

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.

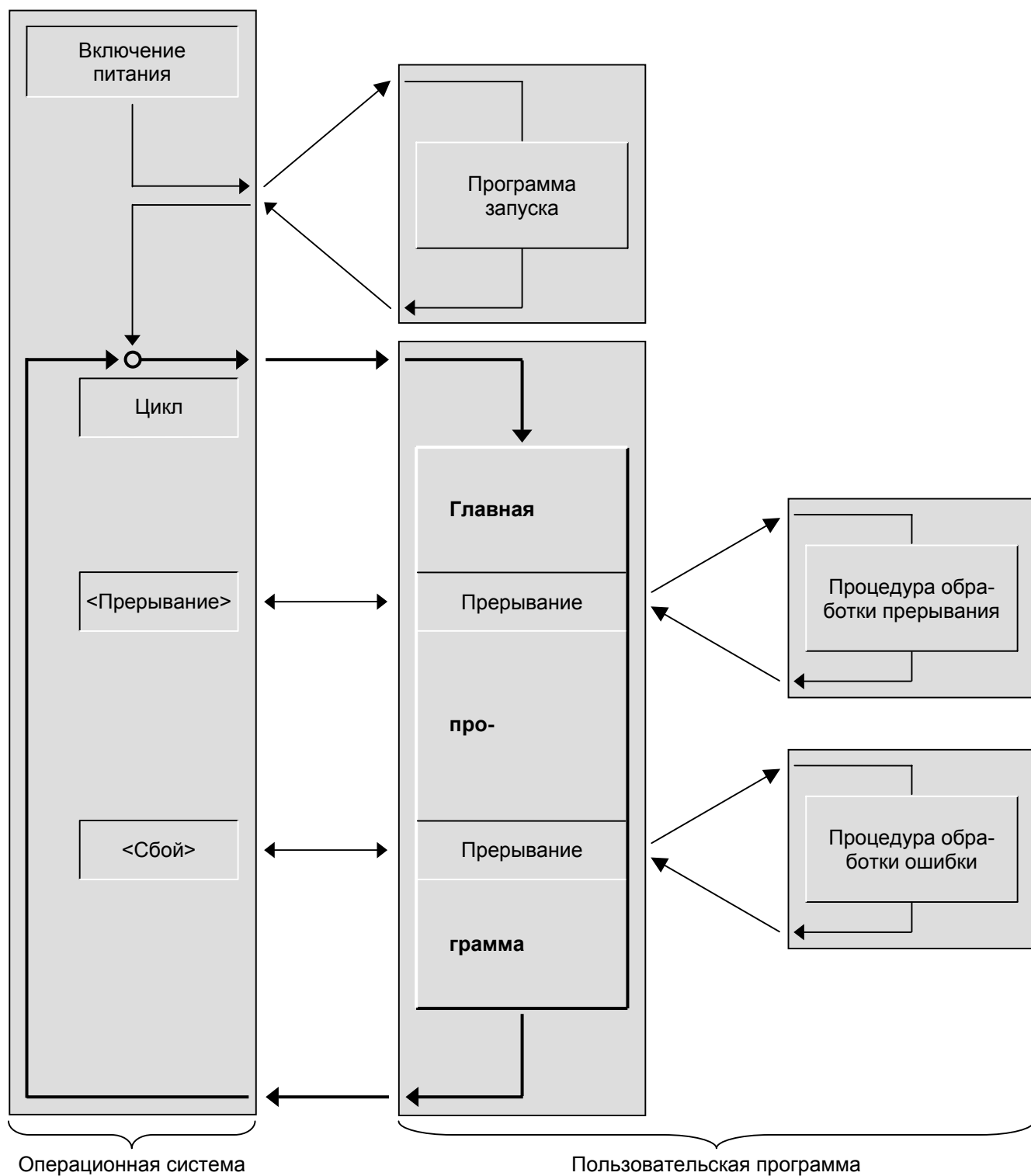


Рисунок 3.1 Методы обработки пользовательской программы

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. Незадействованным приоритетным классам (организацион-

ным блокам) должен быть присвоен нулевой приоритет. Для всех используемых приоритетных классов программируются соответствующие организационные блоки; в противном случае CPU вызовет ОВ 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 (Apparatur)*), 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

Block header (Заголовок блока)

Declaration (Описание)

Адрес	Описание	Имя	Тип	

Program (Программа)

Логический блок, программирование, ориентированное на создание исходных текстов

Block type Address (Адрес блока)
Block header

Var_xxx

name: Data type := Initialization;
name: Data type := Initialization;
...

END_VAR

BEGIN

Program

END_Block Type

Блок данных, пошаговое программирование

Block header

Declaration

Адрес	Имя	Тип	Начальное значение	

Блок данных, программирование, ориентированное на создание исходных текстов

DATA_BLOCK Address (Адрес блока)
Block header

STRUCT

name: Data type := Initialization;
name: Data type := Initialization;
...

END_STRUCT

BEGIN

name := Initialization;

END_DATA_BLOCK

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.

