td> </tr> <tr> <td>3</td> <td>Память меркеров M</td> </tr> <tr> <td>4</td> <td>Глобальный блок данных DB</td> </tr> <tr> <td>5</td> <td>Экземплярный блок данных DI</td> </tr> <tr> <td>6</td> <td>Временные локальные данные L</td> </tr> <tr> <td>7</td> <td>Временные локальные данные вызывающего блока V</td> </tr> </table>	0	Область входа/выхода (I/O) PI или PQ	1	Входная таблица образа процесса I	2	Выходная таблица образа процесса Q	3	Память меркеров M	4	Глобальный блок данных DB	5	Экземплярный блок данных DI	6	Временные локальные данные L	7	Временные локальные данные вызывающего блока V
0	Область входа/выхода (I/O) PI или PQ																	
1	Входная таблица образа процесса I																	
2	Выходная таблица образа процесса Q																	
3	Память меркеров M																	
4	Глобальный блок данных DB																	
5	Экземплярный блок данных DI																	
6	Временные локальные данные L																	
7	Временные локальные данные вызывающего блока V																	
OB121_FLT_REG OB122_MEM_ADDR	WORD	OB 121: Источник ошибки Ошибочный адрес (при доступе чтения/записи) Ошибочная область (в случае ошибки области) Некорректный номер блока, функция таймера/счетчика OB 122: Адрес, по которому возникла ошибка																
OB12x_BLK_NUM	WORD	Номер блока, в котором возникла ошибка (только в S7-400)																
OB12x_PRG_ADDR	WORD	Адрес ошибки в блоке, вызвавшем ошибку (только в S7-400)																
OB12x_DATE_TIME	DT	Время обнаружения программной ошибки																

ОВ синхронной ошибки имеет тот же приоритет, что и блок, в котором возникла ошибка. При таком условии будет возможен доступ к регистрам прерванного блока в ОВ синхронной ошибки, а также программа в ОВ синхронной ошибки (в определенных обстоятельствах с модифицированным содержимым) сможет возвращать регистры в прерванный блок.

Заметьте, что когда вызывается ОВ синхронной ошибки, 20 байтов стартовой информации этого блока также помещаются в L-стек приоритетного класса, вызвавшего ошибку, как и другие временные локальные данные ОВ синхронной ошибки и всех блоков, вызываемых в этом ОВ.

В случае S7-400 в ОВ обработки ошибки может быть вызван другой ОВ синхронной ошибки. Глубина вложения блоков для ОВ синхронной ошибки составляет 3 уровня для CPU S7-400 и 4 для CPU S7-300.

Можно заблокировать и разрешить ОВ обслуживания синхронной ошибки с помощью системных функций SFC 36 MSK_FLT и SFC 37 DMSK_FLT.

23.2 Обработка синхронных ошибок

Для обслуживания синхронных ошибок предоставляются следующие системные функции:

- SFC 36 MSK_FLT
Маскировка синхронные ошибки (запрещает вызов OB);
- SFC 37 DMSK_FLT
Демаскировка синхронные ошибки (возобновляет вызов OB);
- SFC 38 READ_ERR
Чтение регистра ошибки (error register).

Операционная система вводит синхронные ошибки в диагностический буфер независимо от использования системных функций SFC 36, 37 и 38. Параметры этих функций приведены в таблице 23.2.

23.2.1 Фильтры ошибок

Фильтры ошибок используются для управления системными функциями обработки синхронных ошибок. В фильтре программных ошибок для каждой обнаруженной программной ошибки отведен обозначающий ее один бит; в фильтре ошибок доступа также для обозначения каждой возникшей ошибки доступа предназначен один бит. При определении фильтра ошибок вы устанавливаете бит, обозначающий синхронную ошибку, которую вы хотите маскировать, демаскировать или запросить по ней информацию. Фильтры ошибок, возвращаемые системными функциями, показывают «1» для синхронных ошибок, которые все еще маскированы, или которые возникли.

Фильтр ошибок доступа показан в таблице 23.3; столбец кода ошибки отображает содержимое переменной OB122_SW_FLT в стартовой информации OB 122. Фильтр программных ошибок представлен в таблице 23.4; столбец кода ошибки демонстрирует содержимое переменной OB121_SW_FLT в стартовой информации OB 121.

CPU S7-400 различает два типа ошибок доступа: обращение к несуществующему модулю и неверная попытка доступа к существующему модулю. Если во время операции происходит сбой модуля, данный модуль маркируется как «несуществующий» и остается таким примерно 150 с после попытки обращения, и при каждой последующей попытке обращения к модулю сообщается об ошибке доступа к входу/выходу (PZF). CPU также сообщает об ошибке доступа к входу/выходу (I/O access error), когда сделана попытка обращения к несуществующему модулю, независимо от того, была ли попытка прямого доступа (через область входов/выходов) или косвенного (через образ процесса).

Биты фильтров ошибок, не указанные в таблицах, не относятся к обработке синхронных ошибок.

Таблица 23.2 Параметры SFC обработки синхронных ошибок

SFC	Название параметра	Объявление	Тип данных	Содержимое, описание
36	PRGFLT_SET_MASK	INPUT	DWORD	Новый (дополнительный) фильтр программных ошибок
	ACCFLT_SET_MASK	INPUT	DWORD	Новый (дополнительный) фильтр ошибок доступа
	RET_VAL	OUTPUT	INT	W#16#0001 = Новый фильтр перекрывается с существующим фильтром
	PRGFLT_MASKED	OUTPUT	DWORD	Полный фильтр программных ошибок
	ACCFLT_MASKED	OUTPUT	DWORD	Полный фильтр ошибок доступа
37	PRGFLT_RESET_MASK	INPUT	DWORD	Фильтр программных ошибок, предназначенный для сброса
	ACCFLT_RESET_MASK	INPUT	DWORD	Фильтр ошибок доступа, предназначенный для сброса
	RET_VAL	OUTPUT	INT	W#16#0001 = Новый фильтр содержит неустановленные биты (в текущем фильтре)
	PRGFLT_MASKED	OUTPUT	DWORD	Оставшийся фильтр программных ошибок
	ACCFLT_MASKED	OUTPUT	DWORD	Оставшийся фильтр ошибок доступа
38	PRGFLT_QUERY	INPUT	DWORD	Фильтр программных ошибок, предназначенный для опроса
	ACCFLT_QUERY	INPUT	DWORD	Фильтр ошибок доступа, предназначенный для опроса
	RET_VAL	OUTPUT	INT	W#16#0001 = Запрашиваемый фильтр содержит неустановленные биты (в текущем фильтре)
	PRGFLT_CLR	OUTPUT	DWORD	Фильтр программных ошибок с сообщениями об ошибках
	ACCFLT_CLR	OUTPUT	DWORD	Фильтр ошибок доступа с сообщениями об ошибках

Таблица 23.3 Фильтр ошибок доступа

Бит	Код ошибки	Содержимое
3	В#16#42	Ошибка доступа к входу/выходу при чтении S7-300: Модуль не существует или не подтверждает S7-400: Существующий модуль не подтверждает после первой операции доступа (таймаут)
4	В#16#43	Ошибка доступа к входу/выходу при записи S7-300: Модуль не существует или не подтверждает S7-400: Существующий модуль не подтверждает после первой операции доступа (таймаут)
5	В#16#44	Только в S7-400: Ошибка доступа к входу/выходу при попытке прочитать несуществующие модули (PZF) или (в случае старших версий) повторном обращении к модулю, который не подтверждает
6	В#16#45	Только в S7-400: Ошибка доступа к входу/выходу при попытке записи в несуществующие модули (PZF) или (в случае старших версий) повторном обращении к модулю, который не подтверждает

Таблица 23.4 Фильтр программных ошибок

Бит	Код ошибки	Содержимое
1	V#16#21	Ошибка преобразования VCD (во время преобразования обнаружена псевдотетрада)
2	V#16#22	Ошибка размера области при чтении (адрес за пределами области)
3	V#16#23	Ошибка размера области при записи (адрес за пределами области)
4	V#16#24	Ошибка размера области при чтении (неверная область в указателе области)
5	V#16#25	Ошибка размера области при записи (неверная область в указателе области)
6	V#16#26	Неправильный номер таймера
7	V#16#27	Неправильный номер счетчика
8	V#16#28	Ошибка адреса при чтении (адрес бита $< > 0$ совместно с доступом к байту, слову или двойному слову и косвенной адресацией)
9	V#16#29	Ошибка адреса при записи (адрес бита $< > 0$ совместно с доступом к байту, слову или двойному слову и косвенной адресацией)
16	V#16#30	Ошибка записи, глобальный блок данных (блок, защищенный от записи)
17	V#16#31	Ошибка записи, экземплярный блок данных (блок, защищенный от записи)
18	V#16#32	Неправильный номер глобального блока данных (регистр DB)
19	V#16#33	Неправильный номер экземплярного блока данных (регистр DI)
20	V#16#34	Неправильный номер функции (FC)
21	V#16#35	Неправильный номер функционального блока (FB)
26	V#16#3A	Вызываемый блок данных (DB) не существует
28	V#16#3C	Вызываемая функция (FC) не существует
30	V#16#3E	Вызываемый функциональный блок (FB) не существует

23.2.2 Маскирование синхронных ошибок

Системная функция **SFC 36 MSK_FLT** запрещает посредством фильтров ошибок вызовы ОВ синхронных ошибок. «1» в фильтрах ошибок указывает на синхронную ошибку, для которой блоки ОВ не должны вызываться (синхронные ошибки «маскированы»).

Маскирование синхронных ошибок в фильтрах ошибок дополняет маскирование, сохраненное в памяти операционной системы. SFC 36 возвращает значение функции, отображающее, существует ли уже (сохраненное) маскирование по крайней мере одного бита при маскировании, определенном во входных параметрах (W#16#0001).

SFC 36 возвращает «1» в выходных параметрах для всех маскированных в текущий момент ошибок.

Если возникает маскированная синхронная ошибка, то соответствующий ОВ не вызывается, и ошибка записывается в регистр ошибок. Блокировка применяется к текущему приоритетному классу (приоритетному уровню). Например, если вы запре-

тили вызов ОВ синхронной ошибки в главной программе, то ОВ синхронной ошибки все же вызывается, если ошибка возникла в процедуре обслуживания прерывания.

23.2.3 Демаскирование синхронных ошибок

Системная функция **SFC 37 DMSK_FLT** разрешает вызовы ОВ синхронных ошибок посредством фильтров ошибок. Вы должны ввести в фильтры «1», чтобы обозначить синхронные ошибки, ОВ которых вновь могут быть вызваны (синхронные ошибки «демаскированы»). Записи, соответствующие заданным битам, удаляются в регистре ошибок. SFC 37 возвращает W#16#0001 в качестве значения функции, если не существует (сохраненное) маскирование хотя бы одного бита при демаскировании, определенном во входных параметрах.

SFC 37 возвращает «1» в выходных параметрах для всех маскированных в текущий момент ошибок.

Если возникает немаскированная синхронная ошибка, то вызывается соответствующий ОВ, и событие вводится в регистр ошибок. Разблокировка применяется к текущему приоритетному классу (приоритетному уровню).

23.2.4 Чтение регистра ошибок

Системная функция **SFC 38 READ_ERR** считывает регистр ошибок. Вы должны ввести «1» в фильтры ошибок, чтобы обозначить синхронные ошибки, записи которых вы хотите прочитать. SFC 38 возвращает W#16#0001 в качестве значения функции, когда выбор, заданный во входных параметрах, содержит по крайней мере один бит, для которого нет (сохраненного) маскирования.

SFC 38 возвращает «1» в выходных параметрах для выбранных ошибок, когда эти ошибки возникают, и удаляет эти ошибки в регистре ошибок, когда они запрашиваются. Синхронные ошибки, о которых сообщается, находятся в текущем приоритетном классе (приоритетном уровне).

23.2.5 Ввод заменяющего значения

SFC 44 REPL_VAL позволяет вам ввести значение замены в аккумулятор 1 из ОВ синхронной ошибки. SFC 44 используется, когда больше нельзя считывать любые значения из модуля (к примеру, когда модуль неисправен). Когда программируется SFC 44, блок ОВ 122 («ошибка доступа») вызывается каждый раз при попытке осуществить доступ к заданному модулю. Когда вы вызываете SFC 44, в аккумулятор можно загрузить заменяющее значение; в таком случае сканирование программы продолжается со значением замены. Параметры SFC 44 содержатся в таблице 23.5.

SFC 44 можно вызвать только в одном ОВ синхронной ошибки (ОВ 121 или ОВ 122).

