МИНИСТЕРСТВО ВЫСШЕГО И СРЕДНЕГО СПЕЦИАЛЬНОГО
ОБРАЗОВАНИЯ РЕСПУБЛИКИ УЗБЕКИСТАН

ТАШКЕНТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АБУ РАЙХАНА БЕРУНИ

ПРИМЕНЕНИЕ ПРОГРАММИРУЕМЫХ ЛОГИЧЕСКИХ КОНТРОЛЛЕРОВ *SIMATIC S7-200* К РЕШЕНИЮ ЗАДАЧ ЛОГИЧЕСКОГО УПРАВЛЕНИЯ

Учебное пособие

Ташкент 2008

УДК 621:521.57 (076)

Н.Р. Юсупбеков, Ш.М. Гулямов, У.Т.Мухамедхонов, П.М. Матякубова, О.А. Рахманов. Применение программируемых логических контроллеров Simatic к решению задач логического управления: Учебное пособие, Ташкент, ТашГТУ, 2008.

Данное учебное пособие содержит основы программирования на языке "Step-7", базовые алгоритмы управления. Оно поможет синтезировать технологические автоматы управления, освоить программирование основных функций программы пользователя и научит применять программируемые логические контроллеры "Simatic S7-200" к решению задач логического управления.

Пособие рассчитано на студентов бакалавриатуры специальности 5А5211802 – «Автоматизация технологических процессов и производств», магистратуры, аспирантуры, преподавателей высших учебных заведений, слушателей курсов подготовки, переподготовки и повышения квалификации, а также на специалистов в области автоматизации технологических процессов и производств.

Печатается по решению научно-методического совета Ташкентского государственного технического университета

Рецензенты: д.т.н., проф. Исмаилов М.А. (Институт математики и информационных технологий АН РУз);
д.т.н. Адилов Ф.Т. (ОАО «Химавтоматика»).

ВВЕДЕНИЕ

Практика создания программного обеспечения подразумевает поэтапный переход от математического обеспечения через информационное к программному. Все три вида обеспечения взаимосвязаны между собой.

Математическое обеспечение - основное формализованное представление требований технического задания в отношении управления. Этот вид обеспечения требует для своей разработки наличия технического задания.

Информационное обеспечение - своеобразный каталог данных, используемых при контроле и управлении. Оно создается на основе технического задания и дополнительной информации, содержащейся в математическом обеспечении.

Программное обеспечение - реализация всех функций системы управления (в том числе и основных, логика реализации которых заложена в математическом обеспечении). Большинство используемых программных переменных создается на основе материалов информационного обеспечения.

Процесс проектирования должен протекать последовательно.

Предлагаемое учебное пособие охватывает все этапы проектирования математического, информационного и программного обеспечения *локальных систем управления*.

Учебное пособие предназначено для специалистов, работающих с контроллерами **SIMATIC** и при написании его был использован опыт разработки систем рассматриваемого класса

ГЛАВА 1
ГИБКО ПРОГРАММИРУЕМАЯ СИСТЕМА УПРАВЛЕНИЯ SIMATIC

1.1. Общее представление о SIMATIC

Современная автоматизация - сплошь программируемая. Поэтому ее нельзя отделить от базового программного обеспечения. Разработчику прикладных пакетов сегодня нужны средства, с которыми можно сразу решать поставленную задачу. Просто, быстро и экономично. Базовое программное обеспечение от фирмы SIEMENS - это большой, своеобразный "инструментальный ящик". В нем можно найти набор готовых программных инструментов в соответствии с решаемой в данный момент задачей. К примеру, инструмент с названием "STEP-5" необходим для работы с контроллерами.

SIMATIC - семейство гибко программируемых систем управления, состоящее из множества совместимых друг с другом компонентов. В него входят: контроллеры, программаторы, модули интеллектуальной периферии. Каждая из этих групп включает в себя целый ряд отдельных компонентов.

Разработаны также совместимые с семейством **SIMATIC** системы обслуживания и визуализации и открытые средства коммуникации. На этой основе можно подобрать индивидуальное решение к любой задаче автоматизации Подобное разделение по функциональным уровням является сильной стороной семейства **SIMATIC**.

1.2. Контроллер S5-90

Контроллер S5-90U является одним из малых систем управления, но в то же самое время у него есть все, что необходимо для решения простых задач автоматизации. Систему, состоящую из четырех реле, экономичнее заменить на программируемый контроллер S5-90U, поскольку ему не нужен логический монтаж, а только затраты на однократное программирование. У него есть восемь цифровых входов, шесть релейных выходов, по одному аварийному сигналу тревоги и счета. Процессор предоставляет пользователю 4 Кбайта памяти и скорость обработки 2 мс на 1000 операций.

Питание:

Включается в сеть переменного тока 115/230V, кроме того, самостоятельно осуществляется питание датчиков. Это означает, что контроллер не нуждается в отдельной системе питания.

S5-90U можно соединять с другими контроллерами и программаторами через шину SINEC LI.

Расширение:

S5-90U можно наращивать максимально шестью периферийными модулями, S5-100U, используя модуль подключения IM 90.

Применение:

- в малом машиностроении;
- в пекарнях для замеса теста;
- при сортировке почты;
- для контроля веса на цементных заводах;
- при складировании в штабели грузов и т. д.

Программаторы:

P0710, PO720 и т. д.

Для программирования применяется язык STEP 5, который, благодаря различным формам представления, позволяет каждому пользователю найти для себя собственный подход.

Возможности:

- AWL - командный план, когда все функции программируются с помощью легко запоминающихся сокращений;
- FUP - функциональный план - используется, если пользователю привычны логические схемы;
- KOP - контактный план - используется в том случае, если пользователь чаще работает с электрическими схемами.

1.3. Контроллер S5-95U

Другим представителем малого класса контроллеров является S5-95U.

Он компактен и универсален, так как у него имеется 16 цифровых входов и выходов, а также 8 аналоговых входов и один аналоговый выход, 4 сигнала тревоги и 2 быстрого счета. Процессор

предоставляет 8 Кбайт памяти для кода программ и такой же объем для данных. Быстродействие: 1000 команд за 2 миллисекунды.

Расширение:

Контроллер может быть надстроен периферийными модулями S5-100U на:

- 256 - цифровых вх/вых сигналов;
- 16 - аналоговых вх/вых сигналов.

S5-95U открыт для соединения с другими контроллерами, программаторами и сетями SINEC L1/L2.

S5-95F - это двухканальная программируемая отказоустойчивая система. Обе подсистемы устройства размещены в пластмассовых корпусах, каждый содержит процессор, блок питания, входы и выходы. Кроме того, S5-95F можно расширить модулями высокой надежности для приоритетных сигналов.

Программирование:

85-95P-программируется с помощью языка STEP-5. Пользователь составляет свою программу без учета резервирования и испытательных сигналов.

1.4. Контроллер S5-115U

Контроллер состоит из различных функциональных устройств, которые можно комбинировать в зависимости от поставленной задачи. Возможность выбора между 5 различными процессами позволяет обеспечить идеальное соответствие функциональному назначению. Таким образом, стандартного устройства S5-115U не существует. Его конфигурация всегда подбирается так, как того требует конкретная задача.

Применение:

- текстильная промышленность;
- производство пластмасс;
- производство напитков;
- водное хозяйство;
- металлургическая промышленность;
- пищевая и обрабатывающая промышленность;
- химическая промышленность;
- развлекательные аттракционы.

Это - отрасли, где требуется исключительно высокая скорость обработки информации. Все это можно осуществить благодаря CPU945 - он выполняет почти все команды, даже при работе с таймером, требующим выполнения счетных и арифметических операций со скоростью 0,1 мкс.

1.5. Системы повышенной надежности S5 -115H/F

Программируемые контроллеры S5 -115U поставляются также как системы повышенной надежности. Это - либо системы с повышенным коэффициентом надежности (H - системы: S5 - 115H), либо системы с повышенной защитой от ошибок (F - системы; S5 - 115 F). Все системы с повышенной надежностью состоят из двух центральных блоков 115U со специальными CPU, связанными друг с другом. Пользователь составляет свою программу только с учетом требований, предъявляемых к процессу и без учета повышенной надежности и контрольных сигналов.

S5 -115U - состоит из центрального блока (с носителем модулей CR700) и, если требуется, с устройством расширения (EG с носителем модулей ER701). Центральный блок всегда оснащен блоком питания и центральным процессором. В зависимости от задачи устанавливаются различные периферийные модули.

1.6. Модули центральных процессоров

Модули центральных процессоров (CPU) являются мозгом программируемых контроллеров. Они выполняют программу управления. В зависимости от мощности задачи управления, которая стоит перед SIMATIC S5-115U, можно выбрать один из пяти центральных процессоров: CP941, CP942, CP943, CP944 или CP945.

Мощность выбираемого процессора зависит от необходимой скорости выполнения программы и от объема памяти. С помощью процессора 942, 943 и 944 можно работать с аналоговыми модулями и стандартными пакетами программ регулирования, так как в операционную систему этих процессоров встроен алгоритм ПИД - регуляторов. Возможное время опроса регулируемого контура - от 100 мс. Можно реализовать максимум 8 контуров регулирования.

1.7. Модули входов и выходов

Модули входов и выходов являются интерфейсами, узлами связи датчиков и исполнительных механизмов с системой управления. Модули входов и выходов программного контроллера S5-115U дают пользователю возможность удобного управления с помощью быстрого монтажа, механического кодирования и достаточного места для надписей на разъемах модулей.

1.8. Цифровые модули

Разработаны модули, которые соответствуют уровням напряжения и токов вашей установки. Таким образом, не вы должны предоставлять контроллеру необходимые уровни, а S5-115U подстраивается под уровни напряжения и токов вашей машины или станка.

Цифровые модули с помощью удобной техники подключения предоставляют возможность подключения сигнальных кабелей через передний разъем и две возможности подключения на выбор либо винтовые зажимы с защелками (CRIMP - контакты).

1.9. Аналоговые модули

Чем мощнее программируемый контроллер, тем большее значение приобретает в нем обработка аналоговых сигналов. Установка аналоговых модулей необходима при решении задач регулирования, например, регулирования уровня, температуры или числа оборотов. SIMATIC S5-115U предоставляет возможность работы с двумя основными типами аналоговых модулей входов — потенциально связанными и потенциально изолированными. Они подключаются на требуемый уровень сигнала через субмодуль пределов измерения. Для четырех каналов необходим 1 субмодуль. Это означает, что число каналов одного модуля позволяет реализовать до четырех различных областей измерения напряжения или токов аналоговых звеньев управления, которые перекрываются тремя различными модулями аналоговых выходов.

1.10. Модули быстрого счета импульсов

Счет быстрых последовательностей импульсов, получение и обработка инкрементов пути, измерение скорости и времени, регулирование и позиционирование, а также многие другие задачи являются критичными по времени: они не могут достаточно быстро решаться центральным процессором контроллера вместе с основной задачей управления. Поэтому в S5-115U можно использовать модули предварительной обработки сигналов (IP). При этом задачи измерения, регулирования и управления будут решаться параллельно основной программе и гораздо быстрее. Эти модули в основном имеют отдельный собственный процессор и поэтому могут решать задачи самостоятельно. Всем этим модулям свойственна высокая скорость обработки, простое управление и эксплуатация с помощью стандартного программного обеспечения.

1.11. Коммутационные процессоры

Для того чтобы облегчить создание связи между человеком и машиной или машиной и машиной, в ПК S5 -115U предлагается использовать ряд специальных коммутационных процессоров (CP). Они делятся на две основные группы:

- CP для сетевых систем;
- CP для передачи сообщений и протоколирования.

1.12. Возможности расширения

Если мощность центрального устройства (ZG) недостаточна для станка или установки, то можно воспользоваться устройствами расширения (EG). Модули связи соединяют центральное устройство и устройство расширения. В зависимости от желаемой конфигурации устройств можно выбрать соответствующий модуль связи.

1.13. Централизованная конфигурация

При централизованной конфигурации требуется устройство расширения, кабель связи и напряжение питания. При централизованной конфигурации в EG не требуется собственного источника питания. При этом варианте можно подключать к центральному устройству до трех устройств расширения.

Длина кабелей между отдельными устройствами в общем не должна превышать 2,5 м.

1.14. Децентрализованная конфигурация

При децентрализованной конфигурации можно располагать устройство расширения в непосредственной близости от датчиков и исполнительных устройств. При этом стоимость кабельной обвязки датчиков и исполнительных устройств значительно уменьшается.

S5 -115U:

- удобен в обращении благодаря простому монтажу и простой технике подключения (винтовые клеммы, защелкивающиеся контакты или пружины).
- Модули выполнены в виде вставных блоков, поэтому их легко заменить.
- Прочная конструкция, защищенная от электромагнитных помех.
- Можно работать без вентилятора при решении любых стандартных задач.
- Разгрузка центрального модуля процессора с помощью модулей предварительной обработки IP (кроме S5 - 115F).
- Центральный блок расположен рядом с устройством при центральной схеме управления, а при децентрализованной системе управления устройство расширения может находиться на расстоянии 23 км от центрального блока, что позволяет экономить кабель.

Если управление связано с обработкой большого числа сигналов "на месте", то лучше использовать децентрализованные устройства. Их лучше размещать на расстоянии до 3 км от основного устройства.

Существуют следующие методы передачи:

- параллельное сопряжение при высокоскоростной передаче;
- сопряжение по волоконно-оптическому кабелю для обеспечения особо высокой помехозащищенности;
- последовательное сопряжение при обычных требованиях.

При ET200 - децентрализованной периферийной системе - возможно сокращение магистралей сбора и регистрации сигналов. При этом возможно сокращение - подключение до 125 устройств. Благодаря незначительным затратам на прокладку кабельных соединений можно быстро выполнить коммуникацию и простой монтаж.

Существуют следующие варианты периферийных устройств:

1. ET200 – состав:

- универсальное устройство - для пользователей, предъявляющих высокие требования. Оно имеет класс защиты IP20;
- комплектуется обширным спектром модулей, предназначенных для расширения бесчисленного множества индивидуальных задач автоматизации.

Модуль подключения IM318B (профиль протокола SIMEC L2 /DP) - Электрическое питание 24 В. Крепится на стандартной профильной шине.

- вывод шины - SIMEC L2 /DP - двухпроводный кабель: разъем для подключения шины IP20 с дополнительным выводом для программатора.

2. ET 200B - состав:

- компактная конструкция: блок терминала и модуль электроники.
- питание 24 В, вывод шины L2/DP.

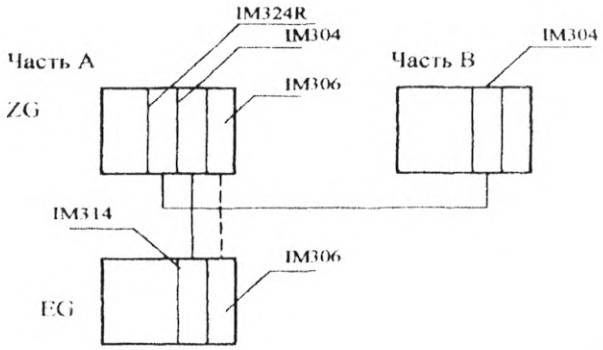
3. ET 200C - состав:

- компактная конструкция;
- периферийные устройства и соединительный тройник;
- питание 24 В, вывод шины L2/DP. Устройство обеспечивает работу в тяжелых производственных условиях.

4. SIMATIC S5 - 95U/DP - для интеллектуального решения децентрализованных задач.

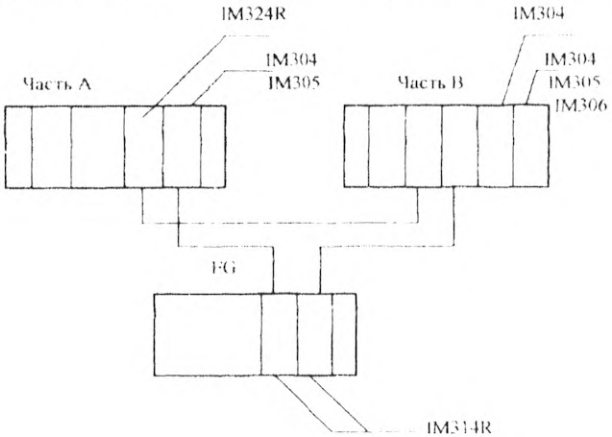
1.15. Конфигурации периферийных модулей

Односторонняя конфигурация периферийных модулей



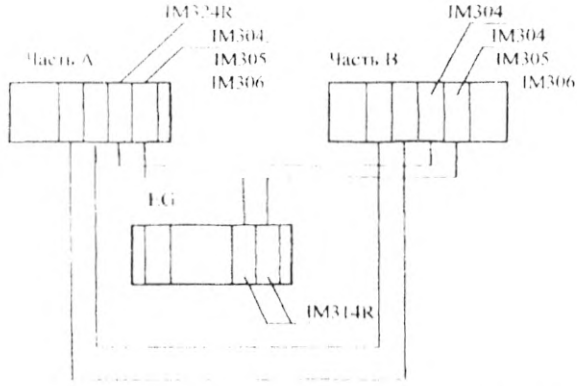
При выходе из строя ZG (центральный блок) - центрального процессора, прекращает работу вся периферия.