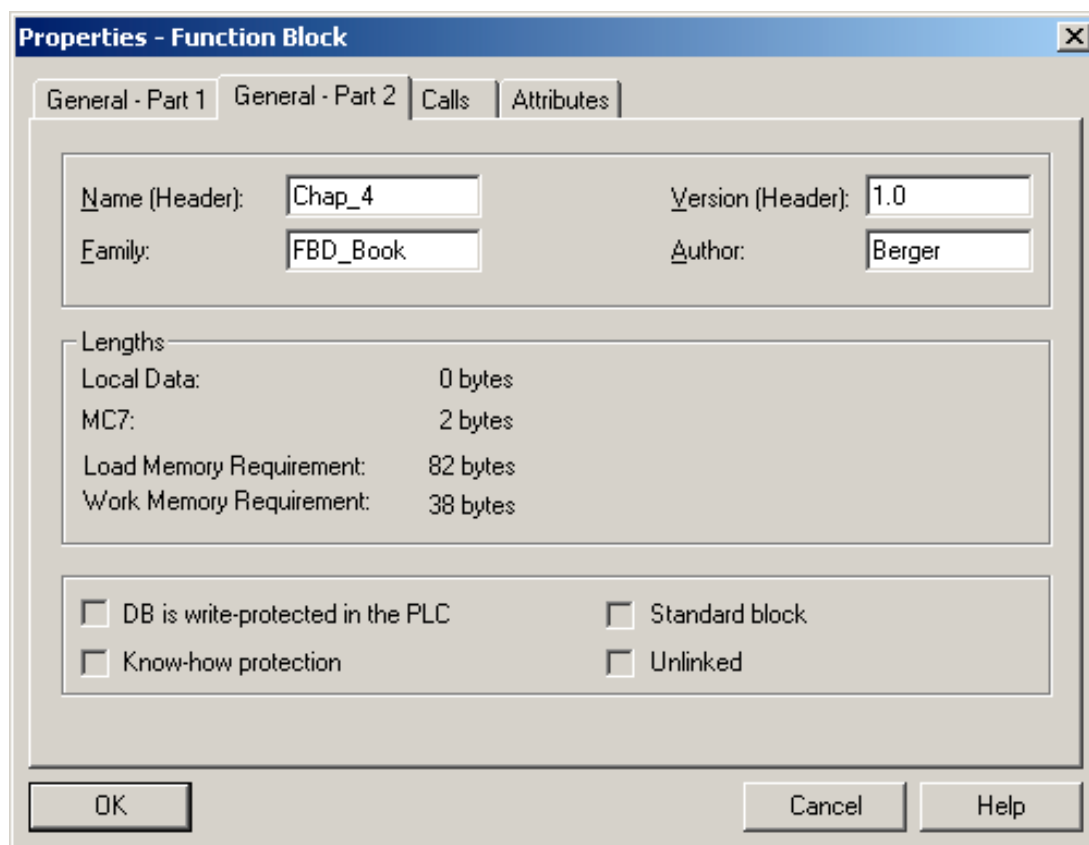


Рисунок 3.3 Свойства блока

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.