Таблица 23.5 Параметры SFC 44 REPL_VAL

SFC	Имя параметра	Объявление	Тип данных	Содержимое, описание
44	VAL	INPUT	DWORD	Значение замены
	RET_VAL	OUTPUT	INT	Информация об ошибке

23.3 Асинхронные ошибки

Асинхронные ошибки – это ошибки, которые могут возникнуть независимо от сканирования программы. При появлении асинхронной ошибки операционная система вызывает один из организационных блоков, приведенных ниже:

- ОВ 80 Временная ошибка (Timing error);
- ОВ 81 Сбой в подаче электропитания (Power supply error);
- ОВ 82 Диагностическое прерывание (Diagnostic interrupt);
- ОВ 83 Прерывание вставки/удаления модуля (Insert/remove module interrupt);
- ОВ 84 Сбой в оборудовании CPU (CPU hardware fault);
- ОВ 85 Ошибка исполнения программы (Program execution error);
- ОВ 86 Сбой стойки (Rack failure);
- ОВ 87 Коммуникационная ошибка (Communication error).

Вызов ОВ 82 (диагностическое прерывание) подробно описывается в параграфе 23.4 «Системная диагностика».

В S7-400H имеется три дополнительных блока ОВ обработки асинхронных ошибок:

- ОВ 70 Ошибки резервирования входов/выходов (I/O redundancy errors);
- ОВ 71 Ошибки резервирования CPU (CPU redundancy errors);
- ОВ 72 Ошибки резервирования коммуникаций (Communications redundancy errors);

Вызов этих организационных блоков обработки асинхронных ошибок может быть запрещен и разблокирован с помощью системных функций SFC 39 DIS_IRT и SFC 40 EN_IRT, а также задержан и разрешен при помощи системных функций SFC 41 DIS_AIRT и SFC 42 EN_AIRT.

Временные ошибки

Операционная система вызывает организационный блок ОВ 80 при возникновении одной из следующих ошибок:

- Время наблюдения цикла (максимальное время цикла) превышено;

- Ошибка запроса ОВ (запрашиваемый ОВ еще выполняется, или ОВ запрашивался слишком часто в данном приоритетном классе);
- Ошибка прерывания по времени суток (прерывание по времени суток прошло, поскольку часы установлены вперед или после перехода в режим RUN).

Если ОВ 80 недоступен, а временная ошибка произошла, то CPU переходит в состояние STOP. CPU также переходит в состояние STOP, если ОВ вызывается во второй раз в одном программном цикле.

Ошибки в подаче электропитания

Операционная система вызывает организационный блок ОВ 81, если возникает одна из следующих ошибок:

- По крайней мере, одна резервная батарея центрального контроллера или блока расширения разряжена;
- Нет напряжения батареи центрального контроллера или блока расширения;
- Сбой питания 24 В в центральном контроллере или блоке расширения.

ОВ 81 вызывается для приходящих и уходящих событий. Если ОВ 81 отсутствует, то в случае возникновения сбоев электропитания CPU продолжает функционирование.

Прерывание вставки/удаления модуля

Операционная система осуществляет наблюдение конфигурации модулей один раз в секунду. Каждый раз при вставке или удалении модуля в режимах RUN, STOP или START-UP вносится запись в диагностический буфер и в список состояния системы.

Кроме того, операционная система вызывает организационный блок ОВ 83, если CPU находится в режиме RUN. Если ОВ 83 нет, то по прерыванию вставки/удаления модуля CPU переходит в режим STOP.

Может пройти одна секунда перед генерированием прерывания вставки/удаления модуля. В результате имеется возможность сообщения об ошибке доступа или ошибке, относящейся к обновлению образа процесса, в промежутке между удалением модуля и генерированием прерывания.

Если в сконфигурированный слот вставляется подходящий модуль, CPU автоматически осуществляет параметризацию этого модуля, используя записи данных, уже сохраненные в этом CPU. Только затем вызывается ОВ 83, чтобы сигнализировать о готовности к работе подключенного модуля.

Сбои оборудования CPU

Операционная система вызывает организационный блок OB 84, когда возникает или исчезает ошибка интерфейса (сети MPI, PROFIBUS DP). Если OB 84 отсутствует, CPU при сбое оборудования переходит в режим STOP.

Ошибки исполнения программы

Операционная система вызывает организационный блок OB 85, когда происходит одна из следующих ошибок:

- Запрос запуска организационного блока, который не был загружен;
- Возникла ошибка, пока операционная система обращалась к блоку (например, отсутствует экземплярный блок данных, когда был вызван системный функциональный блок (SFB));
- Ошибка доступа к входу/выходу во время исполнения (автоматического) обновления образа процесса на системной стороне.

В CPU S7-400 и CPU 318 OB 85 вызывается при каждой ошибке доступа к входу/выходу (на системной стороне), то есть при обновлении образа процесса в каждом цикле. В таком случае при каждом обновлении в соответствующий байт входной таблицы образа процесса вводится значение замены или ноль.

В CPU S7-300 (кроме CPU 318) при возникновении ошибки доступа к входу/выходу во время автоматического обновления таблицы образа процесса OB 85 не вызывается. При первом ошибочном обращении в соответствующий байт заносится заменяющее значение или ноль; больше он не обновляется.

При наличии должным образом оборудованного CPU вы можете использовать параметризацию CPU для воздействия на режим вызова OB 85 в случае ошибки доступа к входу/выходу на системной стороне:

- OB 85 вызывается каждый раз. Изменяемый входной байт переписывается заменяющим значением или нулем каждый раз.
- OB 85 вызывается в случае первой ошибки с атрибутом «приходящая» («incoming»). Изменяемый байт только один раз переписывается заменяющим значением или нулем; после этого он больше не обновляется. Если ошибка исправлена, OB 85 вызывается с атрибутом «уходящая» («outgoing»); после этого обновление байта происходит «нормально».
- OB 85 не вызывается при возникновении ошибки доступа. Изменяемые входные байты переписываются заменяющим значением или нулем один раз и больше не обновляются.

Если OB 85 нет, при ошибке исполнения программы CPU переходит в режим STOP.

Сбой стойки

При обнаружении сбоя стойки (перебои в электропитании, разрыв линии, неисправный ИМ), подсети или распределенной станции входов/выходов операционная система вызывает организационный блок ОВ 86. Этот ОВ вызывается для проходящих и уходящих ошибок.

В мультипроцессорном режиме ОВ 86 вызывается во всех CPU при сбоях стойки.

Если ОВ 86 отсутствует, в случае сбоя стойки CPU переходит в состояние STOP.

Коммуникационная ошибка

Операционная система в случае возникновения коммуникационной ошибки вызывает организационный блок ОВ 87. Примерами коммуникационных ошибок являются:

- Неверная идентификация фрейма или размер фрейма, обнаруженные при коммуникации глобальных данных;
- Отправка диагностических записей невозможна;
- Ошибка синхронизации таймера;
- Состояние GD не может быть введено в блок данных.

Если ОВ 87 нет, CPU переходит в состояние STOP, когда возникает коммуникационная ошибка.

Ошибка резервирования входа/выхода

Операционная система CPU серии Н вызывает организационный блок ОВ 70 при потере резервирования в PROFIBUS DP, например, в случае сбоя шины в активном DP-мастере или отказа в интерфейсе DP-ведомого.

Если ОВ 70 не существует, CPU в случае ошибки резервирования входа/выхода продолжает работать.

Ошибка резервирования CPU

Операционная система CPU серии Н вызывает организационный блок ОВ 72, если возникнет одно из следующих событий:

- Потеря резервирования CPU;
- Ошибка сравнения (например, в RAM в PIQ);

- Переключение резервный – основной;
- Ошибка синхронизации;
- Ошибка в подмодуле SYNC;
- Прерывание обновления.

Если ОВ 72 не существует, CPU в случае ошибки резервирования CPU продолжает работать.

23.4 Системная диагностика

23.4.1 Диагностические события и диагностический буфер

Системная диагностика направлена на обнаружение, оценку и сообщение об ошибках, возникающих в программируемых контроллерах. Примерами могут служить ошибки в пользовательской программе, сбой модулей или обрыв провода в сигнальных модулях. *Диагностическими событиями* могут быть:

- Диагностические прерывания от модулей с такой возможностью;
- Системные ошибки и переходы CPU из режима в режим;
- Пользовательские сообщения через системные функции.

Модули с диагностическими возможностями отличают программируемые и непрограммируемые диагностические события. О программируемых диагностических событиях сообщается, только когда вами установлены параметры, необходимые для активирования (разрешения) диагностики. О не программируемых диагностических событиях сообщается всегда, вне зависимости от того, разрешена ли диагностика. В случае диагностического события, о возникновении которого может быть сообщено,

- загорается светодиод сбоя на CPU;
- диагностическое событие передается операционной системе CPU;
- генерируется диагностическое прерывание, если вы установили параметры, разрешающе такие прерывания (по умолчанию диагностические прерывания запрещены).

Все диагностические события, о которых было сообщено операционной системе CPU, вводятся в *диагностический буфер* в порядке их возникновения с датой и временной меткой. Диагностический буфер – это снабженная резервным батарейным питанием область памяти CPU, которая сохраняет свое содержимое даже в случае сброса памяти. Диагностический буфер является кольцевым, и его размер зависит от версии CPU. Когда диагностический буфер заполнен, самая старая запись переписывается новой.

Вы можете в любой момент считать диагностический буфер с использованием программирующего устройства. В блоке параметров CPU *System Diagnostics (Системная диагностика)* вы можете определить, требуются ли вам расширенные диагностические записи (все вызовы ОВ). Вы также можете задать, должна ли последняя диагностическая запись, сделанная перед переходом CPU в режим STOP, быть отправлена в особый узел шины MPI.

23.4.2 Запись пользовательского ввода в диагностический буфер

Системная функция **SFC 52 WR_USMSG** вносит запись в диагностический буфер, она может быть передана всем узлам в сети MPI. Таблица 23.6 содержит параметры SFC 52.

Таблица 23.6 Параметры SFC 52 WR_USMSG

SFC	Имя параметра	Объявление	Тип данных	Содержимое, описание
52	SEND	INPUT	BOOL	Если «1», то передача разрешена
	EVENTN	INPUT	WORD	ID события
	INFO1	INPUT	ANY	Дополнительная информация 1 (одно слово)
	INFO2	INPUT	ANY	Дополнительная информация 2 (одно двойное слово)
	RET_VAL	OUTPUT	INT	Информация об ошибке

Ввод записи в диагностический буфер соответствует по формату записи системного события, например, стартовая информация для организационного блока. Из области допустимых значений вы можете выбрать ваш собственный ID (параметр EVENTN) дополнительную информацию (параметры INFO1 и INFO2).

ID события идентичен первым двум байтам записи буфера (рисунок 23.1). Для пользовательской записи допустимы классы событий 8 (диагностические записи для сигнальных модулей), 9 (стандартные пользовательские события), А и В (произвольные пользовательские события).

Дополнительная информация (INFO1) соответствует байтам 7 и 8 записи буфера (одно слово), а дополнительная информация (INFO2) – байтам с 9 по 12 (одно двойное слово). Содержимое обеих переменных может быть определено пользователем.



Рисунок 23.1 ID события для записей диагностического буфера

Чтобы передать диагностическую запись соответствующему узлу, установите SEND в «1». Даже если передача невозможна (к примеру, потому, что нет зарегистрированных узлов, или буфер передачи заполнен), запись все же заносится в диагностический буфер (когда бит 9 ID события установлен).

23.4.3 Оценка диагностических прерываний

Когда появляется входящее или уходящее прерывание, операционная система приостанавливает сканирование программы пользователя и вызывает организационный блок OB 82. Если OB 82 не запрограммирован, то по диагностическому прерыванию CPU переходит в режим STOP. Вы можете заблокировать или отменить блокировку OB 82 с помощью системных функций SFC 39 DIS_IRT или SFC 40 EN_IRT, а также ввести задержку или разрешить SFC 41 DIS_AIRT или SFC 42 EN_AIRT.

В первом байте стартовой информации B#16#39 означает входящее диагностическое прерывание, B#16#38 – уходящее диагностическое прерывание. Шестой байт содержит идентификатор адреса (B#16#54 обозначает вход, B#16#55 – выход); следующая переменная INT содержит адрес модуля, сгенерировавшего диагностическое прерывание. Следующие четыре байта содержат диагностическую информацию, предоставленную этим модулем.

Чтобы получить детальную информацию об ошибке, можно в OB 82 воспользоваться системной функцией SFC 59 RD_REC (чтение записи данных). Диагностическая информация действительна, пока OB 82 не окончил работу, то есть она остается «замороженной». Выход из OB 82 подтверждает диагностическое прерывание в модуле.