Переключаемая конфигурация периферийных модулей



Это устройство может работать либо с главным центральным (ZG), либо с главным резервным блоком ZG.

Смешанная конфигурация периферийных модулей



Все степени готовности можно произвольно комбинировать. Смешанная конфигурация, то есть комбинация двухканальной, переключаемой и односторонней конфигурацией часто является самым экономичным решением.

Для работы контроллера можно выбрать один из пяти существующих центральных процессоров:

-CPU 941 - 1000 команд за 10 мс ЗУ - 18 Кбайт,
-CPU 942 - 1000 команд за 10 мс ЗУ - 42 Кбайт,
-CPU 943 - 1000 команд за 5 мс ЗУ - 48 Кбайт,
-CPU 944 - 1000 команд за 1,5 мс ЗУ - 96 Кбайт.

Второй интерфейс в CPU 943 и 944 может использоваться в зависимости от потребностей:

- для подключения программатора;
- для подключения устройства обслуживания;
- для сопряжения с шиной SINEC L1;
- для сопряжения "точка к точке".

Самый быстродействующий процессор в своем классе - CPU 945. Благодаря этому процессору, система удовлетворяет самым высоким требованиям к скорости, предъявляемым, например, в производстве упаковочных материалов.

CPU 945 в минимальной конфигурации имеет ОЗУ емкостью 256 Кбайт, а в максимальной - 384 Кбайт (памяти хватит для любых функций).

Два независимых интерфейса (один из которых выполнен в виде сменного модуля) позволяют подключить:

- программатор;
- устройство обслуживания и визуализации;
- шину SINEC L1 через первый интерфейс, а через второй интерфейс дополнительно:
- программатору;
- панель оператора OP;
- SINECL1 ;
- ASCII драйвер.

До последнего времени цена аппаратной части падала, а цена программного обеспечения росла, поскольку постоянно усложнялись автоматизируемые процессы, росли требования к безопасности, повышалась зарплата персонала и постоянно ужесточались эргономические требования. С помощью данной разработки SIEMENS изменил ситуацию. SIMATIC S5-115U обладает программным обеспечением, стоимость которого понижается из-за:

- использования удобного для пользователя языка программирования STEP-5 с его четырьмя видами представления и удобной возможности структурирования;
- богатого каталога программного обеспечения и стандартных функциональных блоков;
- простых в использовании программирующих устройств.

Контроллер S5-115U состоит из различных функциональных устройств, которые можно комбинировать в зависимости от поставленной задачи.

1.16. Блок питания

Преобразует из сетевого напряжения 115/220 В переменного тока или 24 В постоянного тока рабочее напряжение для ПК и подпитывает память RAM от батареи при отключенном напряжении питания. Кроме того, он выполняет функции контроля напряжения и выдачи сообщений.

1.17. Модуль центрального процессора

Считывает информацию о состоянии сигналов на входах, выполняет программу управления и устанавливает сигналы на выходах. Наряду с функциями обработки программ, CPU предоставляет пользователю внутренние маркеры, таймеры и счетчики и допускает выполнение предварительной установки способа запуска и диагностики ошибок с помощью светодиодов, начиная с CPU 943. Кроме того, имеется возможность, используя переключатели на передней панели, производить сброс памяти PAM (полное стирание).

Программу управления можно вводить в CPU через программатор или через модуль памяти.

1.18. Коммуникационные процессоры

Для связи «человек-машина» и «машина-человек» в ПК S5-115U можно устанавливать коммуникационные процессоры. Они служат для:

- диагностики, обслуживания и мониторинга функций станка или хода протекания технологического процесса;
- выдачи сообщений и протоколирования состояния станков и технологических установок. К этим процессорам можно подключать различные периферийные устройства (например: печатающее устройство, клавиатуру, устройство обработки данных, терминалы и мониторы, а также другие устройства ЭВМ).

1.19. Технологические модули

Имеются также модули предварительной обработки сигналов для решения особых задач:

- счет импульсов большой частоты следования;
- получение и обработка инкрементов пути;
- измерение скорости и времени;
- регулирование температуры и приводов и т. д.

1.20. Модули входов и выходов, модули связи, носители модулей

Цифровые модули входов подключают цифровые сигналы (например, контактные или бесконтактные датчики BERO) и согласуют их с внутренним уровнем сигналов S5-115U. Цифровые

модули выходов преобразуют внутренний уровень сигналов процесса, (например, для реле или магнитных пускателей). Аналоговый модуль входов преобразует аналоговый сигнал процесса (например, от датчика), который затем обрабатывается контроллером S5-115U в цифровом виде. Аналоговые модули выходов формируют из внутренних цифровых значений аналоговые сигналы процессов (например, для регулирования числа оборотов двигателя).

ПК S5-115U смонтирован на носителе модулей с определенным количеством мест подключения. Носитель с блоком питания, центральным процессором и модулями периферии называется центральным устройством. Если мест подключения на носителе модулей центрального устройства недостаточно, то можно подключить другой носитель модулей, называемый устройством расширения (система без центрального процессора). Модуль связи соединяет устройства расширения с центральным устройством.

Они состоят из алюминиевого несущего профиля с механическим креплением всех модулей и имеют одну или две проводящих платы для электрической связи устанавливаемых модулей между собой.

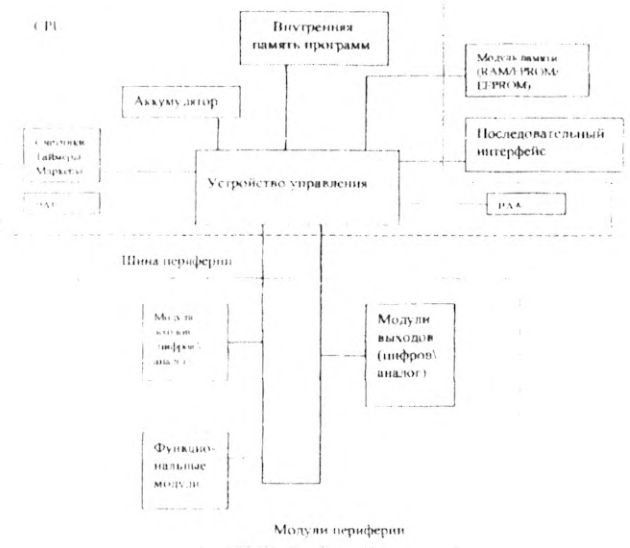
1.21. Память программ

Программа управления хранится в модуле памяти или во внутренней памяти RAM. В CPU 944 вся программа может находиться во внутренней памяти RAM (96Кбайт). Для того чтобы программа сохранялась при отсоединении модуля памяти от ПК необходимо использовать модуль памяти EPROM или EEPROM.

В отличие от них память RAM и модуль памяти RAM имеют следующие отличия:

- можно оперативно менять содержимое памяти;
- можно сохранять и изменять данные пользователя;
- при отключении сетевого напряжения и выходе из строя буферной батареи происходит пропадание данных в памяти.

1.22. Работа контроллера
Схематическое представление ПК S5-115U



1.23. Область отображения процесса (PAE, PAA)

Состояние сигналов модулей входов и выходов заносится центральным процессором в область отображения процесса. Область отображения процесса является зарезервированной областью в памяти RAM центрального процессора. Для модулей входов и выходов имеются различные области:

- область отображения процесса входов (PAE);
- область отображения процесса выходов (PAA).

Последовательный интерфейс используется для подключения программатора, устройств диагностики и мониторинга. Кроме того, через этот интерфейс контроллеры можно подключать к локальной сети SINEC L1 (в качестве SLAVE). ЦПУ 943 и 944 имеют вариант с двумя последовательными интерфейсами. Это позволяет выполнять следующие функции:

- связь "от точки к точке" с другим программируемым

контроллером;

• драйвер ASCII (только при CPU 944) для подключения клавиатуры, принтера и т.д.

1.24. Таймеры, счетчики, маркеры

Центральный процессор предоставляет в распоряжение пользователя внутренние таймеры, счетчики, маркеры, которые могут опрашиваться управляющей программой.

1.25. Аккумулятор

Аккумулятор - это регистр вычислений, с помощью которого происходит загрузка значений во внутренние счетчики и таймеры. Кроме того, в аккумуляторе производятся операции сравнения, вычисления, преобразования.

1.26.Устройство управления

Устройство управления в соответствии с программой управления вызывает последовательно одну за другой команды из памяти программ и выполняет их. При этом обрабатывается информация из области отображения входов, значения внутренних таймеров и счетчиков, а также состояние сигналов внутренних маркеров.

Шина периферии является электрическим соединителем для всех сигналов, которыми обмениваются между собой ЦПУ и остальные модули в центральном устройстве или устройстве расширения.

1.27. Описание центральных процессоров

Программируемый контроллер S5-115U/H/F выполнен в блочном исполнении.

115H - непрерывный режим - (повсюду, где требуется высокий коэффициент готовности и надежности)

- работает по принципу Master /Reserve:
- главный контроллер управляет процессом, а резервный - в горячем резерве.

	CPU 941	CPU 942	CPU 943	CPU 944	CPU 945
Язык программирования	STEP5	STEP5	STEP5	STEP5	STEP5, SCL
Общий объем памяти для программы пользователя, Кбайт	18	42	48	96	256\384
Внутренняя память RAM, Кбайт	2	10	48	96	256\384
Внешняя память, CP516/CP581,Мбайт	8/120	8/120	8/120	8/120	8/120
Время на одну команду в мкс примеры Операция с битами	1.6	1.6	0,8	0,8	0,1
Важные дополнительные функции	Встроенный алгоритм ПИД регулирования Защита программ, измерение времени цикла				
	-	-	Часы реального времени	Часы реального времени	
	факультативно				
Таймеры/счетчики	128/128	128/128	128/128	128/128	256/256
Цифровые входы/выходы	4096/4096	4096/4096	4096/4096	4096/4096	4096/4096
Аналоговые входы/выходы	256/256	256/256	256/256	256/256	256/256

При отказе главного контроллера резервный берет управление на себя; неисправность устраняется без прерывания процесса.

S115H		S115F	
1	2	3	4
1й центральный процессор CPU942H	2й центральный процессор CPU942H	1й центральный процессор CPU942F	2й центральный процессор CPU942F
Блок питания		Блок питания	
PS951	PS951	PS951	PS951
Модуль подключения		Модуль подключения	
IM304	IM324	IM304	IM324
Коммуникационный процессор		Коммуникационный процессор	

Все		Только CP523	
1	2	3	4
Модули предварительной обработки		Модули предварительной обработки	
Все кроме IP241		-	
Стандартные специальные блоки	+	Только стандартные блоки испытанные на опытном образце	

F - режим повышенной защиты от ошибок.

Область применения:

- в теплоустановках;
- на пассажирском транспорте и т.д.

Производится циклическое сравнение отображений управляемого процесса в обоих контроллерах и высококачественное дополнительное самотестирование обоих устройств. При обнаружении ошибки вся система переходит в состояние «СТОП».

S5-115U поставляется также как система повышенной надежности. Это - системы с повышенным коэффициентом готовности (H) или система (F) с повышенной защитой от ошибок. Системы H/F с двумя центральными блоками S5-115U со специальными CPU, связанными между собой. Пользователь составляет свою программу только с учетом требований,

предъявляемых к процессу и без учета повышенной надежности и контрольных сигналов.

1.28. Визуализация процесса

Производственные процессы требуют наглядности представления информации, поступающей как от станка, так от технологического процесса производства.

Все эти вопросы можно решать с помощью системы обслуживания и визуализации - COROS.

Информация о состоянии станка или технологического процесса выдается на текстовые дисплеи открытым текстом и в виде индикации измеренных значений (например: температура, число деталей и т.д.).

Эти дисплеи можно размещать в шкафах или встраивать в пульты управления.

Панели оператора отличаются удобством для пользователя. Они работают в различных форматах индикации и позволяют выбирать размер шрифта.

Программные и функциональные клавиши обеспечивают надежное и эффективное обслуживание процесса.

COROS LS-B - это высокопроизводительная система визуализации процессов, используемая как для контроля состояния станков, так и для обслуживания и контроля технологических установок.

Кроме того, устройства должны обмениваться информацией между собой и управляющим компьютером. Поэтому возникает проблема коммуникации. В системах SIMATIC S5 есть три решения:

- сопряжение «точка к точке» через второй интерфейс центрального процессора;
- сопряжение «точка к точке» с помощью коммуникационных процессоров;
- связь по шине через одну из локальных сетей SINEC HI, SINEC L2, SINECL1.

Если возникает большое количество устройств, которые необходимо связать друг с другом, то связь «точка к точке» становится нерентабельной. Шинные системы требуют малых затрат

на прокладку кабелей, легко расширяются и обеспечивают прямую связь между пользователями.

С помощью шинной системы SINEC HI можно создавать большие и сложные системы связи для всех уровней автоматизации. Максимальное количество пользователей 1024.

Шинная система SINEC L2 по стандарту PROFIBUS фирмы SIEMENS является недостающим звеном между SINEC L1 и SINEC H1. L2 - применяют во всех областях техники на уровне периферийных устройств и модулей производственных систем.

SINEC L1 – самая дешевая шинная система. Она особенно выгодна для некритичных по времени случаев применения. Данная система позволяет связать друг с другом до 31 пользователя SIMATIC S5, которые могут находиться друг от друга на расстоянии до 50 км.

1.29. Программирование

На языке STEP5 в форме представления:

- AWL - командный план;
- FUP - функциональный план;
- KOP - контактный план.

Имеются следующие варианты для языка STEP5:

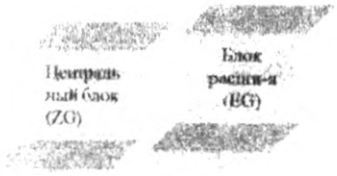
- S5-DOS / ST (на базе MS-DOS);
- S5-DOS / MT (на базе FlexOS).

Это означает, что под управлением S5-DOS можно составить программы на языке STEP5 для всех "обычных" случаев применения SIMATIC S5, а под управлением S5-DOS / MT работать даже в многозадачном режиме.

1.30. Контроллеры 85-1351Д S5-155U/H

S5-135U/S5-155U/H - многопроцессорная система в компактном исполнении с блоком питания и вентилятором. Простая техника подключения (винтовые и защелкивающиеся контакты).

Как и все системы семейства Simatic S5, многопроцессорная система S5-135U/S5-155U/H обладает возможностью адаптации - благодаря совместимости с различными стандартами входных и выходных напряжений, а также простому модульному наращиванию входов/выходов и обмена памяти.



Гибкость этой системы позволяет увеличивать возможности ее простым модульным наращиванием входов, выходов и памяти. В одну систему S5-135U/1155U одновременно вмещаются до 4 центральных процессоров (CPU). Одновременно используя до четырех процессоров, можно разделить критичные и некритичные по времени задачи по времени задания и составить структурированные программы.

Каждый CPU имеет накопитель программ: данных и оперативной памяти, а также большое количество специальных функций, которые облегчают программирование, например:

а) готовые регистры сдвига;

б) PID - регулирование.

Преимущества мультипрограммирования:

- повышение скорости обработки при помощи параллельной обработки;
- несложное структурирование автоматических задач.

Разгрузка центрального процессора производится с помощью модулей предварительной обработки сигналов (IP).

- Регулирование - IP244, IP253, IP260.

Позиционирование → управление IP240, IP241, IP247

Позиционирование → регулирование IP2461/A, IP288.

- Счет/дозирование IP240, IP242A, IP242B и т. д.
- Обработка сигналов IP243, WF705.
- Кулачковое управление IP288, WF707.

Простая коммуникация с другими контроллерами и вычислительными устройствами возможна через собственные коммуникационные процессоры (CP523, CP524, CP5-2 -при соединении "от точки к точке") и шинные системы (CP530 SINEC).

Через шинные системы SINEC LI, SINEC L2, SINEC HI также могут объединяться в сеть.