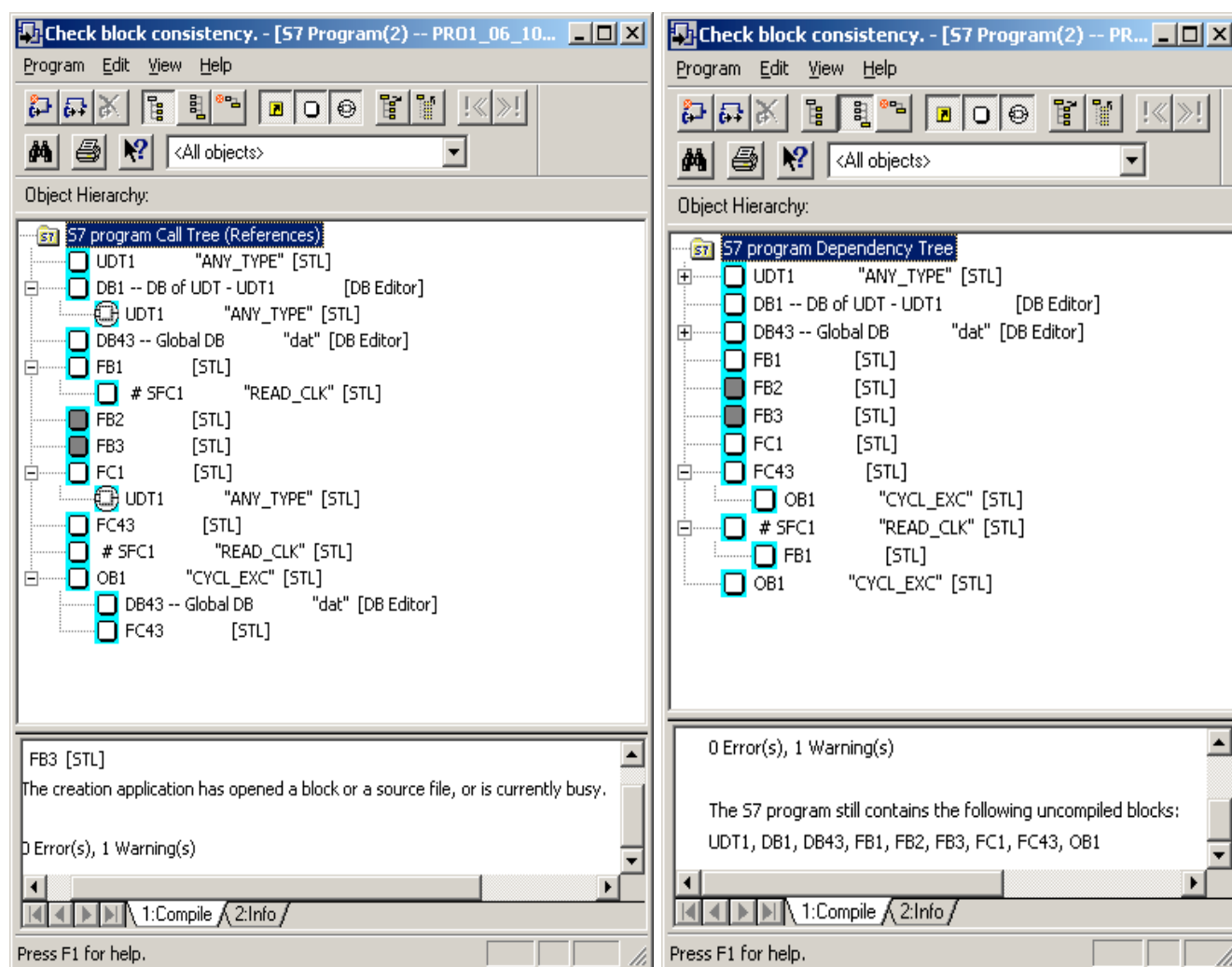


Рисунок 3.4 Пример представления зависимостей в окне проверки согласованности блоков

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.