Диагностические данные модуля находятся в записях данных DS 0 и DS 1. Запись данных DS 0 содержит четыре байта диагностических данных, описывающих текущее состояние модуля. Содержимое этих четырех байтов идентично данным, находящимся в байтах с 8 по 11 стартовой информации OB 82.

Запись данных DS 1 содержит четыре байта записи данных DS 0 и, кроме того, определяемые модулем диагностические данные.

23.4.4 Чтение списка состояний системы

Список состояний системы (system status list, SSL) описывает текущее состояние программируемого контроллера. Используя функции, предназначенные для получения информации, список можно прочитать, изменить его нельзя. Вы можете прочитать часть списка (то есть подсписок) с помощью системной функции **SFC 51 RDSYSST**. Подсписки – это виртуальные списки; под этим подразумевается, что операционная система CPU предоставляет к ним доступ только по запросу. Параметры SFC 51 показаны в таблице 23.7.

REQ = «1» инициирует операцию чтения, а BUSY = «0» сообщает о ее завершении. Операционная система может выполнять несколько асинхронных операций чтения квазиодновременно; число операций зависит от используемого CPU. Если SFC 51

сообщает через значение функции о недостатке ресурсов (W#16#8085), вы должны заново послать запрос на чтение.

Таблица 23.7 Параметры SFC 51 RDSYSST

SFC	Имя параметра	Объявление	Тип данных	Содержимое, описание
51	REQ	INPUT	BOOL	Если «1»: передача запроса
	SSL_ID	INPUT	WORD	ID подписка
	INDEX	INPUT	WORD	Тип или номер объекта подписка
	RET_VAL	OUTPUT	INT	Информация об ошибке
	BUSY	OUTPUT	BOOL	Если «1»: операция чтения еще не завершена
	SSL_HEADER	OUTPUT	STRUCT	Размер и число прочитанных записей данных
	DR	OUTPUT	ANY	Поле для прочитанных записей данных

Содержимое параметров SSL_ID и INDEX определяется CPU. Параметр SSL_HEADER имеет тип данных STRUCT с переменными LENGTHDR (тип данных WORD) и N_DR (WORD) в качестве компонентов. В LENGTHDR содержится размер записи данных, N_DR показывает количество считанных записей данных.

Параметр DR используется для указания переменной или области данных, куда SFC 51 введет записи данных. Например, P#DB200.DBX0.0 WORD 256 предоставляет область 256 слов данных в блоке данных DB 200, начиная с DBB 0. Если область недостаточного размера, то будет введено допустимое количество записей. Пересылаются только целые записи данных. Определенная область должна вместить хотя бы одну запись данных.

Приложение

Данный раздел книги содержит сведения о полезных дополнениях к языкам программирования LAD и FBD, обзор содержимого библиотек блоков STEP 7 (STEP 7 Block Libraries) и обзор функций всех элементов LAD и FBD.

- Вы можете с помощью программы LAD/FBD обеспечить блоки **защитой (block protection)**. Для этой цели используется ориентированный на источник (исходные файлы) редактор языка программирования STL.
- Вы можете воспользоваться дополнительной функцией STL, **косвенной адресацией (indirect addressing)**, чтобы передавать области данных в языках программирования LAD и FBD; в этом случае адреса таких областей данных не вычисляются до выполнения программы. Библиотеки «LAD_Book» и «FBD_Book» на прилагаемой дискете содержат «Пример фрейма сообщения» («Sample Message Frame»), показывающий способ установки и передачи областей данных.
- Стандартный пакет STEP 7 включает в себя **библиотеки блоков (block libraries)** с загружаемыми функциями и функциональными блоками, а также с заголовками блоков и описаниями интерфейсов для системных блоков (SFC и SFB).
- Завершает книгу **обзор всех функций** LAD и FBD.

На прилагаемой к книге дискете вы можете найти архивные библиотеки «LAD_Book» и «FBD_Book». Эти библиотеки вы можете разархивировать в SIMATIC-менеджере с помощью команды **File → Retrieve (Файл → Разархивировать)**. Выберите архив (дискету) в открывшемся диалоговом окне. В следующем поле можно определить директорию назначения. Вообще, библиотеки располагаются в директории ... \STEP7\S7LIBS; но вы можете выбрать любую другую директорию, например, ... \STEP7\S7PROJ, которая обычно содержит проекты. В последнем поле «Retrieve - Options» («Разархивирование - Опции») вы можете отменить опцию «Restore full path» («Восстановить полный путь»).

Библиотеки «LAD_Book» и «FBD_Book» содержат по восемь программ, являющиеся примерами представлений LAD и FBD. Два больших примера показывают программирование функций, функциональных блоков и локальных экземпляров (пример конвейера) и обработку данных (пример фрейма сообщения). Требуется примерно 1,93 Мб дискового пространства.

Чтобы опробовать пример, установите проект, который соответствует вашей конфигурации аппаратных средств, и скопируйте программу, включая таблицу символов, из библиотеки в проект. Теперь вы можете протестировать программу в онлайн-режиме.

24 Дополнения к графическому программированию

Защита блока; косвенная адресация; пример фрейма сообщения

25 Библиотеки блоков

Организационные блоки; системные функциональные блоки; функциональные блоки IEC; блоки преобразования S5-S7; блоки преобразования TI-S7; блоки управления PID; коммуникационные блоки

26 Обзор функций LAD

Все функции LAD

27 Обзор функций FBD

Все функции FBD

Содержание главы 24

<u>24</u>	<u>Дополнения к графическому программированию</u>	4
<u>24.1</u>	<u>Защита блока</u>	4
<u>24.2</u>	<u>Косвенная адресация</u>	6
<u>24.2.1</u>	<u>Указатели: общие замечания</u>	6
<u>24.2.2</u>	<u>Указатель области</u>	6
<u>24.2.3</u>	<u>DB-указатель</u>	8
<u>24.2.4</u>	<u>ANY-указатель</u>	8
<u>24.2.5</u>	<u>«Переменный» ANY-указатель</u>	10
<u>24.3</u>	<u>Краткое описание «Примера фрейма сообщения»</u>	11

24 Дополнения к графическому программированию

24.1 Защита блока

Защиту блока представляет ключевое слово `KNOW_HOW_PROTECT`. Блок с этим атрибутом нельзя просмотреть, распечатать или изменить. Редактор только отображает заголовок блока и таблицу описаний с параметрами блока. При ориентированном на источник (исходный файл) вводе вы сами можете защитить любой блок с помощью `KNOW_HOW_PROTECT`. Это означает, что никто, включая вас, не сможет просмотреть скомпилированный блок (сохраните исходный файл в безопасном месте!).

Вы можете ввести защиту блока `KNOW_HOW_PROTECT` при помощи STL, и она должна быть ориентированной на источник. Чтобы реализовать это, действуйте следующим образом:

- 1) Создайте блок в LAD или FBD обычным способом. Позднее этот блок будет перезаписан в пользовательской программе *Blocks* (Блоки) блоком с ключевым словом. Если вы хотите сохранить (исходный) блок (настоятельно рекомендуется при вводе защиты блока), то перед вводом ключевого слова вы можете сохранить блок, к примеру, в (созданной пользователем) библиотеке. Таким же образом вы также можете сохранить всю пользовательскую программу.
- 2) Создайте контейнер исходных файлов. Если в *S7 program* (S7-программа) нет объекта *Source Files* (Исходные файлы) (на том же уровне, что и программа пользователя *Blocks*), вы должны создать его. Выберите *S7 program* и вставьте объект *Source Files* с помощью команды меню Insert → S7 Software → Source Directory (Вставка → Программное обеспечение S7 → Директория исходных файлов).
- 3) Из блока сгенерируйте исходный файл STL. Перейдите в редактор (через панель задач, например, или откройте любой блок в *Blocks* и затем закройте его) и выберите пункт меню File → Generate Source File (Файл → Генерирование исходного файла). В появившемся диалоговом окне установите ваш проект, выберите объект *Source Files* и назначьте имя исходному файлу в поле «Object Name» («Имя объекта»). Нажмите «ОК». В следующем диалоговом окне вам будут показаны все блоки в контейнере *Blocks*; отметьте блок(и), из которого вы хотите создать исходный файл. Нажмите «ОК».
- 4) Откройте исходный файл (например, двойным щелчком на символе исходного файла в SIMATIC-менеджере или с помощью редактора и команды File → Open, Файл → Открыть). Теперь вы можете увидеть исходный ASCII-файл вашего LAD/FBD-блока. Если вы предварительно отметили несколько блоков, то эти блоки будут скомпонованы по порядку в исходном файле.

Ввод для кодовых блоков осуществляется в следующем порядке:

- Ключевое слово для типа блока (FUNCTION, FUNCTION_BLOCK, ORGANIZATION_BLOCK) с указанием адреса. Затем может следовать заголовок блока (начинающийся с TITLE=...) и комментарий блока (начинающийся с //...).
- Атрибуты блока (в зависимости от того, заполнили вы поля в окне свойств (Properties) блока или нет; если да, то сколько).
- Описание (объявление) переменных (несколько разделов с ключевыми словами VAR_xxx, ..., END_VAR); оно зависит от того, объявлены локальные блочные переменные, и если так, то какие.
- Программа, начинающаяся с BEGIN и заканчивающаяся ключевым словом, определяющим конец блока (к примеру, END_FUNCTION_BLOCK).

Ввод для блоков данных производится в следующем порядке:

- Ключевое слово для типа блока (DATA_BLOCK) с указанием адреса.
 - Атрибуты блока (зависящие от того, заполнили вы поля в окне свойств (Properties) блока или нет; если да, то сколько).
 - Описание переменных (начинающееся со STRUCT и заканчивающееся END_STRUCT) или, в случае экземплярных блоков данных, адрес соответствующего функционального блока.
 - Инициализация переменных, начинающаяся с BEGIN и заканчивающаяся END_DATA_BLOCK.
- 5) В исходном файле вводится ключевое слово KNOW_HOW_PROTECT во всех случаях в своей строке после атрибутов блока и перед объявлением переменных. Если вы создали исходный файл из нескольких блоков, введите ключевые слова во все выбранные блоки. В заключении сохраните исходный файл.
- 6) Откомпилируйте исходный файл, вызвав команду меню File → Compile (Файл → Компилировать). Компилятор создаст блок с определенными атрибутами (в используемом для создания языке STL в случае кодовых блоков; используемый для создания язык здесь значения не имеет, так как KNOW_HOW_PROTECT означает, что блок нельзя будет просмотреть или распечатать). Компилированный (новый) блок находится в пользовательской программе *Blocks* и заменяет (старый) блок с тем же номером.

24.2 Косвенная адресация

Язык программирования STL предоставляет вам метод доступа к операндам, адреса которых не вычисляются до исполнения программы. В меньшей степени это возможно в LAD или FBD: вы можете подождать до выполнения программы, чтобы определить, какие области данных требуется скопировать при помощи SFC 20 BLKMOV.

Впрочем, сначала немного полезной информации об указателях.

24.2.1 Указатели: общие замечания

Для косвенной адресации вам потребуется формат данных, содержащий адрес бита, а также адрес байта и, если применимо, область операнда. Этим форматом данных является *указатель (pointer)*. Указатель также используется для ссылки на операнд. Имеется три вида указателей:

- указатели областей (area pointers); их размер – 32 бита, и они содержат определенный операнд или его адрес;
- DB-указатели (DB pointers); их размер составляет 48 битов, и вдобавок к указателю области они также содержат номер блока данных;
- ANY-указатели (ANY pointers); они имеют размер 80 битов и в дополнение к DB-указателю содержат дополнительную информацию, например, тип данных операнда.

24.2.2 Указатель области

Указатель области содержит адрес операнда и, если применимо, область операнда. Без области операнда это внутренний указатель области (area-internal); если указатель содержит также область операнда, то он называется указателем пересечения области (area-crossing).