Расширение системы возможно центральное (EG расположен рядом или над ZG) или децентральное (135U/1155U: EG удален до 23 км от ZG).

Данные контроллеры S5-135U/1155U могут обеспечить работу системы в многопроцессорном режиме (кроме S5-155H).

Контроллер S5-155U имеет наибольший набор команд и допускает самое большое расширение памяти.

Для примера можно привести три центральных процессора, используемые в S5-135U контроллере CPU928, CPU922, CPU920, которые, работая в одиночном режиме, достаточно мощны, а чтобы стать еще сильнее, они могут работать сообща.

CPU928, CPU922 - любой из них гарантирует опрос 8 регулирующих контуров за 20 мс.

CPU920 – мультипроцессор, проводит полную интеграцию программируемого языка в систему STEP5. Он понимает Ассемблер, Бейсик и "С".

Программирование производят с помощью программаторов PG750, PG720, PG740, PG750D, PG720P.

Для программирования применяют **STEP5**.

Для программирования обработки операций в графической форме применяют GRAPH.

Контроллеры **S5-135U** или **S5-155U** состоят из одного центрального блока и по необходимости, из блоков расширения. Центральный блок всегда оснащен встраиваемым блоком питания с вентилятором и одним или несколькими центральными процессорами (**CPU**). Выпускаются различные центральные процессоры.

Блоки расширения поставляются с блоками питания или без них (с вентиляторами или без них). В зависимости от задачи автоматизации в котроллер вставляются различные периферийные модули:

- модули цифрового ввода - вывода
- модули аналогового ввода - вывода;
- коммуникационные процессоры;
- модули предварительной обработки сигналов.

Все модули устанавливаются в контроллер.

1.31. Системы повышенной надежности S5 -155H

Контроллер **S5 —155H** поставляется также как система с повышенной надежностью и высокой готовностью. Система состоит из двух центральных блоков **S5 -155U** со специальным **CPU**, сопряженным друг с другом. Однако программа пользователя составляется без учета повышенной надежности системы и контрольных сигналов.

Возможные конфигурации S5 -155H:

Односторонняя структура периферийных модулей.

Переключаемая конфигурация периферийных модулей

Двухканальная конфигурация периферийных модулей.

Трехканальная конфигурация модулей цифрового и аналогового ввода.

Смешанная конфигурация.

ГЛАВА 2
МАТЕМАТИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ БАЗОВЫХ АЛГОРИТМОВ УПРАВЛЕНИЯ

2.1. Основные термины и обозначения

Схема управления приводом – это аппаратно реализованная электрическая схема включения/выключения/блокировки привода. Как правило, кроме автоматического режима управления существует также и режим местного управления приводом, который выполняется с помощью кнопок со шкафа/поста, в котором и реализована схема управления приводом.

Команда управления приводом – это промежуточная информация, инициирующая процесс логического управления приводом (например, включения/отключения).

Сигнал управления приводом – это выходной сигнал контроллера, выдаваемый в схему управления приводом.

Технологический процесс – это последовательность технологических операций, необходимых для выполнения определенного вида работ. В данном случае – это последовательность состояний оборудования по истечении определенного времени и/или для достижения определенного значения технологического параметра.

Автомат – это алгоритм, созданный на принципах SWITCH-технологии.

Технологический автомат – это автомат, реализующий логику технологического процесса. Основная функция – формирование команд управления приводами.

Алгоритм управления приводом – это автомат, реализующий логику непосредственного управления приводом по командам управления и осуществляющий контроль состояния привода. Основная функция – формирование сигналов в схему управления приводом и формирование информации о ненормальной его работе.

2.2. Функции локальной системы управления

После **изучения технического задания** и особенностей технологического процесса и работы оборудования можно приступить к определению функций системы.

Функции любой системы управления можно сформулировать как "принял – обработал – выдал". Локальная система управления с использованием контроллера SIMATIC S7-200, как правило, призвана реализовать следующие функции:

- Первичная обработка аналоговых сигналов (включая проверку на достоверность) и масштабирование значений параметров, представленных в виде аналоговых и сигналов.
- Контроль значений параметров, представленных аналоговыми и дискретными сигналами.
- Технологические алгоритмы.
- Алгоритмы управления приводами.
- Поддержка буквенно-цифрового индикатора TD200.
- Поддержка обмена по сети (как правило, сеть ProfiBus).

Резидентно контроллер (процессорный модуль и интеллектуальные модули) обеспечивает ввод/вывод сигналов и собственно обмен по сети.

2.3. Разработка информационного обеспечения

Одновременно с определением функций системы следует создать основу информационного обеспечения – *таблицу входных/выходных сигналов контроллера* (сюда не входят данные, получаемые/передаваемые по сети). Пример приведен в табл. 2.1.

Таблица входных/выходных сигналов контроллера является первой из таблиц символического представления элементов памяти контроллера (битов, байтов, слов, двойных слов). Эту и последующие таблицы рекомендуется создавать в редакторе Excel, что позволяет легко переносить записи в таблицу **Symbol Table** среды разработки **STEP7-MicroWin32**.

В таблице входных/выходных сигналов контроллера есть необязательный дополнительный столбец "Адреса при тестировании", предназначенный для замены реальных адресов входов контроллера при применении программных имитаторов работы приводов.

Таблица входных/выходных сигналов контроллера

Таблица 2.1.

Symbol (Наименование)	**Address** (Адрес)	**Comment** (Комментарий)	Адреса при тестировании
Входные/выходные сигналы контроллера			
		Входные аналоговые сигналы	
AI_LCBC	AIW0	16-разрядное значение кода аналогового сигнала "Ток нагрузки"	
AI_OPPW	AIW2	16-разрядное значение кода аналогового сигнала "Выходное давление"	
		Входные дискретные сигналы	
DI_380V	I2.0	Питание 380 В	
DI_OnAS220V	I2.1	Включен автоматический выключатель питания цепей управления ~220 В	
DI_OnKIP220V	I2.2	Включен автоматический выключатель питания КИП ~220 В	
DI_LC	I3.0	Избиратель режима управления в положении "Местный"	
DI_AC	I3.1	Избиратель режима управления в положении "Автоматический"	

Продолжение табл. 2.1.

		Входные дискретные сигналы шкафа насосного агрегата (ШНА)	
DI_CACWP	I4.3	Насосный агрегат выбран для управления	
DI_CConWP	I4.4	Команда от кнопки шкафа "Включить насосный агрегат"	
DI_CCoffWP	I4.5	Команда от кнопки шкафа "Отключить насосный агрегат"	
DI_CCEstopWP	I4.6	Команда от кнопки шкафа "Аварийное отключение насосного агрегата"	
		Входные дискретные сигналы из схемы привода напорной задвижки	
DDI_X0S_FW	I0.0	Включено питание приво-да напорной задвижки	
DDI_X1S_FW	I0.1	Напорная задвижка открывается	V3000.1
DDI_X2S_FW	I0.2	Напорная задвижка закрывается	V3000.2
DDI_X3S_FW	I0.3	Напорная задвижка открыта	V3000.3
DDI_X4S_FW	I0.4	Напорная задвижка закрыта	V3000.4

Окончание табл. 2.1.

DDI_X5S_FW	I0.5	Срабатывание муфты напорной задвижки	
DDI_HTD_FW	I0.6	Превышение температуры двигателя напорной задвижки	
		Выходные дискретные сигналы в схему привода напорной задвижки	
DDO_Z1S_FW	Q0.1	Открыть напорную задвижку	
DDO_Z2S_FW	Q0.2	Закрыть напорную задвижку	
		Входные дискретные сигналы из схемы привода промывного насоса	
DDI_ReadyWP	I4.0	Насосный агрегат к пуску готов	
DDI_X1P_WP	I4.1	Насосный агрегат включен	V3004.1
DDI_EstopWP	I4.2	Насосный агрегат аварийно отключен	
		Выходные дискретные сигналы в схему привода промывного насоса	
DDO_Z1WP	Q0.0	Включить насосный агрегат	
DDO_predpuskWP	Q0.4	Предупреждение пуска	

ГЛАВА 3
ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ "STEP-7"

3.1. Обзор STEP-7

STEP-7 – это пакет стандартного программного обеспечения, используемый для конфигурирования и программирования программируемых логических контроллеров SIMATIC. Он является частью промышленного программного обеспечения SIMATIC. Имеются следующие версии стандартного пакета STEP-7:

-STEP-7 Micro/DOS и STEP-7 Micro/Win для относительно простых автономных приложений на SIMATIC S7-200.

-STEP-7 Mini для относительно простых автономных приложений на SIMATIC S7-300 и SIMATIC C7-620.

-STEP-7 для приложений на SIMATIC S7-300/S7-400, SIMATIC M7-300/M7-400 и SIMATIC C7.

3.2. SIMATIC Manager

"SIMATIC Manager" – это центральное окно, которое становится активным при запуске STEP-7. "SIMATIC Manager" - это основное приложение для проектирования и программирования. В нем вы можете реализовать следующие функции:

- создавать проекты;
- конфигурировать и назначать параметры аппаратным средствам;
- конфигурировать аппаратные сети;
- программировать блоки;
- отлаживать и принимать в эксплуатацию свои программы.

По умолчанию запускается мастер STEP-7 (STEP-7 Wizard), который оказывает вам помощь при создании проекта STEP-7. Структура проекта используется для надлежащего хранения и размещения всех данных и программ. Внутри проекта (Project) данные хранятся в виде объектов в иерархической структуре

- Станция SIMATIC (SIMATIC - Station) и CPU содержат данные о конфигурации и параметрах аппаратного обеспечения
- Программа S7 (S7-Program) включает в себя все блоки (Blocks) с программами, необходимыми для управления процессом

Доступ к различным функциям спроектирован объектно-ориентированно, интуитивно понятным и легким для изучения.

Вы можете работать с SIMATIC Manager одним из двух способов:

- OFF LINE, без подключения программируемого контроллера;
- ONLINE, с подключенным программируемым контроллером.

Обратите внимание на соответствующие указания по безопасности в каждом случае.

3.3. Создание и редактирование проекта

Структура проекта. Проекты используются для хранения данных и программ, которые создаются для решения задачи автоматизации. Данные, собранные в проекте, включают в себя:

-конфигурационные данные о структуре аппаратного обеспечения и параметрах модулей;

-конфигурационные данные о коммуникациях в сетях;

-программы для программируемых модулей.

Главной задачей при создании проекта является подготовка этих данных для программирования. Данные хранятся в проекте в виде объектов. Объекты в проекте упорядочены в виде древовидной структуры. Отображение этой иерархии в окне проекта похоже на отображение Проводника в Windows 95. Другой внешний вид имеют только пиктограммы объектов.

Верхняя часть иерархии проекта структурирована следующим образом:

1-й уровень: проект

2-й уровень: подсети, станции или программы S7/M7

3-й уровень: зависит от объекта на уровне 2

Приведем пример создания проекта при помощи STEP-7 Wizard.

Дважды щелкните на пиктограмме **SIMATIC Manager**. Активизируется мастер STEP-7 (STEP-7 Wizard). В **предварительном обзоре (preview)** вы можете включать и выключать отображение структуры создаваемого проекта.

Чтобы перейти к следующему диалоговому окну, щелкните на кнопке **Next**.

Для примеров проектов в STEP-7 выберите CPU 314. Пример создан таким образом, что вы фактически можете выбрать

CPU, который вам может быть поставлен в любое время. Установка по умолчанию для адреса MPI равна 2.

Щелкните на **Next**, чтобы подтвердить настройки и перейти к следующему диалоговому окну. Выберите организационный блок **OB1** (он еще не выбран). Выберите один из языков программирования: контактный план **(LAD)**, список операторов **(STL)** или функциональный план **(FBD)**. Подтвердите настройки кнопкой **Next**.

Каждый CPU обладает определенными свойствами; например, относительно конфигурации его памяти или адресных областей. Вот почему вы должны выбрать CPU, прежде чем начать программирование.

Адрес MPI (интерфейс) нужен, чтобы ваш CPU мог обмениваться информацией с вашим устройством программирования или PC. OB1 представляет самый высокий уровень программирования и организует другие блоки в программе S7. Позднее вы сможете снова изменить язык программирования.

Дважды щелкните в поле "Project name (Имя проекта)", чтобы выбрать предлагаемое имя и перепишите его.

Щелкните на кнопке **Make**, чтобы сгенерировать свой новый проект в соответствии с предварительным обзором. Когда вы щелкнете на кнопке **Make**, SIMATIC Manager откроет окно для проекта, который вы создали.

Мастер STEP-7 активизируется каждый раз, когда запускается эта программа. Вы можете деактивировать эту установку по умолчанию в первом диалоговом окне для мастера (Wizard). Однако, если вы создаете проекты без мастера STEP-7, то вы должны создавать каждый каталог внутри проекта сами.

Дополнительную информацию вы можете найти, используя команду меню **Help > Contents** в разделе "Setting Up and Editing the Project".

Как только мастер STEP-7 закрывается, появляется SIMATIC Manager с открытым окном проекта. Отсюда вы можете запускать все функции и окна STEP-7.

3.4. Основы проектирования структуры программы

Программы в CPU. В CPU всегда исполняются две программы:

- операционная система;
- программа пользователя.

Операционная система. Каждый CPU содержит операционную систему, которая организует все функции и последовательности в CPU, не связанные с конкретной задачей управления. Задачи операционной системы состоят в следующем:

-обработка "холодного" и "теплого" перезапуска;

-обновление таблицы образа процесса для входов и вывод таблицы образа процесса для выходов;

-вызов программы пользователя;

-обнаружение прерываний и вызов OB прерываний;

-обнаружение и обработка ошибок;

-управление областями памяти;

-обмен информацией с устройствами программирования и другими коммуникационными партнерами.

Если изменить параметры операционной системы (операционной системы по умолчанию), можно повлиять на работу CPU в определенных областях.

Иерархия вызовов в программе пользователя. Для функционирования программы пользователя, составляющие ее блоки должны вызываться. Это делается с помощью специальных команд STEP-7, вызовов блоков, которые могут быть запрограммированы и запущены только в логических блоках.

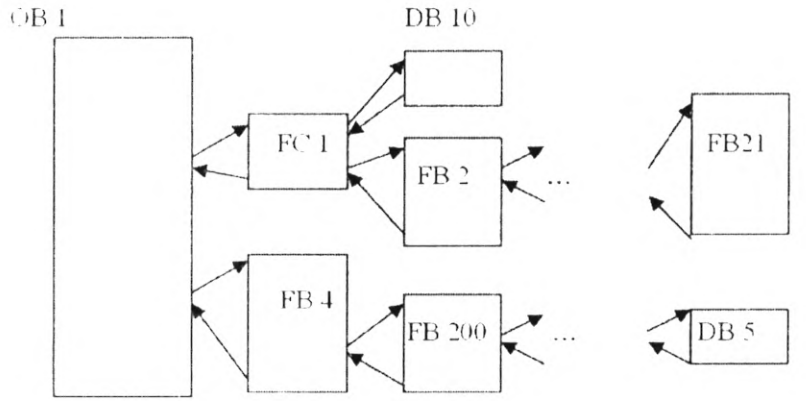
Порядок и глубина вложения. Порядок и вложение вызовов блоков называется иерархией вызовов. Количество блоков, которые могут быть вложены друг в друга (глубина вложения), зависит от конкретного CPU.

Существует установленный порядок создания блоков:

- Блоки создаются сверху вниз, то есть вы начинаете с верхнего ряда блоков.

- Каждый вызываемый блок уже должен существовать, т.е. внутри ряда блока они должны создаваться справа налево.

- Последним создается блок OB.



Программа пользователя. Программа пользователя содержит все функции, необходимые для обработки конкретной задачи автоматизации. Задачи программы пользователя состоят в следующем:

- определение условий для "холодного" и "теплого" перезапуска в CPU (инициализация сигналов с определенным значением);
- обработка данных процесса (логическая комбинация двоичных сигналов, считывание и анализ аналоговых сигналов, задание двоичных сигналов для вывода, вывод аналоговых значений);
- определение реакции на прерывания;
- обработка нарушений в нормальном исполнении программы.

3.5. Линейное и структурное программирование

Вы можете записать всю свою программу в OB1 (линейное программирование). Это целесообразно только в случае простых программ, написанных для CPU S7-300 и требующих мало памяти. Сложными задачами автоматизации проще управлять, если они разделены на более мелкие задачи, которые отражают технологические функции процесса и могут быть использованы неоднократно. Эти задачи представляются соответствующими

программными секциями, известными как блоки (структурное программирование).

3.6. Блоки в программе пользователя.