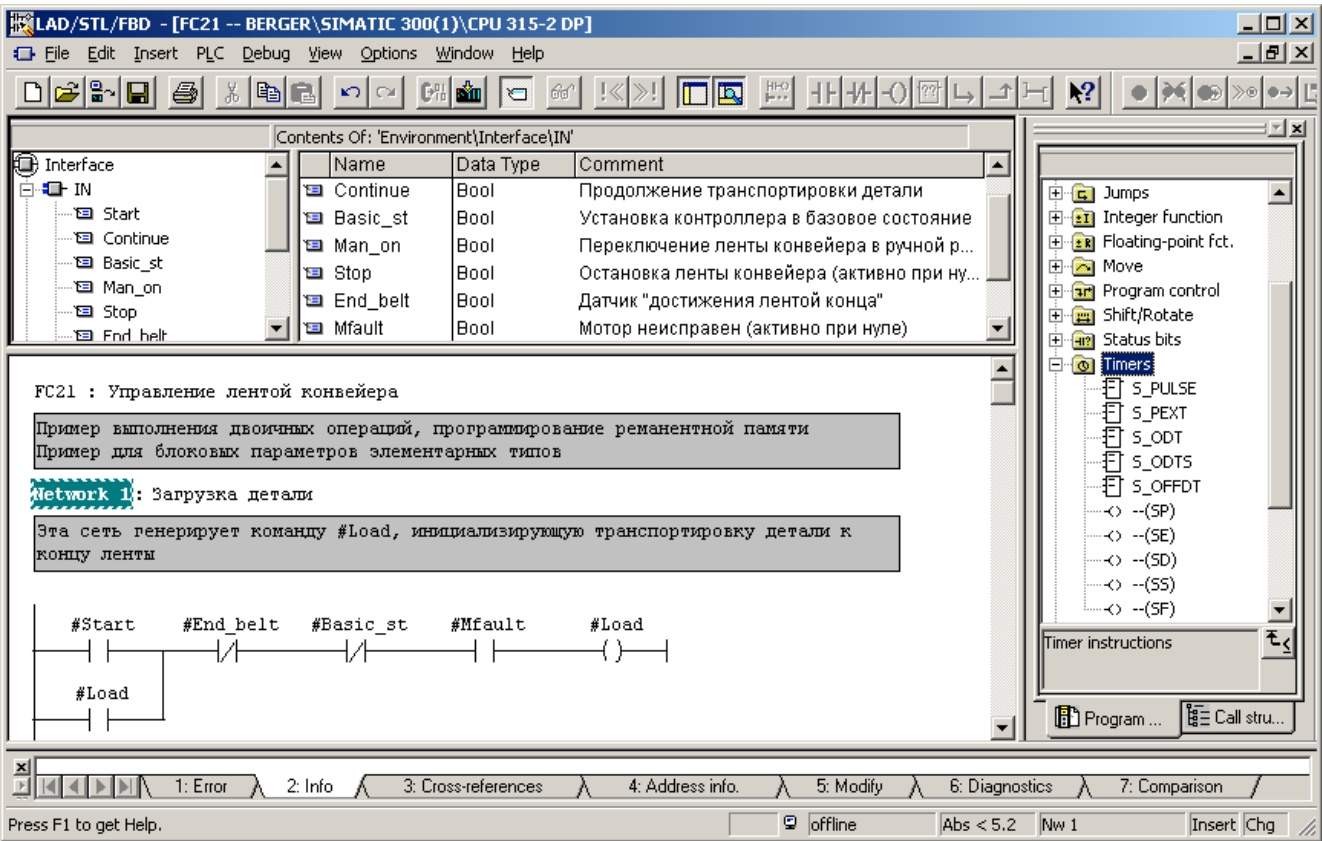


Рисунок 3.5 Пример открытого блока LAD

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 дается 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).

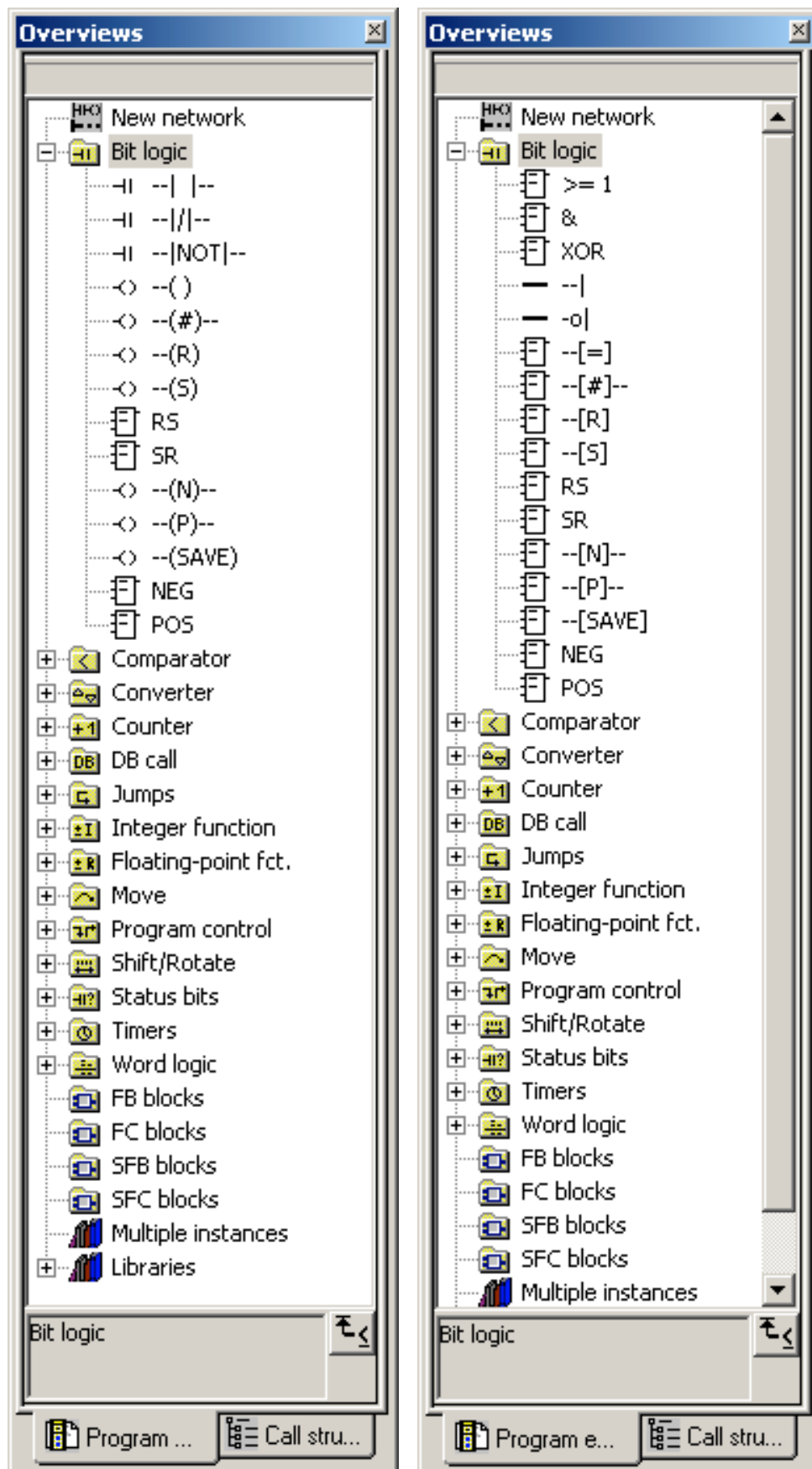


Рисунок 3.6 Каталог программных элементов для LAD и FBD

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 закрытый 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.