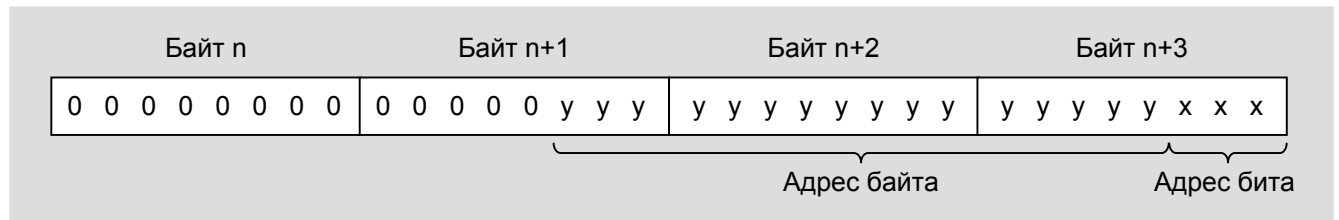
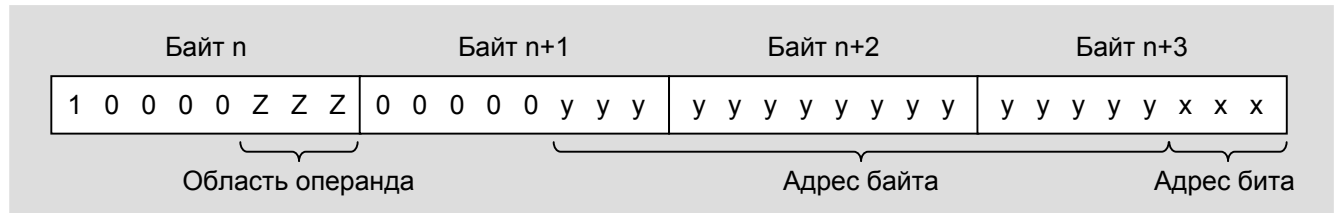
Форма записи представления константы следующая:

R#y.x для внутреннего указателя области, например, R#22.0;

R#Zy.x для указателя пересечения области, например, R#M22.0,

где x = адрес бита, y = адрес байта, а Z = область. Вы можете задать идентификатор (ID) операнда в качестве области. Два типа указателей различаются по значению 31-го бита.

На рисунке 24.1 показаны все типы указателей и их содержимое, как это реализовано в STEP 7.

Внутренний указатель области**Указатель пересечения области**

	ДВ-указатель	ANY-указатель для типов данных	ANY-указатель для таймеров / счетчиков	ANY-указатель для блоков
Байт n	Номер	16#10	16#10	16#10
Байт n+1	блока данных	Тип	Тип	Тип
Байт n+2		Количество	Количество	Количество
Байт n+3				
Байт n+4	Указатель	Номер	16#0000	16#0000
Байт n+5	области	блока данных		
Байт n+6		Указатель	Тип	16#0000
Байт n+7		области	16#00	
Байт n+8			Номер	Номер
Байт n+9				

Область адресов:

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 с адресацией пересечения области

²⁾ Только с передачей параметров блока

Тип в ANY-указателе:

Простые типы данных	Сложные типы данных	Параметрические типы
01 BOOL	0E DT	17 BLOCK_FB
02 BYTE	13 STRING	18 BLOCK_FC
03 CHAR		19 BLOCK_DB
04 WORD		1A BLOCK_SDB
05 INT		1C COUNTER
06 DWORD		1D TIMER
07 DINT		
08 REAL		
09 DATE		
0A TOD		
0B TIME		
0C S5TIME		
	Нулевой указатель	
	00 NIL	

Рисунок 24.1 Структура указателей в STEP 7

Указатель области имеет, в принципе, адрес бита, который всегда должен быть определен даже с числовыми операндами; в случае числовых операндов укажите 0 в качестве адреса бита. Пример: вы можете использовать указатель области P#M22.0, чтобы адресовать (меркерный) бит памяти M 22.0, но, кроме того, байт памяти MB 22, слово памяти MW 22 или двойное слово памяти MD 22.

24.2.3 DB-указатель

В дополнение к указателю области DB-указатель также содержит номер блока данных в виде положительного числа INT. Он определяет блок данных, если указатель области ссылается на глобальные данные или область экземплярных данных. Во всех других случаях первые два байта содержат ноль.

Форма записи указателя знакома вам по полной адресации операндов данных. Здесь также определяются блок данных и операнд данных, разделенные точкой: P#DataBlock.DataOperand (P#БлокДанных.ОперандДанных).

Пример: P#DB 10.DBX 20.5

Вы можете применить этот указатель к параметру блока параметрического типа POINTER для ссылки на операнд данных. Редактор использует этот тип указателя в своей работе для передачи фактических параметров.

24.2.4 ANY-указатель

Кроме DB-указателя ANY-указатель также содержит тип данных и коэффициент повторения. Это дает возможность указывать на область данных.

ANY-указатель доступен в двух вариантах: для переменных с типами данных и для переменных параметрических типов. Если вы ссылаетесь на переменную с типом данных, ANY-указатель содержит DB-указатель, тип и коэффициент повторения. Если ANY-указатель ссылается на переменную параметрического типа, он содержит вместо DB-указателя только номер в дополнение к типу. В случае функции таймера или счетчика тип повторяется в байте (n + 6); байт (n + 7) содержит B#16#00. В остальных случаях эти два байта содержат значение W#16#0000.

В первом байте ANY-указателя хранится синтаксический ID; в STEP 7 он всегда равен 10hex. Тип определяет тип данных переменной, для которой применяется ANY-указатель. Переменные простых типов, DT и STRING приобретают указанный тип и количество 1.

Если вы в параметре ANY применяете переменную типа ARRAY или STRUCT (а также UDT), то редактор генерирует ANY-указатель на поле или структуру. Этот ANY-указатель содержит ID для BYTE (02hex) в качестве типа и байтовый размер переменной в качестве количества. Тип данных отдельного поля или компонента структуры здесь не важен. Таким образом, ANY-указатель ссылается с использованием удвоенного числа байтов на поле WORD. Исключение: указатель на поле, состоящее из компонентов типа CHAR, создается также с типом CHAR (03hex).

Вы можете применить ANY-указатель в параметре блока параметрического типа ANY, если вы хотите получить ссылку на переменную или область операнда. Представление константы для типов данных следующее:

R#[DataBlock.]Operand Type Quantity (R#[БлокДанных.]Операнд Тип Количество)

Примеры:

- R#DB 11.DBX 30.0 INT 12
Область с 12 словами в DB 11, начиная с DBB 30;
- R#M 16.0 BYTE 8
Область с 8 байтами, начиная с MB 16;
- R#E 18.0 WORD 1
Входное слово IW 18;
- R#E 1.0 BOOL 1
Вход I 1.0.

В случае параметрических типов указатели записываются в следующем виде:

L#Number Type Quantity (L#Номер Тип Количество)

Примеры:

- L#10TIMER 1
Функция таймера T 10;
- L#2 COUNTER 1
Функция счетчика C 2.

Затем редактор применяет ANY-указатель, который соответствует по типу и количеству со спецификацией в представлении константы. Обратите внимание, что для типов данных адрес операнда в ANY-указателе должен быть также адресом бита.

Определение постоянного ANY-указателя имеет смысл, если вы хотите обратиться к области данных, для которой не объявлено никаких переменных. В принципе вы также можете применить переменные или операнды в параметре ANY. Например, представление «R#1 1.0 BOOL 1» идентично «I 1.0» или соответствующему символическому адресу.

Если вы не указываете значений по умолчанию при описании (объявлении) параметра ANY в функциональном блоке, то редактор назначает 10hex синтаксическому ID и 00hex остальным байтам. Тогда редактор представляет (пустой) ANY-указатель (в режиме просмотра данных) в следующем виде: R#P0.0 VOID 0.

24.2.5 «Переменный» ANY-указатель

При копировании с использованием SFC 20 вы указываете в параметрах источника и назначения либо абсолютно адресованную область (к примеру, R#DB127.DBX0.0 BYTE 32), либо переменную. В обоих случаях исходная область и область назначения во время программирования фиксированы (переменная индексация невозможна даже для элементов массива). Доступен следующий метод для модификации в ходе выполнения программы области данных, созданной в параметре типа ANY.

Во временных локальных данных создается переменная типа данных ANY и используется для инициализации параметра ANY. Редактор программ в таком случае не генерирует ANY-указатель (как он поступил бы для другой переменной), но использует переменную ANY во временных локальных данных в качестве ANY-указателя на область-источник и область назначения. Переменная ANY во временных локальных данных структурирована точно так же, как ANY-указатель; теперь вы можете изменять отдельные вводы информации (записей) в ходе выполнения программы.

Процедура работает не только в случае SFC 20 BLKMOV, но и с параметрами типа ANY в других блоках.

Библиотеки «LAD_Book» и «FBD_Book» на прилагаемой к книге дискете содержат программу «Message Frame Example» («Пример фрейма сообщения»), которая, в свою очередь, содержит пример «переменного» ANY-указателя.

24.3 Краткое описание «Примера фрейма сообщения»

В первую очередь в примере рассматривается управление данными. Он разбит на разделы следующим образом:

- Данные фрейма сообщения; показана обработка структур данных;
- Опрос времени суток; показано управление системными и стандартными блоками;
- Редактирование фрейма сообщения; демонстрируется использование SFC 20 VLKMOV с фиксированными адресами;
- Непрямое копирование области данных; показано использование функции «непрямого копирования» с помощью «переменных» ANY-указателей;
- Сохранение фрейма сообщения; продемонстрировано применение «непрямого копирования».

На рисунке 24.2 представлена программа и структура данных для этого примера.

Данные фрейма сообщения

Пример показывает вам, как можно определить часто встречающиеся структуры данных в качестве своего собственного типа данных и как использовать этот тип данных при описании переменных и параметров.

Мы конструируем хранилище данных для входящих и исходящих фреймов сообщений: почтовый ящик передачи (Send mailbox) со структурой фрейма сообщения, почтовый ящик приема (Receive mailbox) той же структуры и кольцевой буфер (приема) для обеспечения промежуточного хранения для входящих фреймов сообщений. Так как структура фрейма сообщения встречается часто, мы определим ее как фрейм определенного пользователем типа данных (UDT). Фрейм сообщения содержит заголовок фрейма, структуру которого также требуется назвать. Почтовый ящик передачи (Send) и почтовый ящик приема (Receive) будут блоками данных, содержащими переменную структуры *frame* (*фрейм*). Наконец, имеется кольцевой буфер, блок данных с массивом из восьми компонентов, которые имеют такую же структуру данных, как и *frame* (*фрейм*).

Опрос времени суток

Пример демонстрирует обработку системных и стандартных блоков (выявление ошибок, копирование из библиотеки, переименование).

Функция считывания времени суток должна в качестве своего значения вывести время суток из встроенного в CPU таймера реального времени. Для этой цели нам потребуется системная функция SFC 1 READ_CLK, которая считывает дату и время

суток из таймера реального времени в формате данных DATE_AND_TIME или DT. Так как мы хотим прочитать только время суток, то также понадобится IEC-функция FC 8 DT_TOD. Эта функция получает время суток в формате TIME_OF_DAY или TOD из формата данных DT.