Программное обеспечение STEP-7 дает возможность структурировать пользовательскую программу, иными словами, разбивать программу на отдельные автономные программные секции. Это обеспечивает следующие преимущества:

- Такие программы проще для понимания.
- Отдельные программные секции могут быть стандартизированы.
- Упрощается организация программы.
- Легче производить модификацию программы.
- Отладка упрощается, так как можно тестировать отдельные секции.
- Значительно упрощается прием системы в эксплуатацию.

3.7. Типы блоков

Имеется несколько различных типов блоков, которые можно использовать внутри пользовательской программы S7:

Блок	Краткое описание функции
Организационные блоки (OB)	OB определяют структуру программы пользователя.
Системные функциональные блоки (SFB) и системные функции (SFC)	SFB и SFC встроены в CPU S7 и обеспечивают доступ ко всем важным системным функциям.
Функциональные блоки (FB)	FB – это блоки с "памятью", которые вы можете программировать сами.
Функции (FC)	FC содержат программы для часто встречающихся функций.
Экземплярные блоки данных (DB)	Экземплярные DB связываются с блоком, когда вызывается FB/SFB. Они создаются автоматически при компиляции.
Блоки данных (DB)	DB – это области данных для хранения данных пользователя. Кроме данных, соответствующих функциональному блоку, могут быть определены также данные, совместно используемые любыми блоками.

OB, FB, SFB, FC и SFC содержат секции программы и поэтому известны также как логические блоки. Допустимое количество блоков каждого типа и допустимая длина блоков зависят от CPU.

3.8. Организационные блоки и структура программы

Организационные блоки (OB) образуют интерфейс между операционной системой и программой пользователя. Они вызываются операционной системой и управляют циклическим и по прерываниям исполнением программы, а также запуском программируемого логического контроллера. Они также обрабатывают реакцию на ошибки. Программируя организационные блоки, вы определяете реакцию CPU.

Приоритет организационного блока. Организационные блоки определяют порядок, в котором исполняются отдельные программные секции. Исполнение OB может быть прервано вызовом другого OB. Какому OB разрешается прервать другой OB в зависимости от его приоритета? OB с более низким приоритетом могут прерываться OB с более высоким приоритетом. Фоновый OB имеет самый низкий приоритет.

Типы прерываний и классы приоритета. События, которые приводят к вызову OB, известны как прерывания. Не все перечисленные организационные блоки и их классы приоритета доступны во всех CPU S7.

3.9. Функции (FC)

Функции (FC) относятся к блокам, которые вы программируете сами. Функция – это логический блок. Временные переменные, принадлежащие FC, хранятся в стеках локальных данных. После того как FC выполнена, эти данные теряются. Чтобы хранить эти данные постоянно, функции могут также использовать разделяемые блоки данных. Так как FC не имеет собственной памяти, то всегда необходимо определять для нее фактические параметры.

Применение. FC содержит программную секцию, которая выполняется всегда, когда FC вызывается другим логическим блоком. Функции можно использовать для следующих целей:

-для возврата значения функции в вызывающий блок (математические функции);

-для выполнения технологической функции (отдельная функция управления с битовой логической операцией).

3.10.Функциональные блоки (FB)

Функциональные блоки (FB) относятся к блокам, которые вы программируете сами. Функциональный блок -- это логический блок. В качестве памяти ему назначается блок данных. В экземплярном DB сохраняются параметры, передаваемые FB, и статические переменные. Временные переменные хранятся в стеке локальных данных. Данные, сохраняемые в экземплярном DB, не теряются, когда исполнение FB завершено. Однако, данные, сохраняемые в стеке локальных данных, теряются, когда исполнение FB завершено.

Применение. FB содержит программу, которая исполняется всегда, когда FB вызывается другим логическим блоком. Функциональные блоки значительно облегчают программирование часто встречающихся сложных функций.

Функциональные блоки и экземплярные блоки данных. Экземплярный блок данных назначается каждому вызову функционального блока, который передает параметры. Вызывая более одного экземпляра FB, можно с помощью одного FB управлять более чем одним устройством. Например, FB для некоторого класса двигателей может управлять различными двигателями, используя различные наборы экземпляров данных для разных двигателей. Данные для каждого двигателя (скорость, накопленное время работы и т.д.) могут быть сохранены в одном или нескольких экземплярных DB.

3.11. Экземплярные блоки данных

Экземплярный блок данных назначается каждому вызову функционального блока, который передает параметры. Фактические параметры и статические данные FB хранятся в экземплярном DB. Переменные, описанные в FB, определяют структуру экземплярного блока данных. Экземпляр означает вызов функционального блока. Если, например, функциональный блок вызывается в программе пользователя S7 пять раз, то имеется пять экземпляров этого блока.

Создание экземплярного DB. Перед созданием экземплярного блока данных соответствующий FB уже должен существовать. При создании экземплярного блока данных указывается номер FB.

Один экземплярный DB для каждого отдельного экземпляра. При назначении функциональному блоку (FB), который управляет двигателем, нескольких экземплярных блоков данных этот FB можно использовать для управления разными двигателями. Данные для каждого конкретного двигателя (скорость, время пуска, общее время работы) сохраняются в разных экземплярных блоках данных. DB, связываемый с FB при вызове последнего, определяет, какой из двигателей управляется. При использовании этого метода для нескольких двигателей нужен только один функциональный блок.

3.12. Совместно используемые блоки данных (DB)

В отличие от логических блоков, блоки данных не содержат команд STEP-7. Они используются для хранения данных пользователя, иными словами, блоки данных содержат переменные данные, с которыми работает программа пользователя. Совместно используемые блоки данных применяются для хранения пользовательских данных, к которым могут обратиться все остальные блоки. Размер DB может изменяться. Вы можете структурировать совместно используемые блоки данных любым способом для удовлетворения своих конкретных потребностей.

Совместно используемые блоки данных в программе пользователя. Если логический блок (FC, FB или OB) вызывается, то он временно занимает место в области локальных данных (L-стек). В дополнение к этой области локальных данных логический блок может открыть область памяти в виде DB. В отличие от данных, находящихся в области локальных данных данные в DB не удаляются, когда DB закрывается, иначе говоря, после исполнения соответствующего логического блока. Каждый FB, FC или OB может читать данные из совместно используемого DB или записывать данные в этот DB. Эти данные сохраняются в DB после выхода из него.

3.13. Системные функциональные блоки (SFB) и системные функции (SFC)

Предварительно запрограммированные блоки. У вас нет необходимости программировать каждую функцию самостоятельно. CPU S7 предоставляют в ваше распоряжение заранее запрограммированные блоки, которые можно вызывать в своей пользовательской программе.

Системные функциональные блоки. Системный функциональный блок (SFB) – это функциональный блок, встроенный в CPU S7. SFB являются частью операционной системы и не загружаются как часть программы пользователя. Как и FB, SFB – это блоки. Для SFB тоже нужно создавать экземплярные блоки данных и загружать их в CPU как часть программы. CPU S7 предоставляют в распоряжение следующие SFB:

- для связи через проектируемые соединения;
- для встроенных специальных функций.

Системные функции. Системная функция – это заранее запрограммированная, оттестированная функция, встроенная в CPU S7. Вы можете вызвать SFC в своей программе. SFC являются частью операционной системы и не загружаются как часть программы. Как и FC, SFC являются блоками. CPU S7 предоставляют SFC для следующих функций:

- копирование и блочные функции;
- контроль программы;
- работа с часами и счетчиками рабочего времени;
- передача наборов данных;
- передача событий из CPU всем остальным CPU в мультипроцессорном режиме;
- обработка прерываний по времени и с задержкой;
- обработка синхронных и асинхронных ошибок;
- информация о статических и динамических системных данных (например, диагностика);
- обновление образа процесса и обработка битовых массивов;
- адресация модулей;
- децентрализованная периферия;
- связь с помощью глобальных данных;
- связь через неспроектированные соединения;
- генерирование сообщений, относящихся к блокам.

3.14. Последовательность действий при конфигурировании и параметризации централизованной структуры

Предпосылка. Вы открыли или создали новый проект в SIMATIC Manager.

Принципиальная последовательность действий. Чтобы сконфигурировать и параметрировать структуру, действуйте следующим образом:

- создание станции;
- вызов приложения "Configuring Hardware (конфигурирование аппаратуры)";
- определение свойств модулей;
- размещение стоек;
- сохранение конфигурации;
- загрузка конфигурации в систему автоматизации;
- размещение входных/выходных модулей.

Краткое описание отдельных шагов.

- Установка и авторизация.

При первом использовании STEP-7, установите его и перенесите авторизацию с дискеты на жесткий диск.

- Спланируйте концепцию использования вашего контроллера.

Перед началом работы со STEP-7 спланируйте решение задачи автоматизации от деления процесса на отдельные задачи до создания диаграммы конфигурации

- Спроектируйте структуру программы.

Преобразуйте задачи, описанные в эскизном проекте вашего контроллера, в структуру программы, используя блоки, имеющиеся в STEP-7.

- Запустите STEP-7.

STEP-7 запускается из пользовательского интерфейса Windows 95/98/NT.

- Создайте структуру проекта.

Проект похож на папку, в которой все данные хранятся в виде иерархической структуры и доступны в любое время. После создания проекта все остальные задачи выполняются в этом проекте.

- Сконфигурируйте станцию.

При конфигурировании станции вы указываете, какой программируемый контроллер вы хотите использовать; например, SIMATIC 300, SIMATIC 400, SIMATIC S5.

- Сконфигурируйте аппаратуру.

При конфигурировании аппаратуры вы указываете в конфигурационной таблице, какие модули вы хотите использовать для решения своей задачи автоматизации и какие адреса должны быть использованы для доступа к модулям из программы пользователя. Модулям также могут быть назначены свойства с помощью параметров.

- Спроектируйте сети и коммуникационные связи.

Основой для коммуникаций является предварительно спроектированная сеть. Для этого вам нужно будет создать подсети, необходимые для ваших задач автоматизации, установить свойства подсетей и сетевых подключений и всех коммуникационных связей, требуемых для сетевых станций.

- Определите символы.

Вы можете определить в таблице символов локальные или совместно используемые символы, имеющие более наглядные имена, для использования вместо абсолютных адресов в своей пользовательской программе.

- Создайте программу.

Используя один из доступных языков программирования, создайте программу, связанную с модулем или независимую от модуля, и сохраните ее в виде блоков, исходных файлов или схем.

- Только для S7: сгенерируйте и проанализируйте справочные данные. Вы можете использовать эти справочные данные для облегчения отладки и модификации программы пользователя

- Спроектируйте сообщения.

Сообщения, относящиеся к блокам, создаются, например, с помощью их текстов и атрибутов. Используя передающую программу, вы переносите созданные данные о конфигурации сообщений в базу данных системы взаимодействия с оператором.

- Спроектируйте переменные для управления и наблюдения оператором.

Вы создаете переменные для управления и наблюдения оператором один раз в STEP-7 и назначаете им требуемые атрибуты.

Используя передающую программу, вы переносите созданные переменные для управления и наблюдения оператором в базу данных системы взаимодействия с оператором WinCC.

- Загрузите программы в программируемый контроллер.

Только для S7: после завершения конфигурирования, назначения параметров и программирования задач вы можете загрузить всю свою пользовательскую программу или отдельные блоки из нее в программируемый контроллер. CPU уже содержит операционную систему. Только для M7: выберите подходящую операционную систему для решения своей задачи автоматизации из ряда различных операционных систем и перенесите ее отдельно или вместе с программой пользователя на требуемый носитель данных системы программного управления M 7.

- Протестируйте программу.

Только для S7: для тестирования значения переменных из своей пользовательской программы или CPU, или присвоить значения переменным и создать таблицу для переменных, которые вы хотите отображать или изменять.

- Наблюдайте за работой, диагностируйте аппаратуру.

Причина неисправности модуля определяется отображением информации о модуле в режиме online. Причины ошибок в обработке программы пользователя определяются с помощью диагностического буфера и содержимого стеков. Вы можете также проверить, может ли программа пользователя исполняться на конкретном CPU.

3.15. Используемые стандарты

Языки программирования SIMATIC и встроенные в STEP-7 представления языков соответствуют требованиям стандарта EN 63-3 или IEC 113-3. Стандартный пакет работает в операционной системе Windows 95/98/NT и соответствует графической и объектно-ориентированной философии работы Windows.

3.16. Функции стандартного пакета

Стандартное программное обеспечение оказывает вам поддержку на всех стадиях процесса решения задачи автоматизации – таких, как:

- создание и управление проектами;

- конфигурирование и назначение параметров аппаратуре и связям;

- управление символами;

- создание программ (например, для программируемых контроллеров S7);

- загрузка программ в программируемые контроллеры;

- тестирование системы автоматизации;

- диагностика неисправностей установки.

Пользовательский интерфейс программного пакета STEP-7 спроектирован так, чтобы удовлетворить самым последним достижениям эргономики, и облегчает начало работы.

3.17. Приложения в STEP-7

Стандартный пакет STEP-7 предоставляет ряд приложений внутри программного пакета.

Редактор символов.

С помощью редактора символов вы управляете всеми совместно используемыми символами. Доступны следующие функции:

- установка символических имен и комментариев для сигналов процесса (входов/выходов), маркеров и блоков;

- функции сортировки;

- импорт в другие программы и экспорт из других программ Windows.

Таблица символов, созданная этим инструментальным средством, доступна и всем другим инструментам. Следовательно, любые изменения свойств символа автоматически распознаются всеми инструментальными средствами.

Диагностика аппаратуры.

Эти функции предоставляют вам обзор состояния программируемого контроллера. Обзор может отображать символы, чтобы показать, неисправен какой-нибудь модуль или нет. Двойной щелчок на неисправном модуле отображает подробную информацию о неисправности. Объем этой информации зависит от конкретного модуля:

- отображение общей информации о модуле (номер для заказа, версия, имя) и состояния модуля (работа, неисправен);

- отображение неисправностей модуля (неисправность канала) для центрального устройства и slave-устройств DP;

- отображение сообщений из диагностического буфера.

Для CPU отображается следующая дополнительная информация:

- причины неисправностей при обработке программы пользователя;

- отображение длительности цикла (длинного, самого короткого и последнего);

- возможности и загрузка связей через MPI;

- отображение функциональных характеристик (возможных входов/выходов, маркеров, счетчиков, таймеров и блоков).

Языки программирования.

Языки программирования - контактный план, список операторов и функциональный план для S7-300 и S7-400 - являются составной частью стандартного пакета.

- Контактный план (KOP, на англ. LAD) – это графическое представление языка программирования STEP-7. Его синтаксис для команд похож на релейные схемы: такая схема дает возможность проследить поток энергии между шинами при его прохождении через различные контакты, составные элементы и выходные катушки.

- Список команд (AWL, на англ. STL) – это текстовое представление языка программирования STEP-7, подобное машинному коду. Если программа написана в виде списка команд, то отдельные команды соответствуют шагам, с помощью которых CPU исполняет программу. Для облегчения программирования список команд расширен путем включения в него некоторых конструкций языков высокого уровня.

- Функциональный план (FUP, на англ. FBD) – это графическое представление языка программирования STEP-7, использующее для представления логики логические блоки подобно булевой алгебре. Сложные функции (математические функции) могут быть представлены непосредственно в соединении с

логическими блоками. Другие языки программирования доступны в виде дополнительных пакетов.

Конфигурирование аппаратуры.

Это инструментальное средство используется для конфигурирования и назначения параметров аппаратуре. используемой в проекте автоматизации. В распоряжении имеются следующие функции:

- Для конфигурирования программируемого контроллера вы выбираете стойки из электронного каталога и размещаете выбранные модули в необходимых слотах на этих стойках.

- Конфигурирование децентрализованной периферии идеентично конфигурированию центрального устройства. Поддерживаются также входы/выходы на уровне каналов.

- В процессе назначения параметров CPU вы можете установить такие свойства, как поведение при запуске и контроль времени цикла под управлением меню. Поддерживается многопроцессорный режим. Введенные данные хранятся в системных блоках данных.

- В процессе назначения параметров модулям все устанавливаемые параметры назначаются через диалоговые окна. Настройки, устанавливаемые с помощью двухпозиционных переключателей, отсутствуют. Присвоение параметров модулям производится автоматически при запуске CPU. Это значит например, что модуль может быть заменен без назначения новых параметров.

- Назначение параметров функциональным модулям (FM) и коммуникационным процессорам (CP) производится с помощью инструментального средства Hardware Configuration точно таким же образом, как и для других модулей. Для каждого FM и CP (в сферу действия функционального пакета FM/CP) существуют специфические для модулей диалоговые окна и правила. Система препятствует неправильным вводам, предлагая только допустимые варианты в диалоговых окнах.