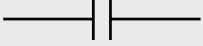
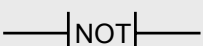
Прямоугольные блочные элементы (Boxes)

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

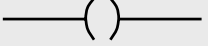
- без механизма EN/ENO, например, функции работы с памятью, функции таймера и счетчика, блочные элементы с функцией сравнения;
- с EN/ENO, например, MOVE (Переместить), арифметические и математические функции, преобразование типов данных.

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)

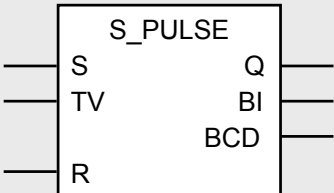
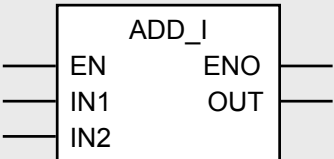
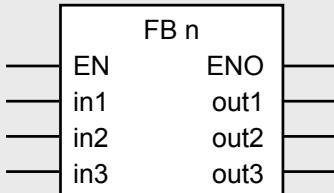
Стандартные прямоугольные блочные элементы без EN/ENO (например, функции таймера и счетчика)	Стандартные прямоугольные блочные элементы с EN/ENO (например, арифметические функции)	Прямоугольные блочные элементы блоков (например, вызовы функциональных блоков)
Адрес таймера 	ADD_I 	DB m 

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.

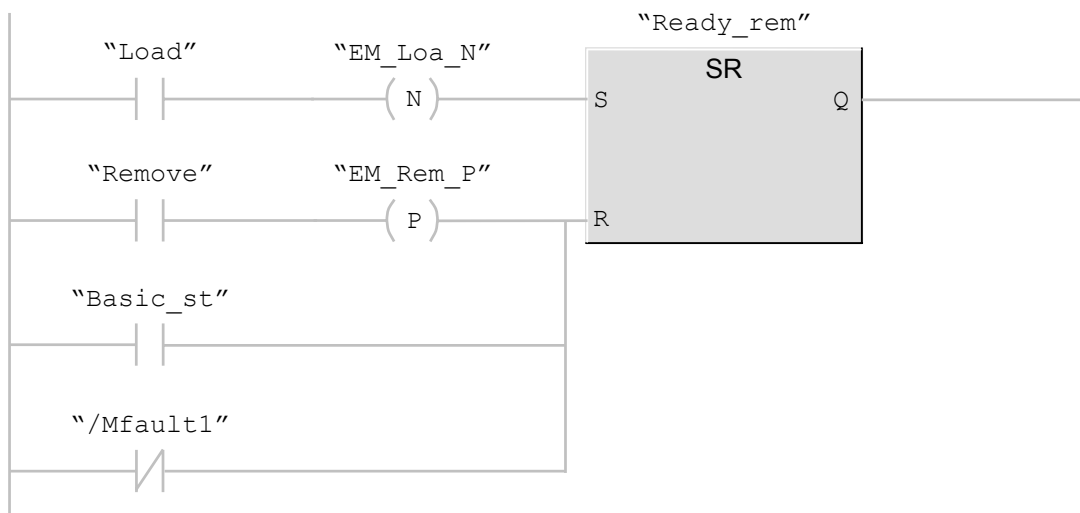


Рисунок 3.8 Пример сегмента LAD в «трехмерном» представлении

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).

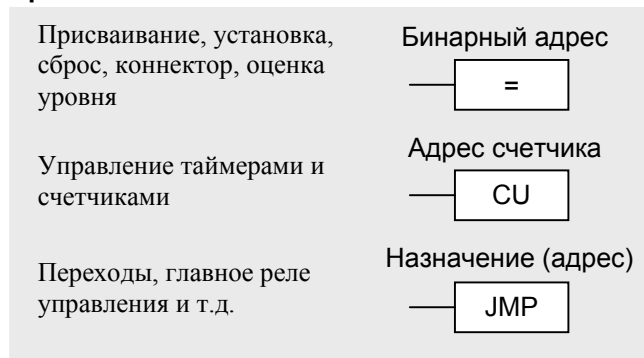
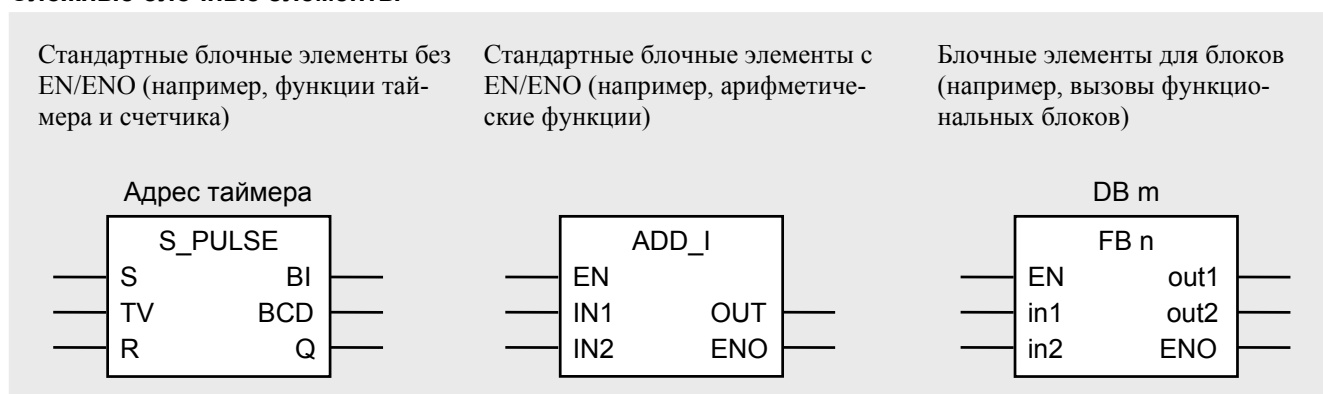
Завершается бинарная логическая операция в простейшем случае блочным элементом присваивания (assign box). Об этом рассказывается в текущем параграфе 3.3 «Программирование кодовых блоков».