Пример фрейма сообщения

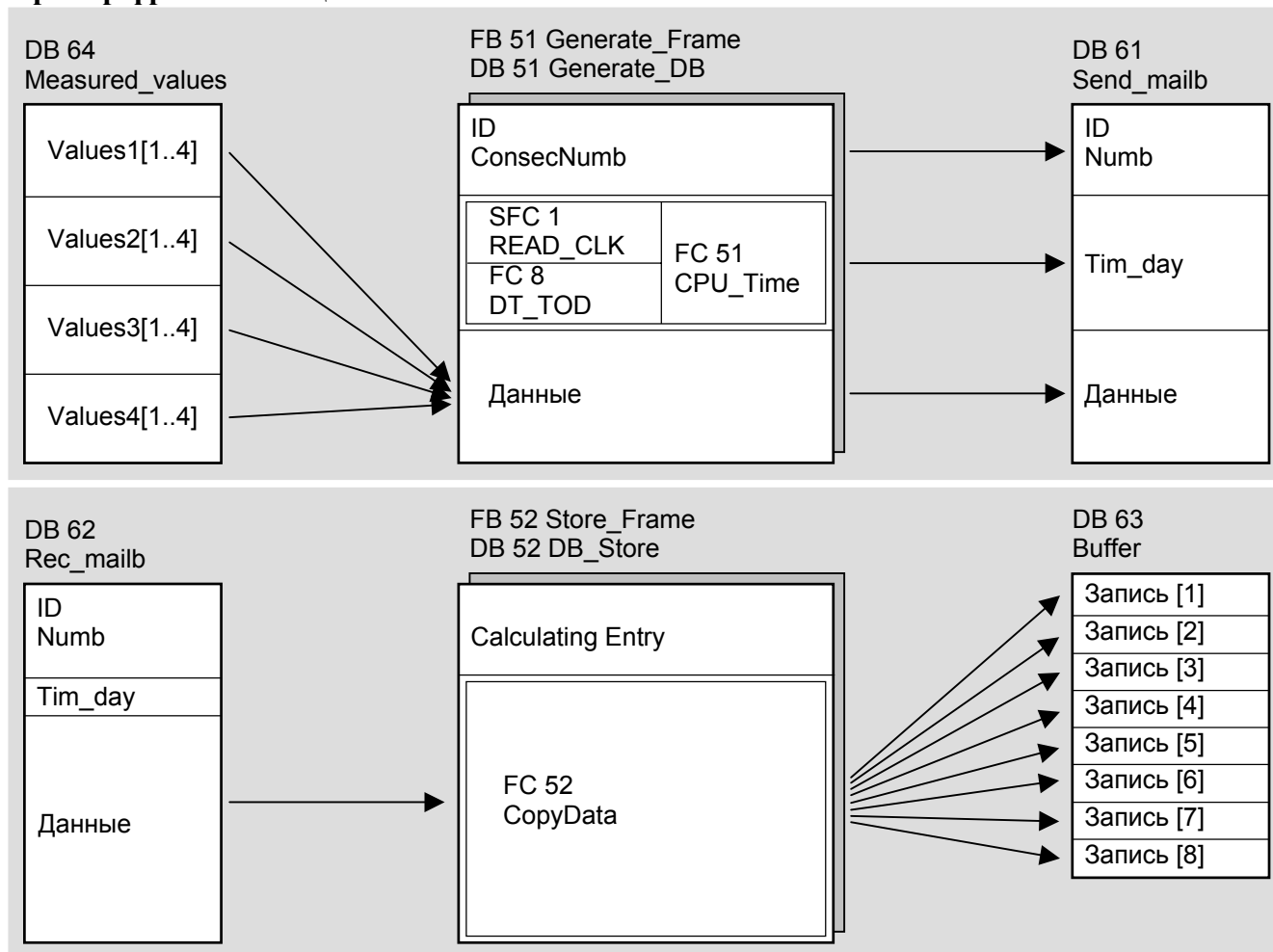


Рисунок 24.2 Структура данных для примера данных фрейма сообщения

Оценка ошибок

Системные функции сообщают об ошибке через параметр BR (бинарный результат) и значение функции RET_VAL. Ошибка возникла, если бинарный результат BR = «0»; значение функции в таком случае отрицательно (бит 15 установлен). Стандартные IEC-функции сигнализируют об ошибке только посредством бинарного результата. В примере показаны оба вида оценки ошибки. Если ошибка была выявлена, то выводится недействительное значение времени суток. Также устанавливается бинарный результат. После вызова опроса времени суток вы можете, таким образом, по бинарному результату определить, возникла ли ошибка.

Автономное программирование системных функций

Перед сохранением блока входа системная функция SFC 1 и стандартная функция FC 8 должны быть внесены в автономную пользовательскую программу. Обе функции включены в стандартный пакет STEP 7. Вы можете найти эти функции в предоставляемой библиотеке блоков. (Что касается системных функций, встроенных в CPU, библиотека содержит только описание их интерфейсов, а не фактическую программу системных функций. Функция может быть вызвана автономно через это описание интерфейса; описание интерфейса не передается в CPU. Загружаемые функции, такие как IEC-функции, доступны в библиотеке как исполняемые программы.)

Выберите стандартную библиотеку *Standard Library* с помощью команды File → Open → Library (Файл → Открыть → Библиотека) в SIMATIC-менеджере и откройте библиотеку *System Function Blocks* (*Системные функциональные блоки*). В *Blocks* (*Блоки*) вы можете найти все описания интерфейсов системных функций. Если окно вашего проекта все еще открыто, то можно отобразить два окна рядом, выбрав команду меню Window → Arrange → Vertically (Окно → Упорядочить → Вертикально), и перетащить с помощью мыши выбранные системные функции в вашу программу (отметьте мышью SFC, удерживая нажатой кнопку мыши, перетащите в *Blocks* или в ее открытое окно и отпустите). Таким же образом скопируйте стандартную функцию FC 8. Вы можете найти ее в библиотеке *IEC Function Blocks* (*Функциональные блоки IEC*). FC 8 является загружаемой функцией; вследствие этого она резервирует пользовательскую память в отличие от SFC 1.

Если стандартный функциональный блок вызывается из раздела «Libraries» («Библиотеки») каталога программных элементов (Program Element Catalog) при помощи редактора, то он автоматически копируется в *Blocks* и вносится в таблицу символов.

Переименование стандартных функций

Вы можете переименовать загружаемую стандартную функцию. Выберите стандартную функцию (например, FC 8) в окне проекта и щелкните (еще раз) на обозначении. Имя отобразится в рамке, и вы сможете указать новый адрес (например, FC 98). Если вы нажмете клавишу F1, пока отмечена стандартная функция (переименованная в FC 98), то вы тем не менее получите онлайн-помощь по исходной стандартной функции FC 8.

Если при копировании имеется идентично адресованный блок, то появляется диалоговое окно, где вы можете выбрать между перезаписью и переименованием.

Символический адрес

Присвоить имена системным и стандартным функциям вы можете в таблице символов с тем, чтобы также можно было обратиться к этим функциям символически. Вы можете произвольно назначить имена в рамках правил, применимых к именам блоков. В примере имя блока каждый раз выбирается как символическое имя (для лучшей идентификации).

Редактирование фрейма сообщения

Блок данных *Send_Mailb* должен быть заполнен данными фрейма сообщения. Мы используем функциональный блок, который имеет ID и последовательный номер, записанные в его экземплярном блоке данных. Сетевые данные окончательно записываются в глобальный блок данных; они копируются в почтовый ящик передачи (Send) с помощью системной функции BLKMOV. Мы используем функцию опроса времени суток для считывания времени из таймера реального времени CPU.

Первая сеть (network) в функциональном блоке FB *Generate_Frame* передает ID, сохраненный в экземпляре блока данных, в заголовок фрейма. Последовательный номер увеличивается на 1 и также пересылается в заголовок фрейма.

Вторая сеть содержит вызов функции *READ_CLK*, который считывает время суток из таймера реального времени и вводит его в заголовок фрейма в формате TIME_OF_DAY.

В последующих сетях вы увидите метод копирования выбранных переменных в ходе выполнения программы при помощи системной функции SFC 20 BLKMOV и без использования косвенной адресации. Таким образом, нет необходимости знать абсолютный адрес или структуру переменных. Принцип чрезвычайно прост: требуемая функция выбирается с использованием функций сравнения. В качестве критерия выбора допустимы номера с 1 по 14.

FB *Generate_Frame* программируется таким образом, что он вызывается для генерирования фрейма сообщения посредством фронта сигнала.

Косвенное копирование области данных

Пример показывает редактирование и использование «переменного» ANY-указателя с графическими программными элементами.

Функция *I_Copy* копирует область данных, требуемые адрес и размер которой вы можете установить через параметры блока. Отдельные параметры блока соответствуют отдельным элементам ANY-указателя (обратитесь к параграфу 24.2.4 «ANY-указатель»). Указанные в параметрах блока значения должны быть действительными; они не проверяются (SFC 20 BLKMOV сообщает об ошибке копирования в своем параметре значения функции, который пересылается в параметр значения функции *I_Copy*).

Существенными элементами являются две временных переменных *SoPointer* и *DesPointer* типа данных ANY. Они содержат ANY-указатели для системной функции SFC 20 BLKMOV. *SoPointer* указывает на исходную область данных, предназначенных для передачи, *DesPointer* ссылается на область назначения. На рисунке 24.3 показана структура переменной *SoPointer*; *DesPointer* имеет такую же структуру. Доступ к отдельным байтам, словам и двойным словам переменных ANY осуществляется через их абсолютные адреса.

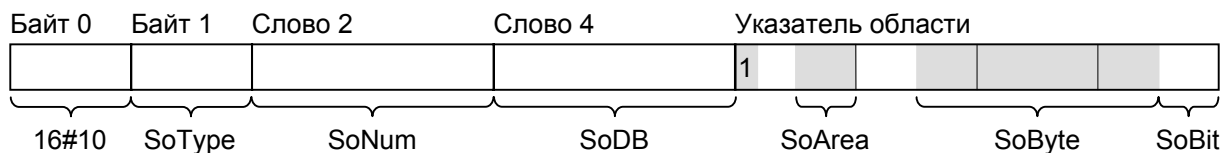


Рисунок 24.3 Структура переменной *SoPointer*

Сохранение фрейма сообщения

Пример демонстрирует использование функции *I_Copy* (копирование области данных с программируемым адресом).

Фрейм сообщения в блоке данных *Rec_Mailb* должен быть записан в следующую ячейку блока данных *Buffer*. Локальная блочная переменная *Entry* определяет местоположение в кольцевом буфере; значение этой ячейки используется для вычисления адреса в кольцевом буфере.

Переменная *Entry* имеет значение из области от 0 до 7. В первой сети компаратор определяет, *Entry* меньше 7 или нет. Если меньше, то *Entry* инкрементируется на 1 в следующей сети, иначе она устанавливается в 0. *Entry*, умноженная на 16, дает абсолютный байтовый адрес следующей записи данных в кольцевом буфере (структура данных *Message frame* состоит из 16 байтов).

Функция *I_Copy*, которая копирует фрейм сообщения из почтового ящика приема (Receive) (блок данных DB 62) в кольцевой буфер (блок данных DB 63), вызывается в сети 3 (Network 3).

Содержание главы 25

<u>25</u>	<u>Библиотеки блоков</u>	4
<u>25.1</u>	<u>Организационные блоки</u>	4
<u>25.2</u>	<u>Системные функциональные блоки</u>	6
<u>25.3</u>	<u>Функциональные блоки IEC</u>	11
<u>25.4</u>	<u>Блоки преобразования S5-S7</u>	13
<u>25.5</u>	<u>Блоки преобразования TI-S7</u>	16
<u>25.6</u>	<u>Блоки PID-управления</u>	17
<u>25.7</u>	<u>Коммуникационные блоки</u>	17

25 Библиотеки блоков

Базовое программное обеспечение STEP 7 включает в свой состав стандартную библиотеку *Standard Library*, которая содержит следующие библиотечные программы:

- Организационные блоки (Organization Blocks);
- Системные функциональные блоки (System Function Blocks);
- Функциональные блоки IEC (IEC Function Blocks);
- Блоки преобразования S5-S7 (S5-S7 Converting Blocks);
- Блоки преобразования TI-S7 (TI-S7 Converting Blocks);
- Блоки PID-управления (PID Control Blocks);
- Коммуникационные блоки (Communication Blocks).

Из указанных библиотечных программ вы можете копировать блоки и описания интерфейсов в собственные проекты.

25.1 Организационные блоки

(Пр = приоритетный класс по умолчанию)

ОВ	Пр	Назначение
1	1	Главная программа
10	2	Прерывание по времени суток 0
11	2	Прерывание по времени суток 1
12	2	Прерывание по времени суток 2
13	2	Прерывание по времени суток 3
14	2	Прерывание по времени суток 4
15	2	Прерывание по времени суток 5
16	2	Прерывание по времени суток 6
17	2	Прерывание по времени суток 7
20	3	Прерывание задержки времени 0
21	4	Прерывание задержки времени 1
22	5	Прерывание задержки времени 2
23	6	Прерывание задержки времени 3
30	7	Циклическое прерывание 0 (5 с)
31	8	Циклическое прерывание 1 (2 с)

ОВ	Пр	Назначение
32	9	Циклическое прерывание 2 (1 с)
33	10	Циклическое прерывание 3 (500 мс)
34	11	Циклическое прерывание 4 (200 мс)
35	12	Циклическое прерывание 5 (100 мс)
36	13	Циклическое прерывание 6 (50 мс)
37	14	Циклическое прерывание 7 (20 мс)
38	15	Циклическое прерывание 8 (10 мс)
40	16	Аппаратное прерывание 0
41	17	Аппаратное прерывание 1
42	18	Аппаратное прерывание 2
43	19	Аппаратное прерывание 3
44	20	Аппаратное прерывание 4
45	21	Аппаратное прерывание 5
46	22	Аппаратное прерывание 6
47	23	Аппаратное прерывание 7
60	25	Мультипроцессорное прерывание
70	25	Ошибка резервирования входа/выхода ¹⁾
72	28	Ошибка резервирования CPU
73	25	Ошибка коммуникационного резервирования
80	26	Временная ошибка ¹⁾
81	26	Сбой электропитания ¹⁾
82	26	Диагностическое прерывание ¹⁾
83	26	Прерывание вставки/удаления модуля ¹⁾
84	26	Сбой оборудования CPU ¹⁾
85	26	Ошибка приоритетного класса ¹⁾
86	26	Ошибка DP ¹⁾
87	26	Коммуникационная ошибка ¹⁾
90	29	Фоновая обработка
100	27	Полный рестарт
101	27	Рестарт
102	27	«Холодный» рестарт
121	-	Программная ошибка
122	-	Ошибка доступа к входу/выходу

¹⁾ Пр = 28 при рестарте

25.2 Системные функциональные блоки

IEC-таймеры и IEC-счетчики

SFB	Имя	Назначение
0	STU	Счетчик прямого действия (по возрастанию)
1	STD	Счетчик обратного действия (по убыванию)
2	STUD	Счетчик прямого/обратного действия
3	TP	Импульсный
4	TON	Задержка включения
5	TOF	Задержка выключения