Конфигурирование центральных модулей. Правила размещения модулей (SIMATIC-400).

Правила размещения модулей на стойке S7-400 зависят от вида применяемой стойки.

Центральная стойка. Вы можете:

- устанавливать блоки питания только в слоте 1 (резервируемые блоки питания);
- устанавливать не более 6 интерфейсных модулей (IM); из них не более 2 с передачей тока;
- подключить к центральной стойке через интерфейсные модули не более 2 стойки расширения;
- подключить к интерфейсу передающего IM (IM 460-1 с IM 46-1) не более 1 стойки расширения **с передачей тока**; не более 4 стоек расширения **без передачи тока** (IM 460-0 с IM 46-0 или IM 460-3 с 46-3).

Стойки расширения.

Вы можете:

- устанавливать блоки питания только в слоте 1;
- устанавливать интерфейсный модуль (IM) только на внешнем правом слоте (9 или слот 18);
- устанавливать модули K-шины только в стойке, номер которой не выше 6 (обращение к нему невозможно).

Назначение адресов входов/выходов.

Адреса входов/выходов STEP-7 задает уже при размещении модулей в конфигурационной таблице. Благодаря этому каждый модуль имеет свой начальный адрес (первого канала); адреса остальных каналов получаются из этого начального адреса.

Предпосылки:

- Модуль установлен в центральной стойке или стойке расширения, и CPU допускает свободное назначение адресов;
- Модуль установлен в slave-устройстве DP или сам является slave-устройством DP.

Последовательность действий:

1. Щелкните дважды на строке стойки с модулем, начальный адрес которого вы хотите установить, или выделите

соответствующий модуль и выберите команду меню **Edit > Object Properties (Редактировать > Свойства объекта)**.

2. Выберите закладку **Addresses (Адреса)**.

3. Измените начальный адрес, установленный по умолчанию.

Отображение обзора адресов:

Уже применяемые адреса входов и выходов, а также пропуски адресов вы можете отобразить следующим образом:

1 . Откройте станцию, адреса которой вы хотите просмотреть.

2. Выберите команду меню **View >Address Overview (Вид > Обзор адресов)**.

3. Выделите в диалоговом окне "**Address Overview**" модуль, назначенные которому входы и выходы должны быть отображены.

4. Если необходимо, вы можете отфильтровать отображение по видам адресов (только адреса входов). Отображаются адресные области **Inputs (Входы)** и **Outputs (Выходы)** с указанием размещения для модулей (master-система DP, адрес PROFIBUS, стойка, слот, гнездо для интерфейсного субмодуля). Адреса входов, имеющие нулевую длину (адреса интерфейсных модулей), обозначены звездочкой (*).

Язык программирования AWL

Из языков программирования, с помощью которых можно программировать контроллеры S7, AWL наиболее близок к машинному коду MC7 процессора S7. Это значит, что при его использовании для программирования контроллеров S7, вы можете оптимизировать время исполнения и использования памяти. Язык программирования AWL имеет все необходимые элементы для создания всей программы пользователя. Он содержит обширный набор команд. В вашем распоряжении имеется свыше 130 различных основных команд, а также широкий набор адресов. Функции и функциональные блоки позволяют структурировать вашу программу на AWL, делая ее более обозримой.

Список операторов

Список операторов (англ. Statement List, STL; нем. Anweisungsliste, AWL) – это текстовый язык программирования, который может быть использован для создания операторной части

логического блока. Синтаксис его операторов похож на язык ассемблера и состоит из команд, за которыми следуют адреса (операнды), на которые команда действует. В дальнейшем будет обычно использоваться сокращение AWL.

Программный пакет

Программный пакет AWL – это составная часть стандартного программного обеспечения STEP-7. Это значит, что после установки программного обеспечения STEP-7 вам доступны все функции редактирования, компиляции и тестирования/отладки для AWL. Используя AWL, вы можете создать свою собственную пользовательскую программу:

- с помощью редактора пошагового ввода; при этом ввод структуры локальных данных облегчается с помощью табличных редакторов;
- с помощью исходного файла в текстовом редакторе; при этом ввод текста облегчается с помощью шаблонов блоков.

В стандартном программном обеспечении имеется три языка программирования: STL (AWL), FBD (FUP) и LAD (KOP). Вы можете переходить от одного языка программирования к другому почти без ограничений, выбирая наиболее подходящий язык для конкретного блока, который вы программируете.

Если вы пишете программу в LAD или FBD, то всегда можете перейти к представлению STL. Если вы преобразуете программу на языке LAD в программу на языке FBD и наоборот, то элементы программы, которые не могут быть представлены на целевом языке, отображаются на STL.

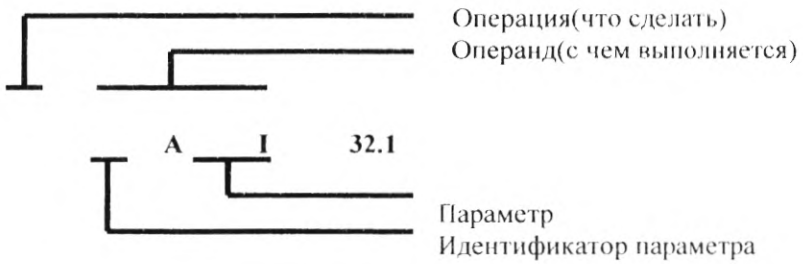
Структура оператора. Компоненты оператора

В зависимости от своей структуры оператор относится к одной из двух следующих основных групп:

- оператор, состоящий только из команды (например, NOT);
- оператор, состоящий из команды и операнда (адреса).

Оператор является наименьшим блоком программы на языке STEP-7. Он выполняет функции задания для работы PLC. Один оператор обычно занимает два байта памяти. Программа состоит из операторов, загружаемых в PLC. Результатом разработки программы является список операторов (AWL).

Структура оператора следующая:



Примечание: Операция PLC задает действие над операндом. Параметр указывает адрес операнда.

Операнды ввода и вывода:

I - Операнд ввода (интерфейс от процесса к PLC);

Q - Операнд вывода (интерфейс от PLC к процессу).

Операнды ввода и вывода имеют адресацию от x.0 до x.7 (x меняется в зависимости от CPU).

Операнд команды. Операнд команды задает константу или адрес, по которому команда находит значение (объект данных). А последним она должна выполнить операцию. Операнд может иметь символическое имя или абсолютное обозначение. Он может указывать на следующие элементы:

- константа, значение таймера или счетчика или строка символов ASCII, которые должны загружаться в аккумулятор 1 (например, L +27);
- бит слова состояния программируемого логического контроллера;
- символическое имя (например, A Motor On);
- блок данных и адрес внутри области этого блока данных (например, L DB4.DBD 0);
- функция (FC), функциональный блок (FB), встроенная системная функция (SFC) или встроенный системный функциональный блок (SFB) и номер функции или функционального блока;

• идентификатор операнда и адрес внутри области памяти, задаваемой идентификатором операнда (например, A I 1.0).

Идентификаторы операндов. Некоторые операнды включают в себя идентификатор операнда и адрес внутри области памяти, указанной в идентификаторе операнда. Идентификатор операнда может быть одного из следующих трех основных типов:

- идентификатор операнда, указывающий область памяти и размер объекта данных в этой области следующим образом:

• область памяти, в которой команда находит значение (объект данных), с которым она должна выполнить операцию (например, «I» для области входов образа процесса);

• размер значения (объекта данных), с которым команда должна выполнить операцию (например, B для «байта», W для «слова» и D для «двойного слова »).

- идентификатор операнда, который указывает область памяти, но не указывает размер объекта данных в этой области (например, идентификатор, указывающий область T для «таймера», C для «счетчика» или DB или DI для «блока данных» плюс номер этого таймера, счетчика или блока данных;

- идентификатор операнда, который указывает размер объекта данных, но не область памяти. Область памяти закодирована в адресе, который следует за идентификатором операнда.

Адресация. Непосредственная адресация

При непосредственной адресации операнд кодируется непосредственно в команде, то есть за командой непосредственно следует значение, с которым должна работать данная команда (например, загрузка). Команда также может предоставить свое собственное значение (например, SET).

```
SET             Установить RLO в 1.
L       27      Загрузить целое число 27 в аккумулятор 1.
L S5T#10S       Загрузить значение времени (10 с.) в
                аккумулятор 1.
```

Прямая адресация

Команда, использующая прямую адресацию, имеет операнд, который указывает местоположение значения, которое команда будет обрабатывать. Этот операнд состоит из следующих частей:

- идентификатора операнда (например, «IB» для «входного байта»);
- точного адреса внутри области памяти, указанной идентификатором операнда

Операнд прямо указывает на адрес значения.

A I 0.0	Выполнить логическую операцию "И" с входным битом I 0.0
= M 115.4	Присвоить RLO биту памяти M 115.4
L MW64	Загрузить слово памяти MW64 в аккумулятор 1.

Операции перехода

Операции перехода могут осуществляться как внутри блока, так и из одного блока в другой. Для осуществления операции перехода внутри блока необходимо предварительно в программе создать метку (любые четыре символа), по которой должен осуществиться переход. Операции перехода могут выполняться как по условию, так и безусловно.

```
JU wer    безусловный переход к метке wer
JC eser   условный переход к метке eser

JU qwer   безусловный переход к метке qwer
          .........
qwer : .........

 A I 12.4
JC asdf   переход к метке asdf  при наличии входа 12.4
          .........
asdf : .........
```

Операции перехода из блока в блок осуществляются также: без условия и по условию.

```
UC FC 1          безусловный вызов функции 1
```

```
AN M 21.7
CC FB 28      вызов функционального блока 28 при условии
              отсутствия маркера 21.7
```

Битовые логические операции

Логические команды, выполняемые над битами, называют также релейными логическими командами, так как они могут выполнять функцию релейной логической схемы. Ниже объясняется, как релейная логическая схема может быть воспроизведена с помощью команд AWL.

Операции установки, сброса и логическое «И».

Операция установки осуществляется с использованием операции «S».

```
A     I     0.0
S     Q     4.0
```

При первой обработке программы, если вход I 0.0 = 1, то выход Q 4.0 устанавливается в «1». Последующее изменение I 0.0 не приводит к изменению этого состояния.

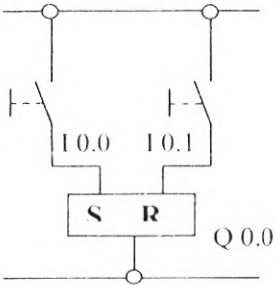
Операция сброса осуществляется с использованием операции «R».

```
A     I     0.0
R     Q     4.0
```

При первой обработке программы, если вход I 0.0 = 1, то выход Q 4.0 устанавливается в «0». Последующее изменение I 0.0 не приводит к изменению этого состояния.

Функция запирания

Запирание осуществляется с использованием операций «S» и «R».




```
A     I     0.0
S     Q     4.0
A     I     0.1
R     Q     4.0
```

Когда контакт I 0.0 кратковременно замыкается, то с выхода Q 4.0 идет сигнал запирания. Когда замыкается контакт I 0.1, то выход Q 4.0 сбрасывается. Выход остается сброшенным, если оба контакта замкнуты одновременно.

Операция равенства

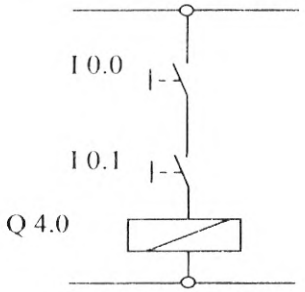
```
A  I    0.0
=  Q    4.0
```

При каждом цикле выполнения программы выход Q 4.0 будет устанавливаться в состояние согласно текущему состоянию входа I 0.0.

Основные булевы логические операции.

Функция «И»

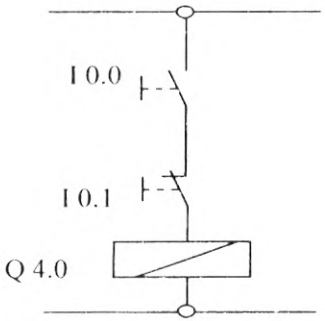
С помощью этой операции опрашивается одновременность выполнения различных условий.




```
A     I     0.0
A     I     0.1
=     Q     4.0
```

На выходе Q 4.0 индицируется «1», если на всех входах присутствует «1». На выходе Q 4.0 «0», если хотя бы на одном из входов «0».

Функция «И НЕ»

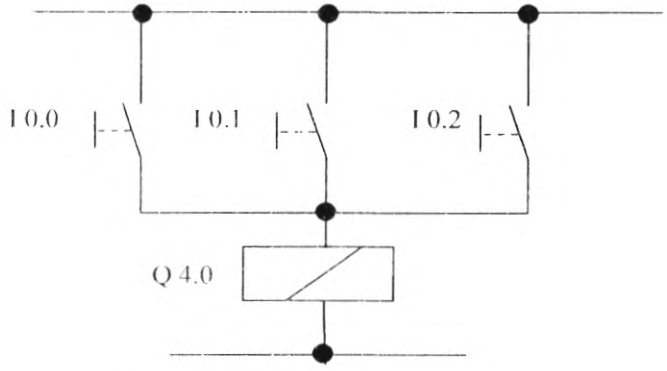



```
A     I     0.0
AN    I     0.1
=     Q     4.0
```

На выходе Q 4.0 индицируется «1», если на входе I 0.1 присутствует «0», а на входе I 0.2 – «1».

Функция «ИЛИ»

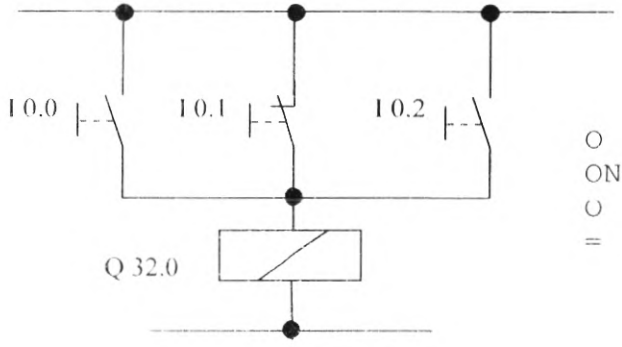
С помощью этой операции выясняется, выполняется ли хотя бы одно из нескольких условий.




```
O     I     0.0
O     I     0.1
O     I     0.2
=     Q     4.0
```

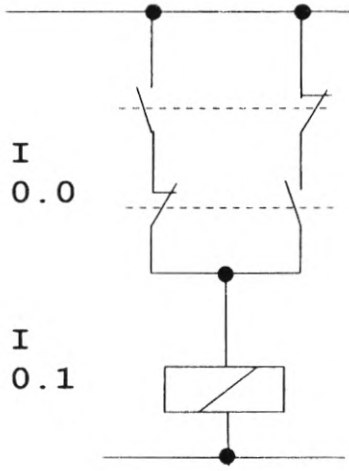
На выходе Q 4.0 – «1», если, по крайней мере, на одном из входов «1». На выходе Q 4.0 – «0», если на всех входах одновременно «0».