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.

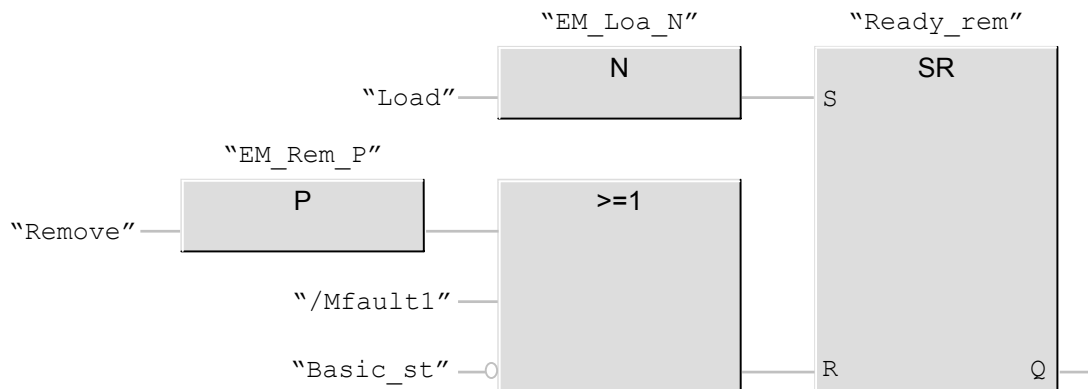


Рисунок 3.10 Пример сегмента FBD в «трехмерном» представлении

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 буде 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.