Коммуникация через сконфигурированные соединения

SFB	Имя	Назначение
8	USEND	Некоординированная передача
9	URVC	Некоординированный прием
12	BSEND	Ориентированная на блок передача
13	BRCV	Ориентированный на блок прием
14	GET	Чтение данных партнера
15	PUT	Запись данных в партнера
16	PRINT	Передача данных на принтер
19	START	Инициация полного рестарта партнера
20	STOP	Перевод партнера в состояние STOP
21	RESUME	Инициация рестарта партнера
22	STATUS	Опрос состояния партнера
23	USTATUS	Получение данных о состоянии партнера
SFC	Имя	Назначение
62	CONTROL	Опрос состояния коммуникации

Встроенные функции CPU 312/314/614

SFB	Имя	Назначение
29	HS_COUNT	Высокоскоростной счетчик
30	FREQ_MES	Частотомер
38	HSC_A_B	Элемент управления «Счетчик А/В»
39	POS	Элемент управления «Позиционирование»
41	CONT_C	Элемент непрерывного циклического управления
42	CONT_S	Элемент управления пошаговыми действиями
43	PULSEGEN	Генерирование импульсов

SFC	Имя	Назначение
63	AB_CALL	Вызов ассемблерного блока

Системная диагностика

SFC	Имя	Назначение
6	RD_SINFO	Чтение стартовой информации
51	RDSYSST	Чтение подписка SYS ST
52	WR_USMSG	Ввод в диагностический буфер

Создание сообщений, относящихся к блоку

SFB	Имя	Назначение
33	ALARM	Сообщения с окном подтверждения
34	ALARM_8	Сообщения без сопутствующих значений
35	ALARM_8P	Сообщения с сопутствующими значениями
36	NOTIFY	Сообщения с окном подтверждения
37	AR_SEND	Передача архивных данных

SFC	Имя	Назначение
9	EN_MSG	Разрешение сообщений
10	DIS_MSG	Запрет сообщений
17	ALARM_SQ	Сообщения, которые могут быть подтверждены
18	ALARM_S	Сообщения, которые всегда подтверждаются
19	ALARM_SC	Определение состояния подтверждения

Внутренний таймер CPU и счетчик рабочего времени

SFC	Имя	Назначение
0	SET_CLK	Установка таймера
1	READ_CLK	Чтение таймера
2	SET_RTM	Установка счетчика рабочего времени
3	CTRL_RTM	Модифицирование счетчика рабочего времени
4	READ_RTM	Чтение счетчика рабочего времени
48	SNC_RTCB	Синхронизация таймеров ведомых
64	TIME_TCK	Чтение системного времени

Барабан

SFC	Имя	Назначение
32	DRUM	Барабан

Копирование и функции для работы с блоками

SFC	Имя	Назначение
20	BLKMOV	Копирование области данных
21	FILL	Предварительное назначение области данных
22	CREAT_DB	Генерирование блока данных
23	DEL_DB	Удаление блока данных
24	TEST_DB	Тестирование блока данных
25	COMPRESS	Сжатие памяти
44	REPL_VAL	Ввод заменяющего значения
81	UBLKMOV	Копирование области данных без промежутков

Адресация модулей

SFC	Имя	Назначение
5	GADR_LGC	Определение логического адреса
49	LGC_GADR	Определение слота
50	RD_LGADR	Определение всех логических адресов

Распределенные входы/выходы

SFC	Имя	Назначение
7	DP_PRAL	Инициация аппаратного прерывания
11	DPSYN_FR	SYNC/FRRZE
12	D_ACT_DP	Деактивирование или активирование DP-ведомого
13	DPNRM_DG	Чтение диагностических данных
14	DPRD_DAT	Чтение данных ведомых
15	DPWR_DAT	Запись данных ведомых

Программное управление

SFC	Имя	Назначение
43	RE_TRIGR	Перезапуск монитор времени цикла
46	STP	Переход в режим STOP
47	WAIT	Ожидание в течение времени задержки

Передача записи данных

SFC	Имя	Назначение
54	RD_DPARM	Чтение предопределенного параметра
55	WR_PARM	Запись динамического параметра
56	WR_DPARM	Запись предопределенного параметра
57	PARM_MOD	Параметризация модуля
58	WR_REC	Запись элемента данных
59	RD_REC	Чтение записи данных

Обновление образа процесса

SFC	Имя	Назначение
26	UPDAT_PI	Обновление входной таблицы образа процесса
27	UPDAT_PO	Обновление выходной таблицы образа процесса
79	SET	Установка битового поля входа/выхода
80	RSET	Сброс битового поля входа/выхода

События прерываний

SFC	Имя	Назначение
28	SET_TINT	Установка прерывания по времени суток
29	CAN_TINT	Отмена прерывания по времени суток
30	ACT_TINT	Активация прерывания по времени суток
31	QRY_TINT	Запрос прерывания по времени суток
32	SRT_DINT	Запуск прерывания задержки времени
33	CAN_DINT	Отмена прерывания задержки времени
34	QRY_DINT	Запрос прерывания задержки времени
35	MP_ALM	Вызов мультипроцессорного предупреждения
36	MSK_FLT	Маскирование синхронных ошибок
37	DMSK_FLT	Демаскирование синхронных ошибок
38	READ_ERR	Чтение регистра состояния события
39	DIR_IRT	Запрет асинхронных ошибок
40	EN_IRT	Разблокировка асинхронных ошибок
41	DIS_AIRT	Задержка асинхронных ошибок
42	EN_AIRT	Разрешение асинхронных ошибок

Коммуникация через неконфигурированные соединения

SFC	Имя	Назначение
65	X_SEND	Внешняя передача данных
66	X_RCV	Внешний прием данных
67	X_GET	Внешнее чтение данных
68	X_PUT	Внешняя запись данных
69	X_ABORT	Разрыв внешнего соединения
72	I_GET	Внутренне чтение данных
73	I_PUT	Внутренняя запись данных
74	I_ABORT	Разрыв внутреннего соединения

Коммуникация глобальных данных

SFC	Имя	Назначение
60	GD_SND	Передача GD-пакета
61	GD_RCV	Прием GD-пакета

H-CPU

SFC	Имя	Назначение
90	H_CTRL	Управление рабочими режимами в H-CPU

25.3 Функциональные блоки ИЕС

Операции сравнения

FC	Имя	Назначение
9	EQ_DT	Сравнение DT «равно»
28	NE_DT	Сравнение DT «неравно»
14	GT_DT	Сравнение DT «больше»
12	GE_DT	Сравнение DT «больше или равно»
23	LT_DT	Сравнение DT «меньше»
18	LE_DT	Сравнение DT «меньше или равно»
10	EQ_STRING	Сравнение STRING «равно»
29	NE_STRING	Сравнение STRING «неравно»
15	GT_STRING	Сравнение STRING «больше»
13	GE_STRING	Сравнение STRING «больше или равно»
24	LT_STRING	Сравнение STRING «меньше»
19	LE_STRING	Сравнение STRING «меньше или равно»

Функции даты и времени

FC	Имя	Назначение
3	D_TOD_DT	Комбинирование DATE и TOD в DT
6	DT_DATE	Получение DATE из DT
7	DT_DAY	Получение дня недели из DT
8	DT_TOD	Получение TOD из DT
33	S5TI_TIM	Преобразование S5TIME в TIME
40	TIM_S5TI	Преобразование TIME в S5TIME
1	AD_DT_TM	Сложение TIME с DT
35	SB_DT_TM	Вычитание TIME из DT
34	SB_DT_DT	Вычитание DT из DT

Математические функции

FC	Имя	Назначение
22	LIMIT	Ограничитель
25	MAX	Выбор максимума
27	MIN	Выбор минимума
26	SEL	Двоичный выбор

Функции для работы со строками

FC	Имя	Назначение
21	LEN	Длина STRING
20	LEFT	Левая часть STRING
32	RIGHT	Правая часть STRING
26	MID	Средняя часть STRING
2	CONCAT	Конкатенация STRING
17	INSERT	Вставка STRING
4	DELETE	Удаление STRING
31	REPLACE	Замена STRING
11	FIND	Поиск STRING
16	I_STRING	Преобразование INT в STRING
5	DI_STRING	Преобразование DINT в STRING
30	R_STRING	Преобразование REAL в STRING
38	STRING_I	Преобразование STRING в INT
37	STRING_DI	Преобразование STRING в DINT
39	STRING_R	Преобразование STRING в REAL

25.4 Блоки преобразования S5-S7

Арифметические операции над числами с плавающей точкой

FC	Имя	Назначение
61	GP_FPGP	Преобразование числа с фиксированной точкой в число с плавающей точкой
62	GP_GPFP	Преобразование числа с плавающей точкой в число с фиксированной точкой
63	GP_ADD	Сложение чисел с плавающей точкой
64	GP_SUB	Вычитание чисел с плавающей точкой
65	GP_MUL	Умножение чисел с плавающей точкой
66	GP_DIV	Деление чисел с плавающей точкой
67	GP_VGL	Сравнение чисел с плавающей точкой
68	GP_RAD	Извлечение квадратного корня из числа с плавающей точкой

Базовые функции

FC	Имя	Назначение
85	ADD_32	Сложение 32-битных чисел с фиксированной точкой
86	SUB_32	Вычитание 32-битных чисел с фиксированной точкой
87	MUL_32	Умножение 32-битных чисел с фиксированной точкой
88	DIV_32	Деление 32-битных чисел с фиксированной точкой
89	RAD_16	Извлечение квадратного корня из 16-битного числа с фиксированной точкой
90	REG_SCHB	Регистр побитного сдвига
91	REG_SCHW	Регистр сдвига слова
92	REG_FIFO	Буфер (FIFO)
93	REG_LIFO	Стек (LIFO)
94	DB_COPY1	Копирование области данных (прямое)
95	DB_COPY2	Копирование области данных (косвенное)
96	RETTEN	Сохранение сверхоперативной памяти (S5-155U)
97	LADEN	Загрузка сверхоперативной памяти (S5-155U)
98	COD_B8	Преобразование BCD – двоичное число, 8 разрядов
99	COD_32	Преобразование двоичное число – BCD, 8 разрядов

Встроенные функции

FC	Имя	Назначение
81	COD_B4	Преобразование BCD – двоичное число, 4 разряда
82	COD_16	Преобразование двоичное число – BCD, 4 разряда
83	MUL_16	Умножение 16-битных чисел с фиксированной точкой
84	DIV_16	Деление 16-битных чисел с фиксированной точкой

Сигнальные функции

FC	Имя	Назначение
69	MLD_TG	Генератор импульса таймера
70	MLD_TGZ	Генератор импульса таймера с функцией таймера
71	MLD_EZW	Одиночное мигание начального значения размером в слово / Initial value single blinking wordwise
72	MLD_EDW	Двойное мигание начального значения размером в слово / Initial value double blinking wordwise
73	MLD_SAMW	Групповой сигнал размером в слово
74	MLD_SAM	Групповой сигнал
75	MLD_EZ	Одиночное мигание начального значения / Initial value single blinking
76	MLD_ED	Двойное мигание начального значения / Initial value double blinking
77	MLD_EZWK	Меркер одиночного мигания начального значения (размером в слово) / Initial value single blinking (wordwise) memory bit
78	MLD_EZDK	Меркер двойного мигания начального значения (размером в слово) / Initial value double blinking (wordwise) memory bit
79	MLD_EZK	Меркер одиночного мигания начального значения / Initial value single blinking memory bit
80	MLD_EDK	Меркер двойного мигания начального значения / Initial value double blinking memory bit

Аналоговые функции

FC	Имя	Назначение
100	AE_460_1	Модуль аналогового входа 460
101	AI_460_2	Модуль аналогового входа 460
102	AI_463_1	Модуль аналогового входа 463
103	AE_463_2	Модуль аналогового входа 463
104	AE_464_1	Модуль аналогового входа 464
105	AE_464_2	Модуль аналогового входа 464
106	AE_466_1	Модуль аналогового входа 466
107	AE_466_2	Модуль аналогового входа 466
108	RLG_AA1	Модуль аналогового выхода
109	RLG_AA2	Модуль аналогового выхода
110	PER_ET1	Распределенный вход/выход ET 100
111	PER_ET2	Распределенный вход/выход ET 100

Математические функции

FC	Имя	Назначение
112	SINUS	Синус
113	COSINUS	Косинус
114	TANGENS	Тангенс
115	COTANG	Котангенс
116	ARCSIN	Арксинус
117	ARCCOS	Арккосинус
118	ARCTAN	Арктангенс
119	ARCCOT	Арккотангенс
120	LN X	Натуральный логарифм
121	LG X	Логарифм по основанию 10
122	B LOG X	Логарифм по любому основанию
123	E H N	Экспонента по основанию e
124	ZEHN H N	Экспонента по основанию 10
125	A2 H A1	Экспонента по любому основанию