Функция «ИЛИ НЕ»




```
O     I     0.0
ON    I     0.1
O     I     0.2
=     Q     4.0
```

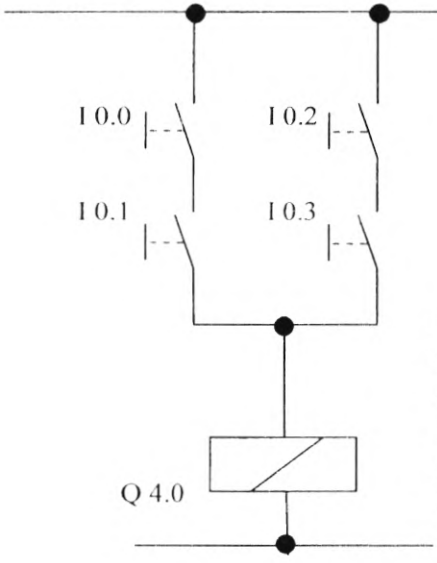
Функция «Исключающее ИЛИ»




```
X     I     0.0
X     I     0.1
=     Q     4.0
```

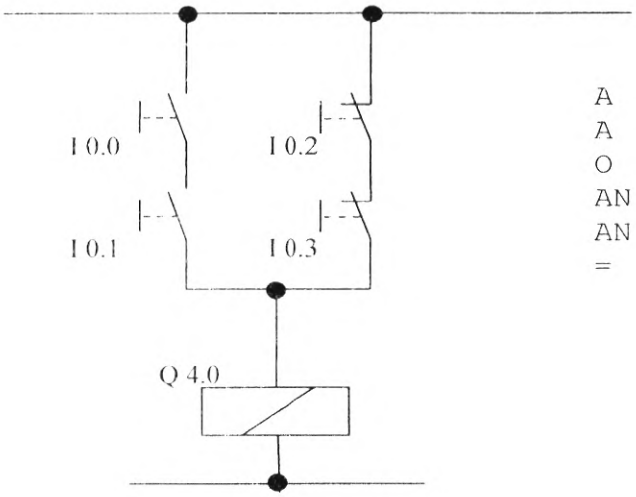
На выходе Q 4.0 – «1», если I 0.0

Операция «И» перед «ИЛИ»



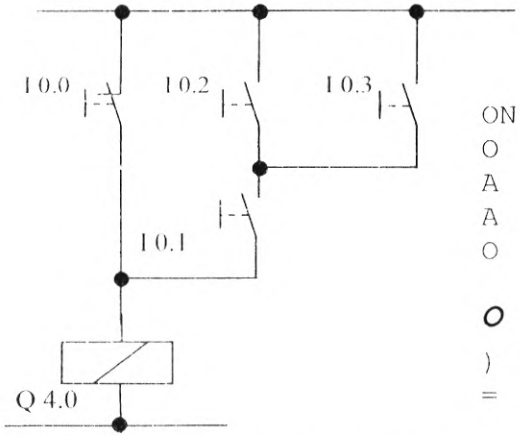

```
A     I     0.0
A     I     0.1
O
A     I     0.2
A     I     0.3
=     Q     4.0
```

На выходе Q 4.0 – «1», если, по крайней мере, одна из операций «И» выполнена (имеет на выходе «1»). На выходе Q 4.0 – «0», если ни одна из операций «И» не выполнена.




```
A     I     0.0
A     I     0.1
O
AN    I     0.2
AN    I     0.3
=     Q     4.0
```

Операция «ИЛИ» перед «И»



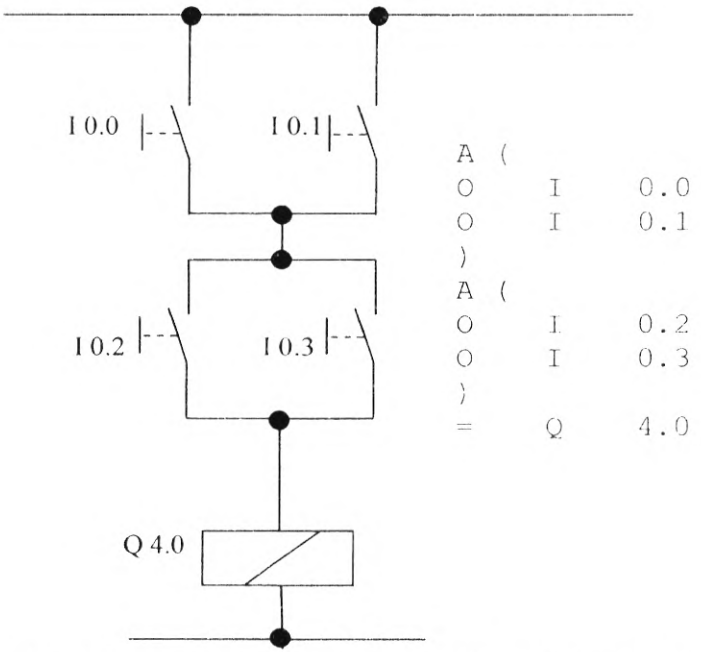

```
ON    I     0.0
O
A     I     0.1
A     (
O     I     0.2
O     I     0.3
)
=     Q     4.0
```

На выходе Q 4.0 будет сигнал «1», если выполнено, по крайней мере, одно из условий:

- на входе I 0.0 сигнал «0»;

- на входе I 0.1 и на одном из входов I 0.2 или I 0.3 установлен сигнал «1».

На выходе Q 4.0 будет «0», если ни одна из операций «И» не выполнена.



На выходе Q 4.0 – «1» только в том случае, если обе операции «ИЛИ» выполнены. На выходе Q 4.0 – «0», если не выполнена хотя бы одна операция «ИЛИ».

Маркер

M (Маркер) - элемент памяти для записи результата в дискретном виде. Маркеры имеют адресацию от x.0 до x.7 (x меняется в зависимости от CPU).

A I 0.0 маркер M 0.6 фиксирует A M 0.6 состояние маркера M 0.6

A I 0.1 выполнение обоих условий S Q 0.0 можно опрашивать в других S M 0.6 частях программы.

Счетчики

Счетчик – это функциональный элемент языка программирования STEP-7, предназначенный для счета. Счетчики имеют зарезервированную область памяти в CPU. Набор команд списка операторов поддерживает 256 счетчиков. Чтобы выяснить количество счетчиков, доступных в CPU, обратитесь к техническим данным CPU. Операции со счетчиками являются единственными функциями, имеющими доступ к области памяти, зарезервированной для счетчиков.

С помощью операций счета выполняется задание счета непосредственно от CPU. Имеется возможность счета, как на сложение, так и на вычитание. Область счета лежит между 0 и 999.

Имеющиеся в распоряжении команды

Представление языка программирования STEP-7 в виде списка операторов предоставляет в распоряжение следующие команды:

- Установка (S);
- Сброс (R);
- Прямой счет (CU);
- Обратный счет (CD);
- Разблокировка счетчика (FR);
- Загрузка счетчика в одном из следующих форматов: двоичный (L), двоично - десятичный (LC);
- Опрос состояния сигнала счетчика и логическое сопряжение результата опроса с помощью булевой логической операции (A, AN, O, ON, X, XN). Опрос состояния сигнала с помощью команды A, O или X дает результат, равный «1», когда значение счетчика больше нуля. Опрос состояния сигнала с помощью команды A, O или X дает результат, равный «0», когда значение счетчика равно нулю.

Для установки счетчика в программе включите в нее три оператора для реализации следующих операций:

- Опрос состояния сигнала на равенство 0 или 1 (A I 2.3)
- Загрузка значения счетчика (L C# 3) в младшее слово аккумулятора 1

- Установка счетчика со значением, которое вы загрузили (S C 1). Эта операция пересылает значение счетчика из аккумулятора 1 в слово счетчика.

В программе изменение результата логической операции (RLO) с 0 на 1 перед командой *Установить* (S) устанавливает счетчик на запрограммированное счетное значение. Запрограммированное счетное значение и команда установки должны следовать сразу за логической операцией, определяющей условия установки счетчика.

Счетчик сбрасывается с помощью команды *Сбросить* (R). CPU сбрасывает счетчик, когда результат логической операции равен 1 непосредственно перед командой R в программе. Пока RLO перед оператором R равен 1, команда A, O или X, опрашивающая состояние сигнала счетчика, дает результат, равный 0, а команда AN, ON или XN – результат, равный 1. Если счетчик должен сбрасываться статическим сигналом на входе сброс (R) и независимо от RLO других команд счетчика, вы должны записать оператор сброса непосредственно после оператора установки, прямого счета или обратного счета и перед опросом сигнала или операцией загрузки

Изменение с 0 на 1 результата логической операции перед командой *Разблокировать* (FR) разблокирует счетчик. CPU выполняет команду FR только при положительном фронте сигнала. Разблокировка счетчика не требуется ни для установки счетчика, ни для нормального счета. Разблокировка используется только для того, чтобы устанавливать счетчик или производить прямой или обратный счет, если перед соответствующим оператором счета требуется положительный фронт сигнала (переход из 0 в 1) и если бит RLO перед соответствующим оператором имеет состояние сигнала 1.

Прямой счет

```
A     I     0.0
CU    C     1
```

При положительном фронте сигнала значение содержимого счетчика увеличивается на 1. При RLO (результат логической операции)=0 значение счетчика остается без изменений.

Обратный счет

```
A     I     0.1
CD    C     1
```

При положительном фронте сигнала значение содержимого счетчика уменьшается на 1. При RLO=0 значение счетчика остается без изменений.

Сброс счетчика

```
A     I     0.2
R     C     1
```

Счетчик остается установленным в «0» до тех пор, пока RLO=1.

Установка счетчика

При установке счетчика, значение данных, находящееся в аккумуляторе AKKU1, воспринимается как значение счетчика. Поэтому сначала в аккумулятор следует загрузить значение счетчика.

```
L     C#37          значение от 0 до 999
A     I     0.3
S     C     1
```

Операции загрузки и передачи

Команды загрузки (L) и передачи (T) позволяют программировать обмен информацией между модулями ввода или вывода и областями памяти или между областями памяти. CPU выполняет эти команды в каждом цикле как безусловные команды, т.е. результат логической операции на них не влияет.

Команды L и T производят обмен информацией через аккумулятор. Команда L записывает (загружает) содержимое своего исходного адреса в аккумулятор 1, сдвигая всю уже содержащуюся там информацию в аккумулятор 2. Старое содержимое аккумулятора 2 заменяется. Команда T копирует содержимое аккумулятора 1 и записывает его в соответствующую целевую память. Поскольку

команда T только копирует информацию, находящуюся в аккумуляторе 1, то эта информация остается доступной для других команд. Команды L и T могут обрабатывать информацию байтами (8 битов), словами (16 битов) и двойными словами (32 бита). Аккумулятор содержит 32 бита. Данные длиной менее 32 битов располагаются в аккумуляторе справа. Остальные биты аккумулятора заполняются нулями.

L X	загрузка константы X в ACCU 1
L Y	загрузка константы Y в ACCU 1, X переходит в ACCU 2
L IB 16	загрузка байта входов 16 в аккумулятор 1, Y переходит в ACCU 2 константа X утеряна
T QB 20	передача содержимого аккумулятора 1 в байт выходов 20

Операнды команд загрузки: непосредственная адресация

Операнд	Пример	Пояснение
±..	L +5	***Загружает 16-битовую целую константу в аккумулятор 1***
B#(.., ..)	L B#(1,10)	Загружает константу как 2 байта в аккумулятор 1. (В этом примере 10 поступает в младший байт младшего слова аккумулятора 1; 1 поступает в старший байт младшего слова аккумулятора 1)
	L B#(1,10,5,4)	Загружает константу как 4 байта в аккумулятор 1. (В этом примере 4 и 5 поступают соответственно в младший и старший байт младшего слова аккумулятора 1; 10 и 1 поступают соответственно в младший и старший байт старшего слова аккумулятора)
L#..	L#+5	Загружает 32–битовую целую константу в аккумулятор 1

16#..	L B#16#EF	Загружает 8–битовую шестнадцатеричную константу в аккумулятор 1
	L W#16#FAFB	Загружает 16–битовую шестнадцатеричную константу в аккумулятор 1
	L DW#1 6#1 FFE_1 ABC	Загружает 32–битовую шестнадцатеричную константу в аккумулятор 1
2#..	L 2#1111 _0000 _1111 _0000	Загружает 16–битовую двоичную константу в аккумулятор 1
	L 2#1111 _0000 _1111_0000 _1111_0000 _1111_0000	Загружает 32–битовую двоичную константу в аккумулятор 1
'..'	L 'AB'	Загружает 2 символа в аккумулятор 1
	L 'ABCD'	Загружает 4 символа в аккумулятор 1
C#..	L C#1 000	Загружает 16–битовую константу счетчика в аккумулятор 1
S5TI ME#. .	L S5TIME#2S	Загружает 16–битовую константу S5TIME в аккумулятор 1
..	L 1.0E+5	Загружает 32–битовое число с плавающей точкой в формате IEEE в аккумулятор 1
P#..	L P#I.0	Загружает 32–битовый указатель в аккумулятор 1
	L P##Start	Загружает 32–битовый указатель на локальную переменную (Start) в аккумулятор 1
D#..	L D#1994–3–15	Загружает 1 6–битовую дату в аккумулятор 1
T#..	L T#0D_1 H_1 M_0S_0MS	Загружает 32–битовое значение времени в аккумулятор 1
TOD #..	L TOD#1:10:3.3	Загружает 32–битовое значение времени суток в аккумулятор 1

Команды L и T могут непосредственно обращаться к байту (B), слову (W) или двойному слову (D).

Блоки данных

Служат для хранения, как постоянных коэффициентов, так и временной информации (считанных значений аналоговых входных модулей, значений, предназначенных для передачи на модули аналоговых выходов, состояния таймеров, счетчиков, маркеров). В блоках данных **не могут** обрабатываться команды языка STEP-7. Максимально возможное количество блоков данных - 256 (DB0-DB255). Максимальная длина блока данных - 256 слов данных.

Перед работой с определенным блоком данных необходимо его открыть!

OPN DB 3 Открытие блока данных 3
L +127 Загрузка константы 127
T DBW 12 Передача константы 127 в слово данных 12

Блок данных остается действительным до тех пор, пока не будет вызван другой блок данных.

При возврате в блок, из которого был сделан переход, действительным снова становится тот блок данных, который был действительным до перехода.

Операции сравнения

Вы загружаете числовые значения в аккумуляторы 1 и 2. Команда *Сравнить* сравнивает значение в аккумуляторе 1 со значением в аккумуляторе 2. Результатом сравнения является двоичная цифра, т.е. 1 или 0. 1 указывает, что результат сравнения является истиной; 0 указывает, что результат сравнения является ложью. Этот результат вы можете использовать в своей программе для дальнейшей обработки.

CPU выполняет команды сравнения независимо от результата логической операции. Содержимое аккумуляторов при этом не изменяется.

A I 0.0 CU C1	счетчик C1 считает количество появлений входа I 0.0
LC C1	загрузить значение счетчика C1 в аккумулятор
L C#10	загрузить константу 10 в формате счета в аккумулятор
==I	сравнить содержимое аккумуляторов

= Q 0.0 если условие выполняется выставить выход Q 0.0

Операция	Описание
= =I	Сравнивает содержимое аккумуляторов на «равенство». Если ACCU1=ACCU2, то RLO=1
< >I	Сравнивает содержимое аккумуляторов на «неравенство». Если ACCU1 не «равно» ACCU2, то RLO=1
>I	Сравнивает содержимое аккумуляторов на «больше». Если ACCU2 больше ACCU1, то RLO=1
>=I	Сравнивает содержимое аккумуляторов на «больше или равно». Если ACCU2>или = ACCU1, то RLO=1
<I	Сравнивает содержимое аккумуляторов на «меньше». Если ACCU2 меньше ACCU1, то RLO=1
<=I	Сравнивает содержимое аккумуляторов на «меньше или равно». Если ACCU2<или = ACCU1, то RLO=1

Арифметические операции

С помощью арифметических операций содержимое аккумуляторов интерпретируются как целые числа и над этими числами выполняются соответствующие арифметические операции.

Операция	Описание
+I	Складывает содержимое младших слов аккумуляторов 1 и 2 и сохраняет результат в младшем слове аккумулятора 1 .
-I	Вычитает содержимое младшего слова аккумулятора 1 из содержимого младшего слова аккумулятора 2 и сохраняет результат в младшем слове аккумулятора 1.
*I	Перемножает содержимое младших слов аккумуляторов 1 и 2 и сохраняет результат (32 бита) в аккумуляторе 1 .
/I	Делит содержимое младшего слова аккумулятора 2 на содержимое младшего слова аккумулятора 1. Результат сохраняется в младшем слове аккумулятора 1. Целый остаток от деления сохраняется в старшем слове аккумулятора 1 .

L 4	загрузить константу 4 в аккумулятор
L 15	загрузить константу 15 в аккумулятор
+ I	сложить содержимое аккумуляторов
T DB1.DBW9	перенести результат в слово данных 9 блока данных 1

Таймеры

Таймер – это элемент языка программирования STEP-7, который реализует и контролирует процессы, управляемые временем. Таймерные команды дают программе возможность выполнять следующие функции:

- обеспечение времени ожидания (например, согласно технологии литья под давлением форма должна оставаться закрытой в течение двух секунд. Ваша программа обеспечивает то, что деталь, отлитая под давлением, будет извлечена из формы по истечении двух секунд);
- обеспечение времени контроля (например, программа контролирует скорость двигателя в течение 30 секунд после нажатия пусковой кнопки);
- генерирование импульсов (например, ваша программа подает импульсы, вызывающие мигание лампы);
- измерение времени (например, ваша программа может определить, сколько времени занимает наполнение резервуара).