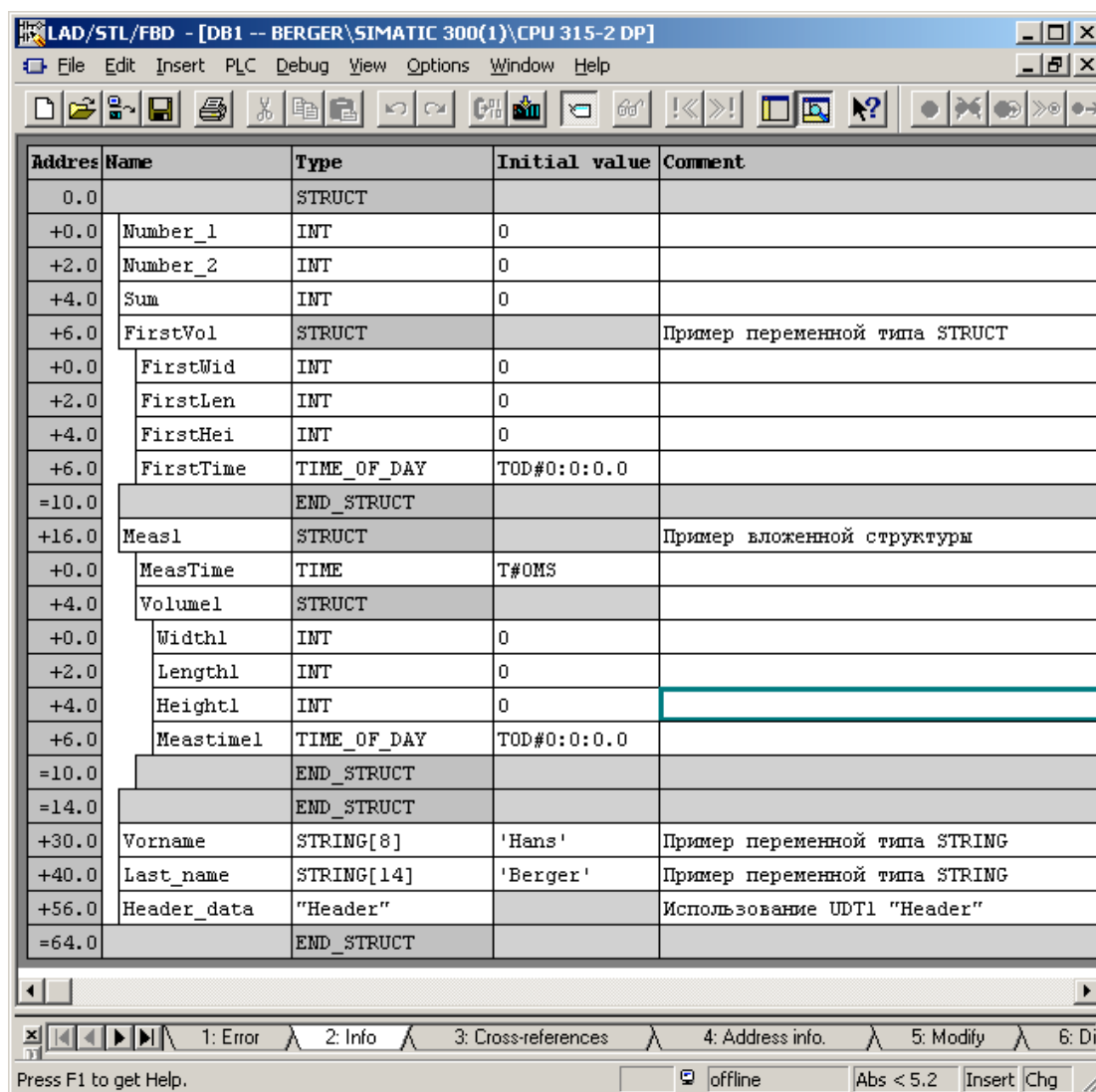


Рисунок 3.11 Пример открытого блока данных (окно просмотра описаний)

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.

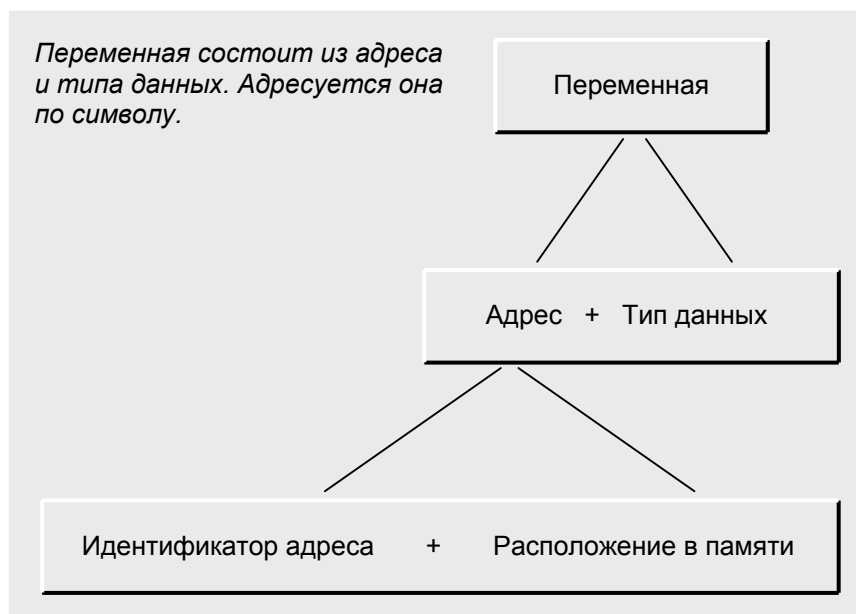


Рисунок 3.12 Структура переменной

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 (внутри-блочными) 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).

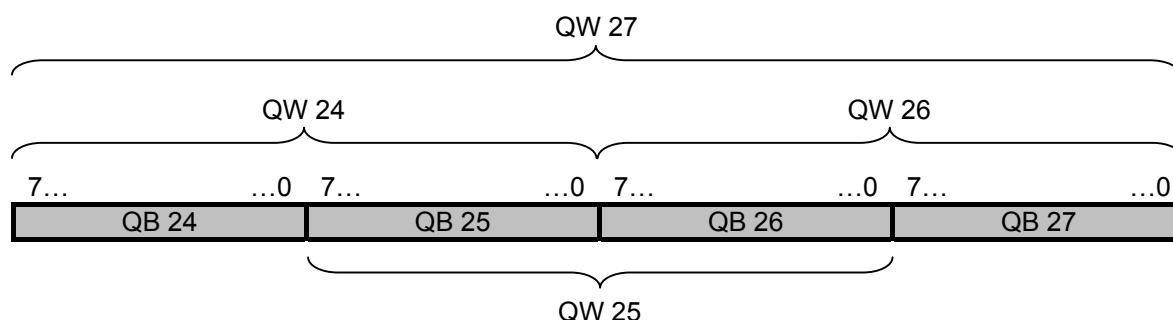


Рисунок 3.13 Содержимое байтов в машинных словах и двойных словах

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, импортирование и экспортирование 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 предопределенные 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 внутреннее представление числа с плавающей точкой.