25.5 Блоки преобразования TI-S7

FB	Имя	Назначение
80	LEAD LAG	Алгоритм задержки сигнала (Lead/lag algorithm)
81	DCAT	Прерывание дискретного контрольного времени
82	MCAT	Прерывание времени управления мотором
83	IMC	Сравнение матрицы индексов
84	SMC	Матричный сканер
85	DRUM	Событие маскируемого барабана
86	PACK	Сбор/распределение табличных данных
FC		
Имя	Назначение	
80	TONR	Блокировка задержки включения
81	IBLKMOV	Косвенная передача области данных
82	RSET	Побитовый сброс образа процесса
83	SET	Побитовая установка образа процесса
84	ATT	Ввод значения в таблицу
85	FIFO	Вывод первого значения таблицы
86	TBL_FIND	Поиск значения в таблице
87	LIFO	Вывод последнего значения таблицы
88	TBL	Выполнение операцию с таблицей
89	TBL_WRD	Копирование значения из таблицы
90	WSR	Сохранить элемент данных
91	WRD_TBL	Объединить элемент таблицы
92	SHRB	Сдвиг бита в регистре побитового сдвига
93	SEG	Битовый шаблон для 7-сегментного отображения
94	ATH	Преобразование ASCII – шестнадцатеричная форма
95	HTA	Преобразование шестнадцатеричная форма – ASCII
96	ENCO	Наименьший значащий установленный бит
97	DECO	Установка бита в слове
98	BCDCPL	Генерирование дополнительного кода в десятичной системе
99	BITSUM	Подсчет установленных битов
100	RSETI	Побайтовый сброс PQ
101	SETI	Побайтовая установка PQ
102	DEV	Вычисление стандартного отклонения
103	CDT	Таблицы корреляционных данных
104	TBL_TBL	Комбинирование таблиц
105	SCALE	Значение коэффициентов масштабирования
106	UNSCALE	Значение коэффициентов демасштабирования

25.6 Блоки PID-управления

FB	Имя	Назначение
41	CONT_C	Непрерывное управление
42	CONT_S	Пошаговое управление
43	PULGEN	Генерирование импульса

25.7 Коммуникационные блоки

FC	Имя	Назначение
1	DP_SEND	Передача данных
2	DP_RECV	Прием данных
3	DP_DIAG	Диагностика
4	DP_CTRL	Управление

Содержание главы 26

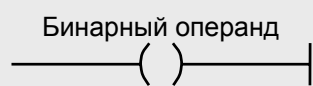
<u>26</u>	<u>Набор функций LAD</u>	4
<u>26.1</u>	<u>Базовые функции</u>	4
<u>26.2</u>	<u>Числовые функции</u>	7
<u>26.3</u>	<u>Управление программным потоком</u>	10

26 Набор функций LAD

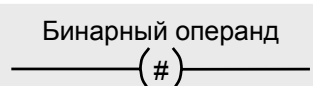
26.1 Базовые функции

Функции для работы с памятью

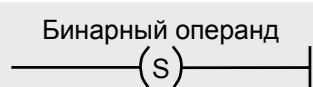
Одиночная катушка



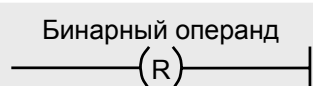
Коннектор



Катушка установки



Катушка сброса



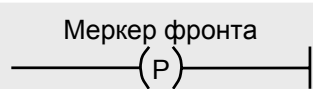
Блочный элемент SR



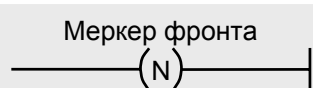
Блочный элемент RS



Положительный фронт
в электрическом токе



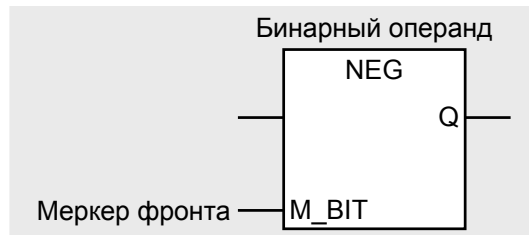
Отрицательный фронт
в электрическом токе



Положительный фронт
в операнде

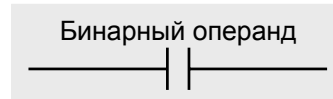


Отрицательный фронт
в операнде



Считывание и комбинирование бинарных значений

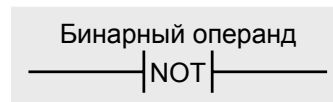
NO-контакт



NC-контакт



NOT-контакт



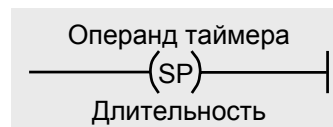
Функции таймера

Блочный элемент
таймера

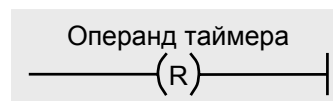


Отдельные элементы

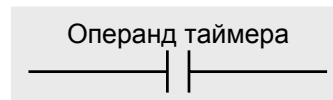
Катушка запуска с
временными характеристиками



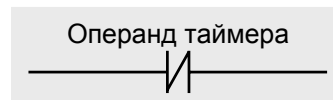
Катушка сброса



NO-контакт



NC-контакт

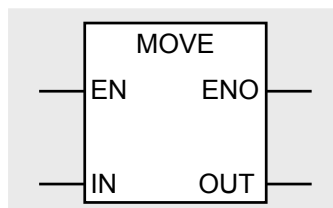


С характеристиками таймера:

S_PULSE	SP	Импульсный
S_PEXT	SE	Расширенный импульсный
S_ODT	SD	Задержка включения
S_ODTS	SS	Задержка включения с запоминанием
S_OFFDT	SF	Задержка выключения

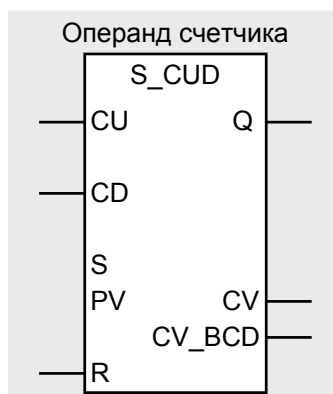
Функции перемещения

Блочный элемент MOVE



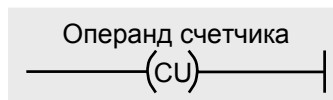
Функции счетчика

Блочный элемент счетчика

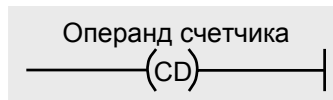


Отдельные элементы

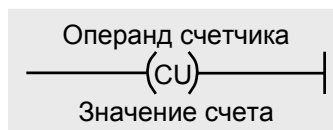
Катушка счета
по возрастанию



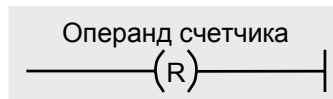
Катушка счета
по убыванию



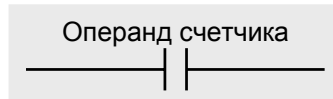
Катушка установки
со значением счета



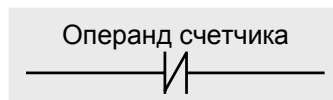
Катушка сброса



NO-контакт



NC-контакт



С характеристиками счетчика:

S_CUD Счетчик прямого/обратного счета

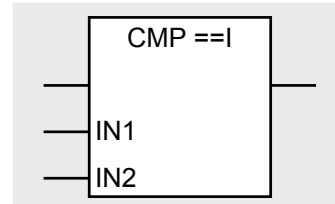
S_CU Счетчик прямого счета

S_CD Счетчик обратного счета

26.2 Числовые функции

Функции сравнения

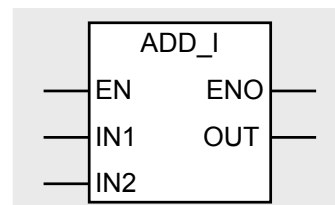
Блочный элемент
сравнения



Сравнение	В соответствии с		
	INT	DINT	REAL
«Равно»	==I	==D	==R
«Не равно»	<>I	<>D	<>R
«Больше»	>I	>D	>R
«Больше или равно»	>=I	>=D	>=R
«Меньше»	<I	<D	<R
«Меньше или равно»	<=I	<=D	<=R

Арифметические функции

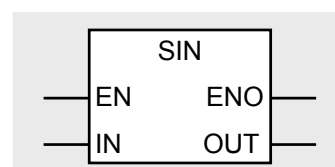
Арифметический
блочный элемент



Операция	В соответствии с		
	INT	DINT	REAL
Сложение	ADD_I	ADD_DI	ADD_R
Вычитание	SUB_I	SUB_DI	SUB_R
Умножение	MUL_I	MUL_DI	MUL_R
Деление с частным в качестве результата	DIV_I	DIV_DI	DIV_R
Деление с остатком в качестве результата	-	MOD_DI	-

Математические функции

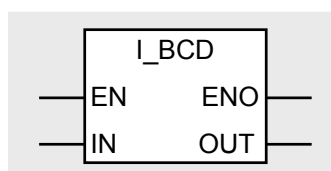
Математический
блочный элемент



SIN	Синус
COS	Косинус
TAN	Тангенс
ASIN	Арксинус
ACOS	Арккосинус
ATAN	Арктангенс
SQR	Возведение в квадрат
SQRT	Извлечение квадратного корня
EXP	Определение экспоненты
LN	Нахождение логарифма

Функции преобразования

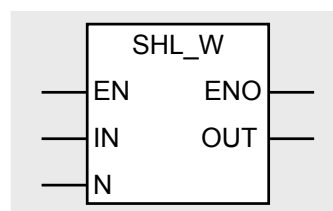
Блочный элемент
преобразования



I_DI	Преобразование INT в DINT
I_BCD	Преобразование INT в BCD
DI_BCD	Преобразование DINT в BCD
DI_R	Преобразование DINT в REAL
BCD_I	Преобразование BCD в INT
BCD_DI	Преобразование BCD в DINT
CEIL	Преобразование REAL в DINT с округлением до следующего большего числа
FLOOR	Преобразование REAL в DINT с округлением до следующего меньшего числа
ROUND	Преобразование REAL в DINT с округлением до следующего целого
TRUNC	Преобразование REAL в DINT без округления
INV_I	Обратный код INT
INV_DI	Обратный код DINT
NEG_I	Инвертирование INT
NEG_DI	Инвертирование DINT
NEG_R	Инвертирование REAL
ABS	Генерирование абсолютного значения REAL

Функции сдвига

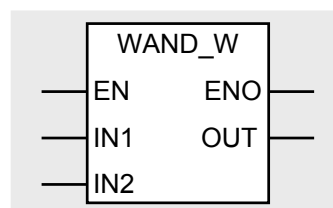
Блочный элемент
сдвига



SHL_W	Сдвиг слова влево
SHL_DW	Сдвиг двойного слова влево
SHR_W	Сдвиг слова вправо
SHR_DW	Сдвиг двойного слова вправо
SHR_I	Сдвиг слова со знаком
SHR_DI	Сдвиг двойного слова со знаком
ROL_DW	Циклический сдвиг влево
ROR_DW	Циклический сдвиг вправо

Побитовые логические операции

Блочный элемент
побитовой логической операции






WAND_W	AND с переменными размером в слово
WOR_W	OR с переменными размером в слово
WXOR_W	Исключающее OR с переменными размером в слово
WAND_DW	AND с переменными размером в двойное слово
WOR_DW	OR с переменными размером в двойное слово
WXOR_DW	Исключающее OR с переменными размером в двойное слово

26.3 Управление программным потоком

Биты состояния

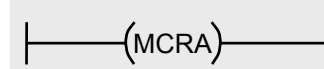
Результат больше нуля	
Результат больше или равен нулю	
Результат меньше нуля	
Результат меньше или равен нулю	
Результат не равен нулю	
Результат равен нулю	
Результат недействителен (неупорядочен)	
Выход за границы области значений числа (переполнение)	
Сохраненное переполнение	
Бинарный результат	
Катушка SAVE (сохранение)	

Функции перехода

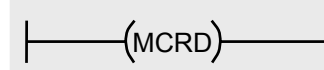
Переход, если RLO = «1»	
Переход, если RLO = «0»	
Точка назначения перехода	

Главное реле управления (MCR)