Имеющиеся в распоряжении команды

Представление языка программирования STEP-7 в виде списка операторов предоставляет в распоряжение следующие команды:

Запуск таймера одного из следующих типов:
- формирователя импульса (SP)
- формирователя удлиненного импульса (SE)
- формирователя задержки включения (SD)
- формирователя задержки включения с запоминанием (SS)
- формирователя задержки выключения (SF)

Сброс таймера (R)

Разрешение на запуск (разблокировка) таймера (FR)

Загрузка таймера в одном из следующих форматов:

- в двоичном коде (L)

- в двоично - десятичном коде (LC)

Опрос состояния сигнала таймера и логическое сопряжение результата опроса с помощью булевой логической операции (A, AN, O, ON, X, XN).

Значение времени

Вы должны предварительно загрузить значение времени

L S5T# aH_bbM_ccS_dddMS

где a = часы , bb = минуты , cc = секунды и ddd = миллисекунды

Например: L S5T#2H30M – 2 часа 30 минут

L S5T#5S – 5 секунд

База времени выбирается автоматически и значение округляется до ближайшего меньшего числа с этой базой времени .

Максимальное значение времени, которое вы можете ввести, равно 9 990 секунд или 2H_46M_30S.

Для запуска таймера в своей программе включите три оператора для реализации следующих операций:

- Опрос состояния сигнала на равенство 0 или 1 (A I 2.0);

- Загрузка значения времени и соответствующей базы времени (L IW 0);

- Запуск таймера одного из следующих типов:

- формирователя импульса (SP, например , SP T 1);
- формирователя удлиненного импульса (SE);
- формирователя задержки включения (SD);
- формирователя задержки включения с запоминанием (SS);
- формирователя задержки выключения (SF).

В программе изменение результата логической операции (RLO) перед командой запуска приводит к запуску таймера. Изменение RLO с 1 на 0 запускает таймер в качестве формирователя задержки выключения (SF); изменение с 0 на 1 запускает любой из остальных таймеров. Запрограммированное время и операторы запуска таймера должны следовать непосредственно за операцией, определяющей условие запуска таймера.

Поскольку таймер работает в обратном направлении от установленного времени к нулю, то вы должны снабдить таймер стартовым временем. Когда вы запускаете таймер в своей

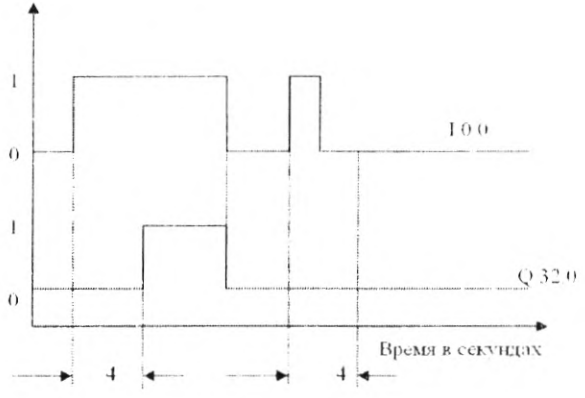
программе, CPU ищет стартовое время в аккумуляторе 1. Диапазон времени составляет от 0 до 9 990 секунд.

Таймер сбрасывается с помощью команды *Сбросить* (R). CPU сбрасывает таймер, если в программе результат логической операции непосредственно перед командой R равен 1. Пока RLO перед командой R равен 1, команда A, O или X, опрашивающая состояние сигнала таймера, дает результат, равный 0, а команда AN, ON или XN дает результат, равный 1. Сброс таймера останавливает текущее функционирование таймера и сбрасывает значение времени в 0.

Смена результата логической операции с 0 на 1 перед командой *Разблокировать* (FR) разблокирует таймер. Эта смена состояния сигнала всегда необходима для разблокировки таймера. CPU выполняет команду FR только при положительном фронте сигнала. Для запуска или нормальной работы таймера разблокировка не требуется. Разблокировка используется только для перезапуска работающего таймера. Такой перезапуск возможен только, когда операция запуска продолжает обрабатываться с RLO, равным 1.

Таймер задержки включения «SD»

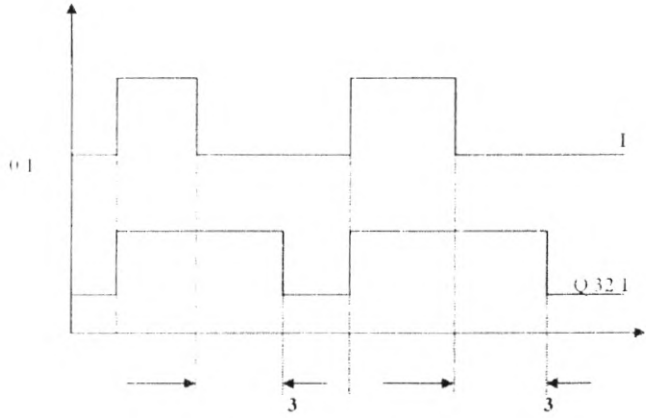
Выходной сигнал Q 32.0 запитывается (выдает сигнал) через 4 секунды после появления сигнала на входе I 0.0. Сигнал выхода считается запитанным до тех пор, пока на входе «1».



```
A     I     0.0
L     S5T#4S
SD    T     1
A     T     1
=     Q     32.0
```

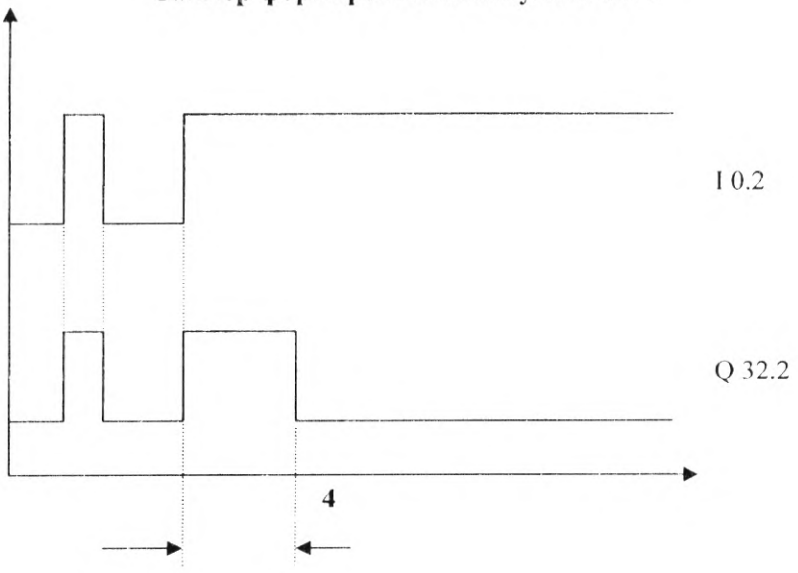
Таймер задержки выключения «SF»

Выход (Q 32.1) распитывается (сигнал пропадает) через 3 секунды после исчезновения сигнала на входе (I 0.1). Появление сигнала на выходе происходит одновременно с появлением сигнала на входе.



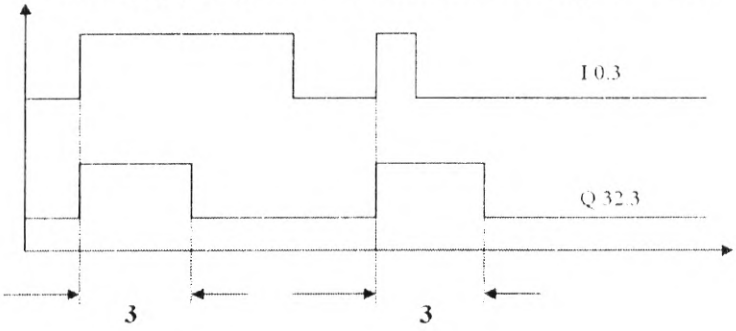

```
A     I     0.1
L     S5T#3S
SF    T     2
A     T     2
=     Q     32.1
```

Таймер формирователь импульса «SP»




```
A     I     0.2
L     S5T#4S
SP    T     3
A     T     3
=     Q     32.2
```

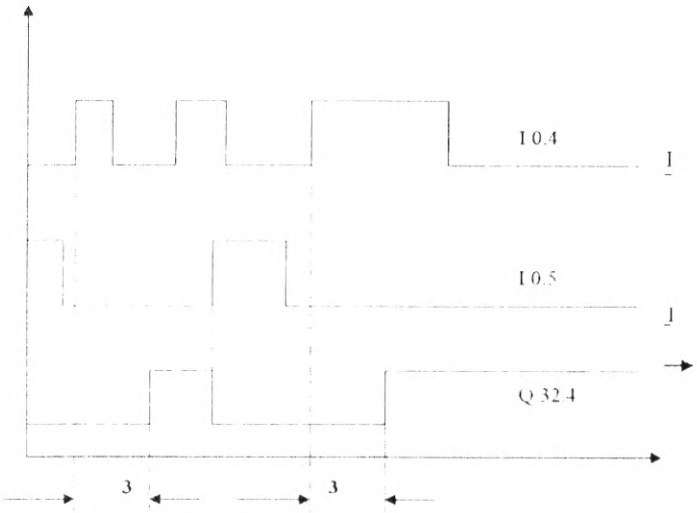
Таймер формирователь удлиненного импульса «SE»



```
A     I     0.3
L     S5T#3S

SE    T     4
A     T     4
=     Q     32.3
```

Таймер задержки включения с запоминанием "SS"




```
A     I     0.4
L     S5T#3S
SS    T     5
A     I     0.5
R     T     5
A     T     5
=     Q     32.4
```

Операнды Step-7

Операнд		Назначение	Пояснение
Английский	Немецкий		
I	***E***	Вход	Интерфейс от процесса к PLC.
Q	A	Выход	Интерфейс от PLC к процессу.
M	M	Маркер	Память для двоичных результатов промежуточных операций.
L		Локальные данные	Память для двоичных результатов промежуточных операций в пределах одного блока.
D		Данные	Память для цифровых результатов промежуточных операций.
T		Таймеры	Память для реализации функций таймеров.
C	Z	Счетчики	Память для реализации функций счетчиков.
P		Периферия	Интерфейс от процесса к PLC.
OB,FC,SFC,FB,SFB,DB		Блок	Вспомогательные средства для структурирования программы.

Логические операции

Операция		Пояснение
Английский	Немецкий	
A	U	Операция «И», опрос сигнала на 1
O		***Операция «ИЛИ», опрос сигнала на 1***
AN	UN	Операция «И», опрос сигнала на 0
ON		Операция «ИЛИ», опрос сигнала на 0
X		Операция «исключающее ИЛИ», опрос сигналов на различие

Операции памяти

Операция	Описание
S	Установка
R	Сброс
=	Назначение

Операции загрузки и передачи

Операция	Описание
L	**Загрузка.** Содержимое операнда, независимо от **RLO**, копируется в аккумулятор **AKKU1. RLO** не изменяется
T	**Передача.** Содержимое аккумулятора **AKKU1,** независимо от **RLO**, передается в операнд. **RLO** не изменяется.

Функции таймеров

Операция		Описание
Английский	Немецкий	
SP	SI	Запуск таймера в виде импульса.
SE	SV	Запуск таймера в виде удлиненного импульса.
SD	SE	Запуск таймера в виде задержки включения.
SS		Запуск таймера в виде задержки включения с запоминанием.
SF	SA	Запуск таймера в виде задержки выключения.
FR		Разблокировка
R		Сброс таймера.

Операции со счетчиками

Операция		Описание
Английский	Немецкий	
S		**Установка счетчика**
FR		Разблокировка
R		***Сброс счетчика***
CU	ZV	Счет вперед
CD	ZR	Счет назад

ГЛАВА 4
БАЗОВЫЕ АЛГОРИТМЫ УПРАВЛЕНИЯ

4.1. Концепции выполнения программы контроллером S7-200

Прежде чем приступать к алгоритмизации рекомендуется выявить концепцию выполнения программы контроллером, в котором будут реализованы алгоритмы.

Связь программы с входами и выходами

Основная работа центрального процессора (CPU) контроллера S7-200:

- CPU считывает значения входов;
- Программа, хранимая в CPU, использует эти значения для вычисления значений выходов. Во время выполнения программы CPU обновляет данные;
- CPU записывает данные в выходы.

Цикл сканирования CPU

CPU S7-200 предназначен для циклического выполнения ряда заданий, включая программу пользователя. Такое циклическое выполнение заданий называется циклом сканирования. В течение цикла сканирования, CPU выполняет все или большинство из следующих задач:

- считывание значений;
- выполнение программы;
- обработка коммуникационных запросов;
- выполнение самодиагностики CPU;
- запись в выходы.

Считывание цифровых входов

Каждый цикл сканирования начинается со считывания текущих значений цифровых входов и последующей записи этих значений в регистр входов образа процесса.

Выполнение программы

В фазе выполнения CPU реализует программу, начиная с первой команды и до последней

Обработка коммуникационных запросов

Во время фазы обработки CPU обрабатывает запросы, принятые из коммуникационного порта.

Выполнение самодиагностики CPU

Во время этой фазы CPU проверяет свое встроенное программное обеспечение и память программы пользователя. Он проверяет также состояние всех модулей ввода-вывода.

Запись в цифровые выходы

В конце каждого цикла сканирования CPU записывает значения, хранимые в регистре выходов образа процесса, в цифровые выходы.

Основные положения

· входная информация остается неизменной на всем протяжении цикла выполнения CPU;

· CPU выполняет программу последовательно, начиная с первой команды и до конечной;

· обработка коммуникационных запросов от/в TD200 и от/в ProfiBus, и, соответственно, использование информации, считываемой из памяти контроллера, а также формирование новой информации при обработке коммуникационных запросов, производится после выполнения программы пользователя;

· CPU записывает значения, хранимые в регистре выходов образа процесса, в цифровые выходы в конце цикла сканирования – после выполнения программы пользователя и обработки коммуникационных запросов.

4.2. Виды и типы алгоритмов управления

Алгоритмизация управления производится с применением **SWITCH-технологии®. Функционально** предлагается выделять следующие **виды** алгоритмов управления:

- **технологические алгоритмы (автоматы);**
- **алгоритмы (автоматы) управления приводами.**

Технологический автомат – это автомат, реализующий логику **технологического процесса**. Это - наиболее сложный и объемный вид алгоритмов. Основная функция – **формирование команд для управления приводами.** Технологический алгоритм должен не только корректно реализовывать последовательность операций по управлению приводами, но и не менее корректно реагировать на ненормальную работу приводов.

Автомат управления приводом – это автомат, реализующий логику **непосредственного управления приводом по командам управления и контроль состояния привода.** Этот вид алгоритмов является базовым для многих систем управления технологическими процессами. Он не зависит от логики технологического процесса, однако учитывает особенности управления приводом (в частности, вид управления приводом – потенциальное или импульсное). Основная функция – **формирование сигналов в схему управления** пускателем (контактором). **Формирование информации о ненормальной работе** (реакции) привода – вторая функция, которая **необходима для корректной реализации технологического алгоритма.**

Таким образом, **технологический алгоритм** решает задачу "КОГДА" привод должен управляться, **а алгоритм управления приводом** обеспечивает реализацию управления, попутно определяя "ПОЧЕМУ НЕ ПОЛУЧИЛОСЬ" и сигнализируя о факте ненормальной работы привода. Причин "ПОЧЕМУ НЕ ПОЛУЧИЛОСЬ" может быть несколько, но для того чтобы правильно отреагировать на отклонение от нормальной работы достаточно самого этого факта, а конкретная причина здесь не важна.

По способу реализации предлагается выделять следующие **типы** алгоритмов управления:

- **непосредственный,** который использует и формирует непосредственные данные, команды, сигналы;

· **подпрограмма** – алгоритм, использующий значения входных формальных параметров и формирующий значения выходных формальных параметров.

4.3. Разработка алгоритмов управления

Базовыми являются автоматы управления приводами. Управляемые приводы, как правило, подразделяют на **нереверсивные** (приводы насосов, агрегатов, вентиляторов и т.д.) и **реверсивные** (приводы задвижек, клапанов, конвейеров и т.д.). Собственно управление (выдача сигналов из контроллера в схему управления приводом) обычно бывает **потенциальным** и **импульсным** (выбор способа управления зависит от используемой схемы управления).

Автоматы управления приводами **будут выполняться** (вызываться) **всегда** независимо от состояния технологического автомата управления.

4.4. Автомат управления нереверсивным приводом

Схемы связей автомата. В соответствии со SWITCH-технологией **схема автомата** состоит из двух частей – **схемы связей** и **графа переходов**.

В схеме связей автомата, задающей его интерфейс, перечисляются наименования и обозначения входных и выходных воздействий. **В качестве редактора диаграмм рекомендуется применять пакет** Visio.

На рис. 1 приведен пример схемы связей для рассматриваемого автомата.