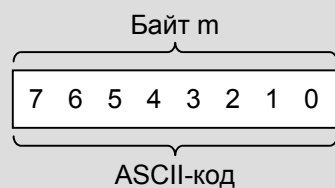
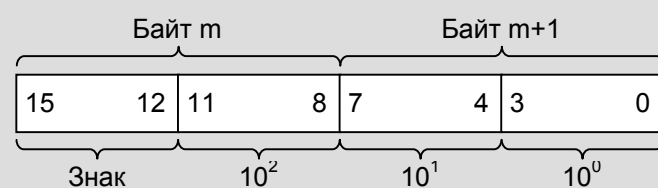
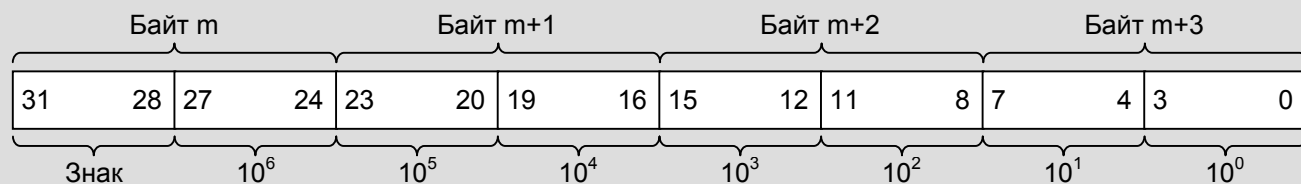
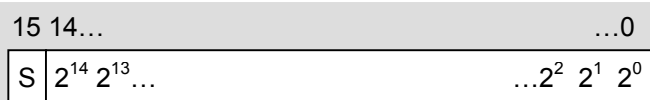
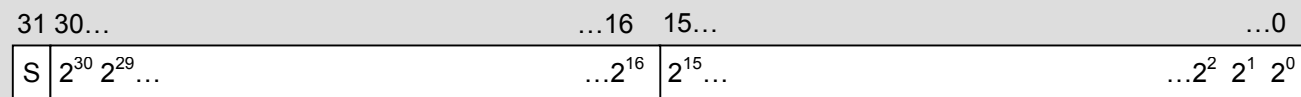
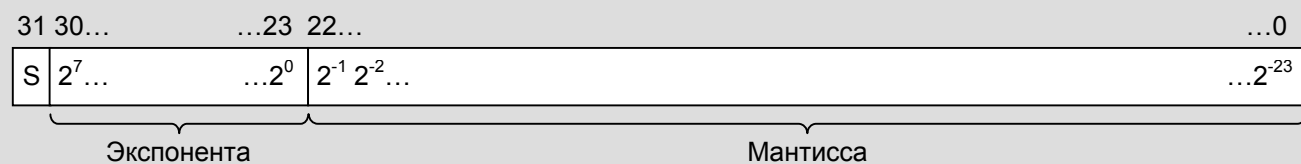
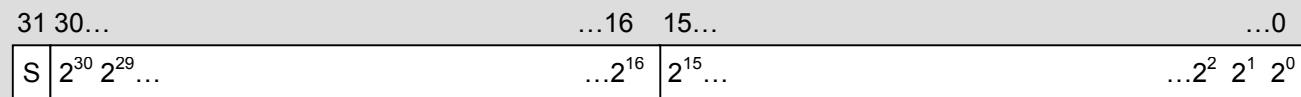
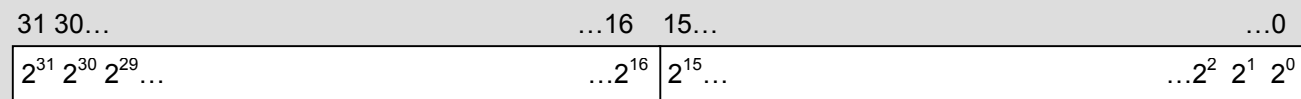
Переменные типа REAL разделяются на числа, которые могут быть представлены с полной точностью («нормализованные» числа с плавающей точкой) и ограниченной точностью («ненормализованные» числа с плавающей точкой). Область значений нормализованных чисел с плавающей точкой:

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).

Тип данных CHAR**BCD-число, 3 декады****BCD-число, 7 декад****Тип данных INT****Тип данных DINT****Тип данных REAL****Тип данных S5TIME****Тип данных DATE****Тип данных TIME****Тип данных TIME_OF_DAY****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 внутреннее 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 од- нотипных элементов.

В квадратных скобках после наименования типа ARRAY вы должны задать область индексов поля. Начальное значение слева должно быть меньше или равно завер- шающему значению справа. Оба индекса являются числами типа INT из области значений -32 768 ... +32 767. Наибольшая размерность поля – 6, границы «измере- ний» разделяются запятой.

Тип данных отдельного компонента поля располагается в строке под определением типа данных ARRAY. Допускаются все типы данных за исключением ARRAY; так- же это может быть пользовательский тип данных.

Предустановка

На этапе описания вы можете предустановить значения отдельных компонентов по- ля (не как в случае параметров блока в функции, входных/выходных параметров в функциональном блоке или временной переменной). Тип данных предустанавливае- мого 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 ком- понентов не должно превышать числа компонентов поля. Предустановленные зна- чения 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.