Активирование
MCR-области



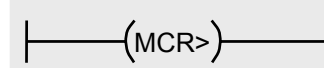
Деактивирование
MCR-области



Открытие
MCR-зоны

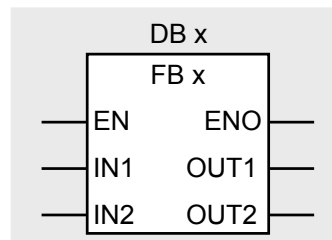


Закрытие
MCR-зоны

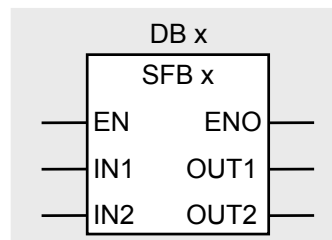


Функции для работы с блоками

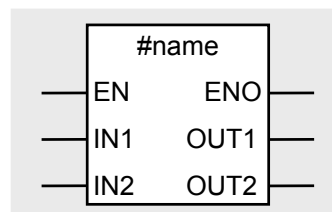
Вызов функционального блока
(с блоком данных)



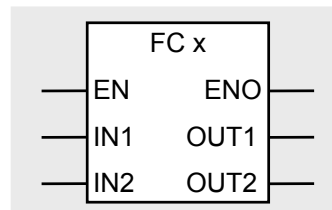
Вызов системного функционального
блока (с блоком данных)



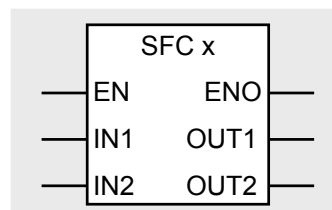
Вызов функционального блока или
системного функционального блока
(как локальный экземпляр)



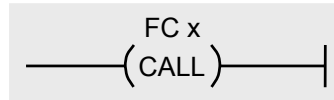
Вызов функции



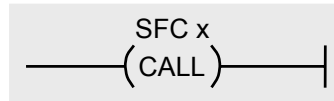
Вызов системной функции



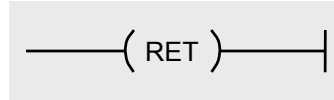
Вызов функции без параметров



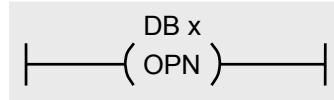
Вызов системной функции без параметров



Катушка RET
(условное завершение блока)



Открытие блока данных



Содержание главы 27

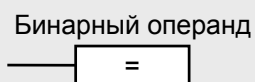
<u>27</u>	<u>Набор функций FBD</u>	4
<u>27.1</u>	<u>Базовые функции</u>	4
<u>27.2</u>	<u>Числовые функции</u>	7
<u>27.3</u>	<u>Управление программным потоком</u>	10

27 Набор функций FBD

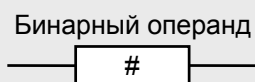
27.1 Базовые функции

Функции для работы с памятью

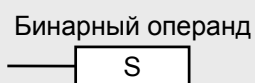
Назначение



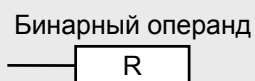
Коннектор



Установка



Сброс



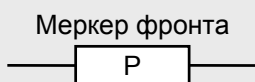
Блочный элемент SR



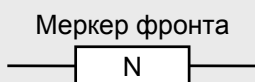
Блочный элемент RS



Положительный фронт
результата RLO



Отрицательный фронт
результата RLO



Положительный фронт
в операнде

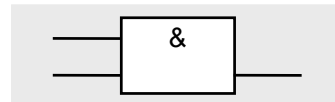


Отрицательный фронт
в операнде

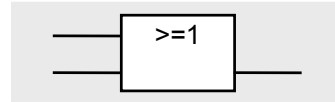


Считывание и комбинирование бинарных значений

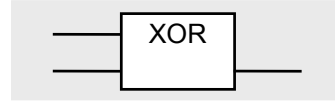
Функция AND



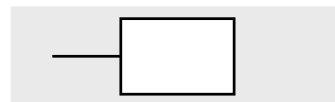
Функция OR



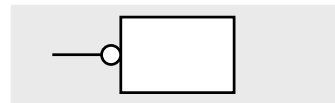
Функция исключающее OR



Считывание сигнального состояния «1»



Считывание сигнального состояния «0»



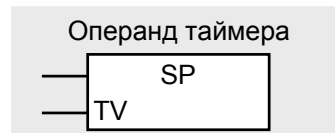
Функции таймера

Блочный элемент таймера

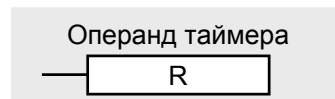


Отдельные элементы

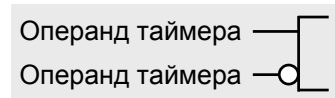
Блочный элемент запуска с характеристиками таймера



Блочный элемент сброса



Опрос состояния таймера

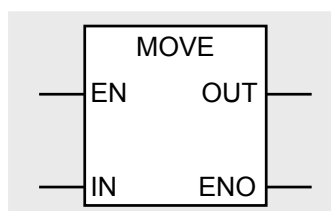


С характеристиками таймера:

S_PULSE	SP	Импульсный
S_PEXT	SE	Расширенный импульсный
S_ODT	SD	Задержка включения
S_ODTS	SS	Задержка включения с запоминанием
S_OFFDT	SF	Задержка выключения

Функции перемещения

Блочный элемент MOVE



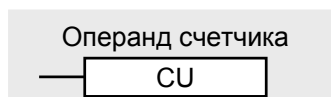
Функции счетчика

Блочный элемент счетчика



Отдельные элементы

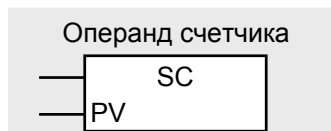
Блочный элемент счета по возрастанию



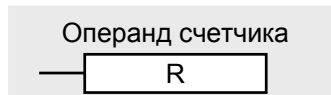
Блочный элемент счета по убыванию



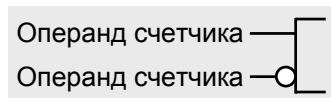
Блочный элемент установки со значением счета



Блочный элемент сброса



Опрос состояния счетчика



С характеристиками счетчика:

S_CUD Счетчик прямого/обратного счета

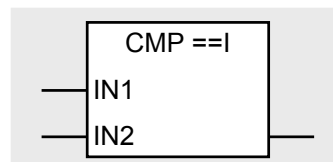
S_CU Счетчик прямого счета

S_CD Счетчик обратного счета

27.2 Числовые функции

Функции сравнения

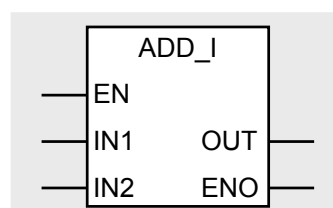
Блочный элемент
сравнения



Сравнение	В соответствии с		
	INT	DINT	REAL
«Равно»	==I	==D	==R
«Не равно»	<>I	<>D	<>R
«Больше»	>I	>D	>R
«Больше или равно»	>=I	>=D	>=R
«Меньше»	<I	<D	<R
«Меньше или равно»	<=I	<=D	<=R

Арифметические функции

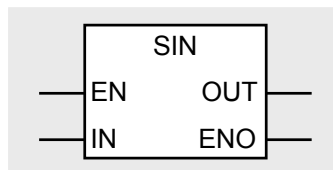
Арифметический
блочный элемент



Операция	В соответствии с		
	INT	DINT	REAL
Сложение	ADD_I	ADD_DI	ADD_R
Вычитание	SUB_I	SUB_DI	SUB_R
Умножение	MUL_I	MUL_DI	MUL_R
Деление с частным в качестве результата	DIV_I	DIV_DI	DIV_R
Деление с остатком в качестве результата	-	MOD_DI	-

Математические функции

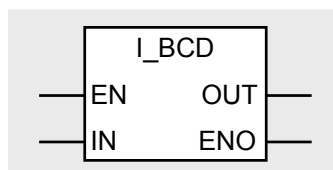
Математический
блочный элемент



SIN	Синус
COS	Косинус
TAN	Тангенс
ASIN	Арксинус
ACOS	Арккосинус
ATAN	Арктангенс
SQR	Возведение в квадрат
SQRT	Извлечение квадратного корня
EXP	Определение экспоненты
LN	Нахождение логарифма

Функции преобразования

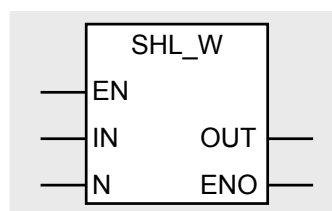
Блочный элемент
преобразования



I_DI	Преобразование INT в DINT
I_BCD	Преобразование INT в BCD
DI_BCD	Преобразование DINT в BCD
DI_R	Преобразование DINT в REAL
BCD_I	Преобразование BCD в INT
BCD_DI	Преобразование BCD в DINT
CEIL	Преобразование REAL в DINT с округлением до следующего большего числа
FLOOR	до следующего меньшего числа
ROUND	до следующего целого
TRUNC	без округления
INV_I	Обратный код INT
INV_DI	Обратный код DINT
NEG_I	Инвертирование INT
NEG_DI	Инвертирование DINT
NEG_R	Инвертирование REAL
ABS	Генерирование абсолютного значения REAL

Функции сдвига

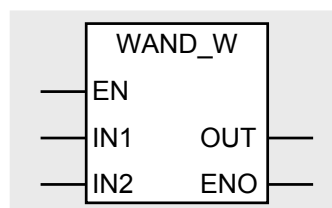
Блочный элемент
сдвига



SHL_W	Сдвиг слова влево
SHL_DW	Сдвиг двойного слова влево
SHR_W	Сдвиг слова вправо
SHR_DW	Сдвиг двойного слова вправо
SHR_I	Сдвиг слова со знаком
SHR_DI	Сдвиг двойного слова со знаком
ROL_DW	Циклический сдвиг влево
ROR_DW	Циклический сдвиг вправо

Побитовые логические операции

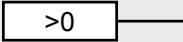
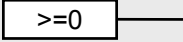
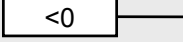
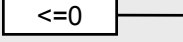
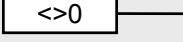
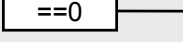


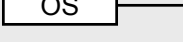


Блочный элемент
побитовой логической операции



WAND_W	AND с переменными размером в слово
WOR_W	OR с переменными размером в слово
WXOR_W	Исключающее OR с переменными размером в слово
WAND_DW	AND с переменными размером в двойное слово
WOR_DW	OR с переменными размером в двойное слово
WXOR_DW	Исключающее OR с переменными размером в двойное слово

27.3 Управление программным потоком

Биты состояния

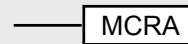
Результат больше нуля	
Результат больше или равен нулю	
Результат меньше нуля	
Результат меньше или равен нулю	
Результат не равен нулю	
Результат равен нулю	
Результат недействителен (неупорядочен)	
Выход за границы области значений числа (переполнение)	
Сохраненное переполнение	
Чтение бинарного результата BR	
Назначить бинарный результат BR	

Функции перехода

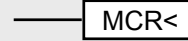
Переход, если RLO = «1»	
Переход, если RLO = «0»	
Точка назначения перехода	

Главное реле управления (MCR)

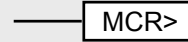
Активирование
MCR-области



Открытие
MCR-зоны



Закрытие
MCR-зоны

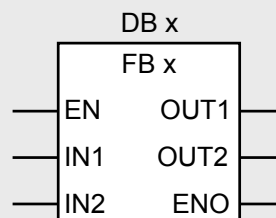


Деактивирование
MCR-области

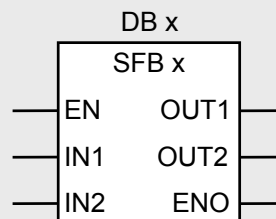


Функции для работы с блоками

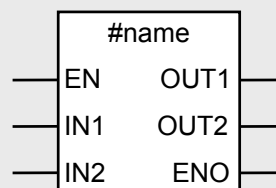
Вызов функционального блока
(с блоком данных)



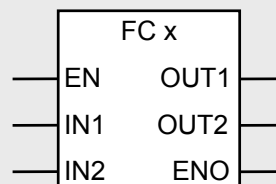
Вызов системного функционального
блока (с блоком данных)



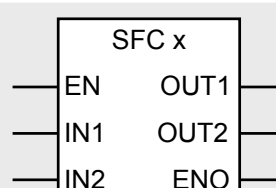
Вызов функционального блока или
системного функционального блока
(как локальный экземпляр)



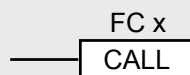
Вызов функции



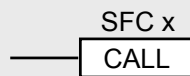
Вызов системной функции



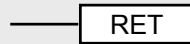
Вызов функции без параметров



Вызов системной функции без параметров



Условное завершение блока



Открытие блока данных