Наименования автомата, входных и выходных воздействий могут быть произвольными. Рекомендуется продумать **систему обозначений** перед созданием алгоритмов. **Обозначения** входных и выходных воздействий в схеме связей и графе переходов должны **строго соответствовать.**

Автомат управления приводом по типу является подпрограммой, поскольку чаще всего предназначен для управления несколькими аналогичными приводами. Входную информацию рекомендуется "приводить" к булевой, даже если это просто сравнение двух чисел. Таким образом, в автомате все

условия являются булевыми формулами. Это намного упрощает разработку, анализ и тестирование.

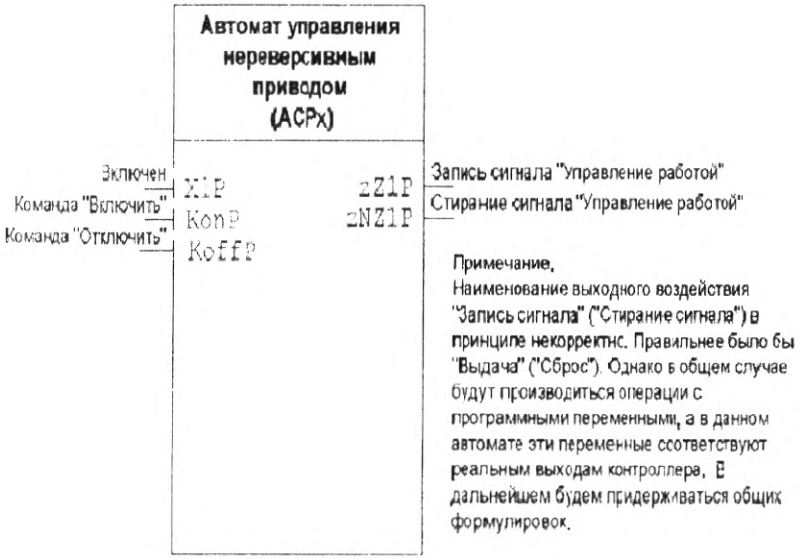


Рис. 1. Схема связей автомата.

В соответствии с принципами автоматного управления в общем случае выходными воздействиями автомата являются признаки выполнения процедур – чаще всего простейших операций по записи/стиранию (установке/сбросу) булевых переменных.

При этом должно соблюдаться важное условие: **значение любой переменной** по возможности **формируется только в одном автомате.** Если это невозможно (например, большой алгоритм разбит на несколько), то необходимо, чтобы такие алгоритмы не выполнялись в одном цикле программы, поскольку в противном случае значение переменной может быть "затерто" последующими процедурами.

4.5. Построение графа переходов автомата

Граф переходов, реализующий функцию потенциального управления нереверсивным приводом, приведен на рис. 2.

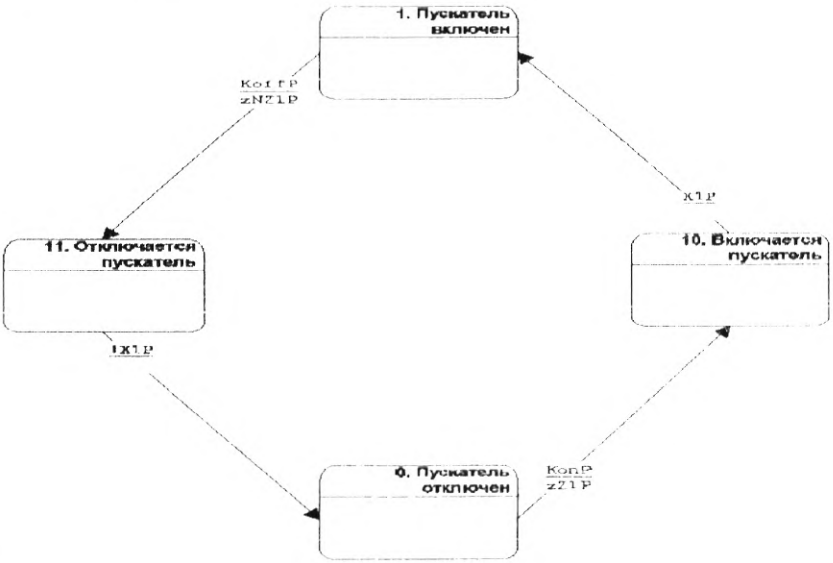


Рис. 2. Граф переходов, реализующий функцию потенциального управления нереверсивным двигателем.

Номера и наименования состояний – произвольные. Рекомендуется "выделять" специальными номерами, например, начальные и конечные состояния.

4.6. Дополнение функции автомата.

Выделим в автомате два типа состояний, первые из которых от времени не зависят, а вторые – зависят. В рассматриваемом примере к состояниям первого типа относятся:

0. Пускатель отключен

1. Пускатель включен

а второго –

10. Включается пускатель

11. Отключается пускатель

Такое деление состояний не является необходимым. Например, в технологическом автомате практически все состояния зависят от времени. Делается это только для того, чтобы подчеркнуть зависимость пребывания в состоянии от времени.

В данном случае ясно, что сигнал на включение пускателя (и соответственно, привода) не будет выдан контроллером пока нет соответствующей команды, а момент выдачи команды практически неизвестен. Теоретически ожидание в состоянии 0 может быть неограниченно по времени.

С другой стороны, в состоянии 10 ясно, что через определенное время после подачи напряжения реле пускателя "сработает", появится сигнал X1P (пускатель включен) от контакта реле схемы управления и будет возможен переход в состояние 1. Это - наиболее реальная реакция реле пускателя.

Важнейшим дополнением условий выхода из состояний второго типа является **контроль времени.**

Если выдача команды является результатом действий человека или логических операций в технологическом автомате и имеет субъективный характер, то появление сигнала X1P (пускатель включен) объективно ожидаемая (нормальная) и наиболее вероятная реакция. Вместе с тем необходимо предусмотреть возможность ненормальной реакции объекта управления на выдаваемые контроллером сигналы, поскольку оборудование очень часто может отказать. Естественно, что это делается только в случае, когда предполагается автоматическое управление технологическим процессом. Поэтому дополним схему связей и граф переходов функциями контроля времени в состояниях второго типа (рис. 3, 4).

В графе переходов на рис. 4 в каждом из состояний 10 и 11 переходы могут быть противоречивы – выполняться одновременно. Это может быть устранено расстановкой приоритетов.

Рассматриваемый автомат – это весьма простой автомат потенциального управления нереверсивным приводом. Его можно еще более упростить, если исключить контроль отключения пускателя по времени (в отличие от не включения, вероятность не отключения мала). При этом если пускатель не отключился, то система будет ожидать отключения и ничего более. Решение об упрощении должно быть хорошо продуманным.

Рис. 3. Схема связей автомата, дополненного функциями контроля времени в состояниях второго типа.

Однако автомат лучше не упрощать, а наоборот, добавить в него несколько переходов и еще одно состояние, которые повысят информативность контроля, особенно при наладке, когда вероятны не только логические, но и монтажные ошибки (рис. 5, 6).

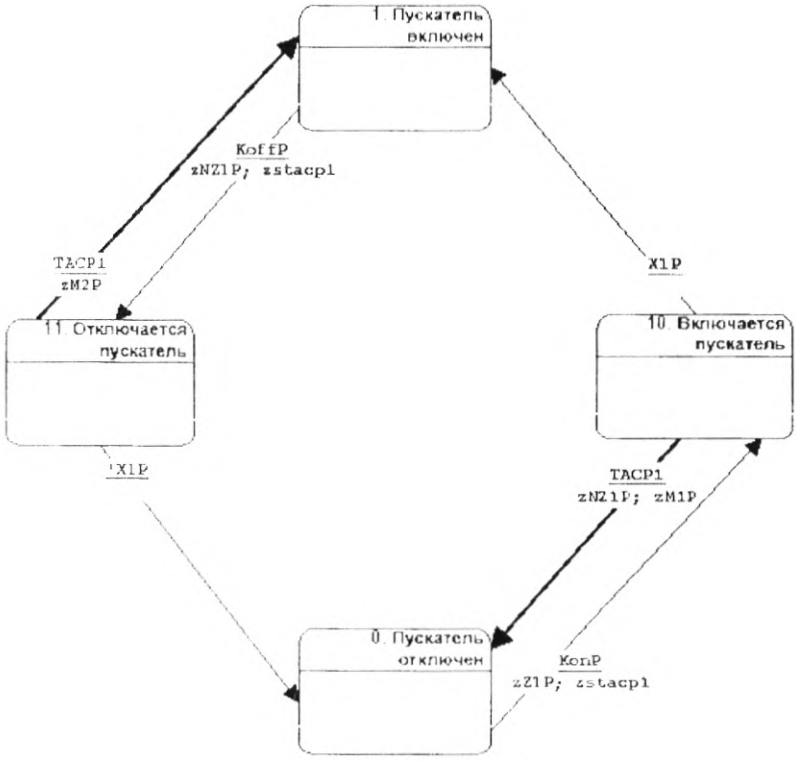


Рис. 4. Граф переходов автомата, дополненного функциями контроля времени в состояниях второго типа.

Система управления создается не только **для освобождения человека от рутинных или динамичных операций,** но и **для контроля работы оборудования.** Поэтому высокая информативность системы позволит в дальнейшем быстрее и качественнее определить причину сбоя в технологическом процессе.

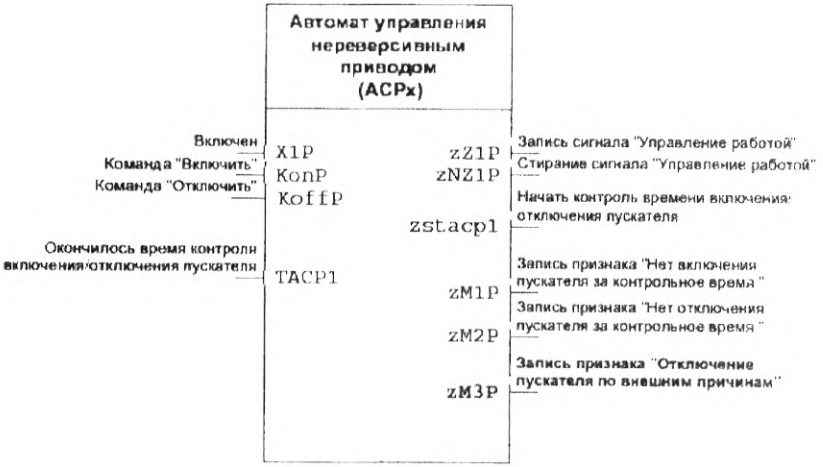


Рис. 5. Схема связей автомата управления нереверсивным приводом.

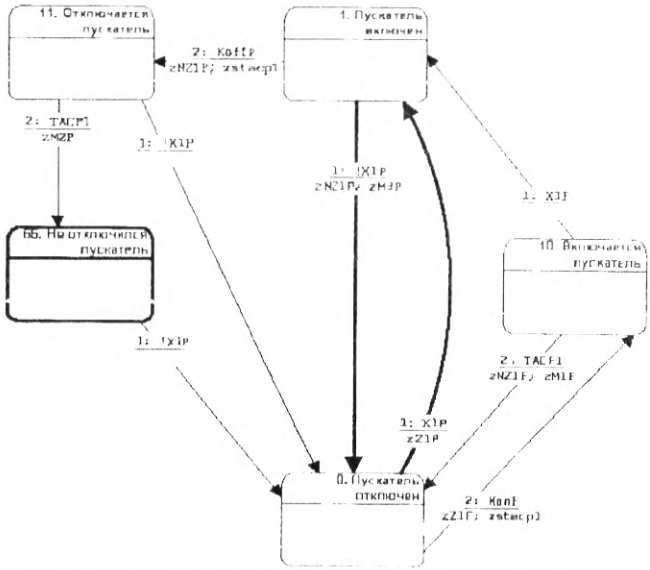


Рис. 6. Граф переходов автомата управления нереверсивным приводом.

Этот граф переходов отличается от предыдущего (рис. 4) еще и тем, что в нем используются приоритеты проверки условий для перехода в другое состояние – перед условием приписана цифра. При этом, **чем меньше цифра, тем выше приоритет проверяемого условия.** Понятно, что если выполнилось условие с более высоким приоритетом, то условия с более низким уже не проверяются, поскольку произойдет изменение состояния.

Отметим важность перехода с приоритетом 1 из состояния 0 в состояние 1. Если схема управления позволяет включить привод в обход контроллера при режиме управления от контроллера, или включить привод, а потом произвести перевод в режим управления от контроллера, то контроллеру необходимо "подхватить" управление приводом (если привод управляется потенциально).

Ниже на рис.7, 8 приводятся схема связей и граф переходов автомата управления реверсивным приводом.

Решения, принятые при построении этого автомата.

1. Как следует из графа переходов, устранены состояния ожидания отключения пускателя (контактора). Это сделано исключительно для упрощения и уменьшения графа и основано на том, что в схеме управления реверсивным приводом (например, задвижки или клапана) реализовано отключение (сброс напряжения в цепи управления) пускателя при достижении конечного (открытого или закрытого) положения задвижки. Если это аппаратно не реализовано, то необходимо отслеживать отключение привода при достижении конечного положения механизмом введением дополнительных состояний.

2. Ситуацию "Отключение привода по внешней причине" (например, переход из состояния 103 в состояние 0 с приоритетом 3 можно диагностировать по пропаданию сигнала "Привод включен"). Однако данный автомат применяется реально, и в процессе наладки приходится подстраховаться от одной неприятной ситуации. Создание схемы управления тоже является процессом творческим (хотя и более строгим по сравнению с алгоритмизацией программных функций) и созданная схема обладает определенными характеристиками, влияющими на управление. Имеются в виду временные характеристики срабатывания реле, на которых построена схема. Чем больше количество реле в составе цепи, тем позже сработает контакт, сигнализируя о каком-то событии.

Рис. 7. Схема связей главного технологического автомата

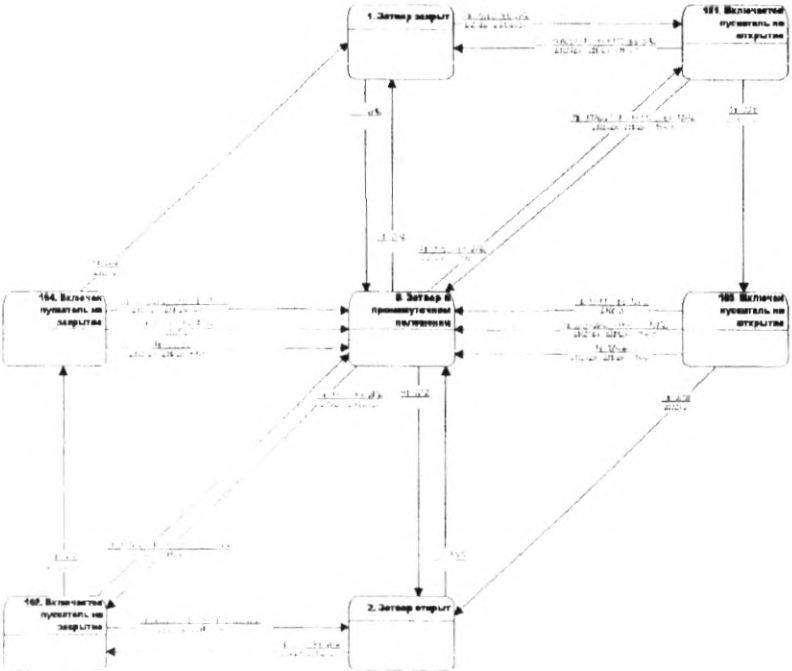


Рис. 8. Граф переходов главного технологического автомата.

Сигналы, поступающие в контроллер, являются результатом замыкания/размыкания контактов определенных реле схемы. Часто это - промежуточные реле, применяемые только для передачи сигналов в контроллер. Получается, что сигнал от конечного выключателя (допустим, "Затвор закрыт"), принимаемый контроллером, поступает не напрямую (непосредственный сигнал размыкает цепь питания, аппаратно отключая привод). Таким образом, вероятна ситуация, когда сигнал от конечного выключателя "придет" в контроллер позже того момента, когда пропадет сигнал о включенном состоянии привода.

Ситуация эта чревата тем, что получается неверная трактовка реакции привода. Система просигнализирует "Привод отключился по внешней причине", что является ненормальной ситуацией при

управлении. Однако указанное отключение по существу является нормальной, штатной работой схемы управления. Если бы это была только информационная "оплошность" системы, то особых проблем не было бы, но здесь иная ситуация – практически всегда при ненормальной работе оборудования (а именно так система будет "рассматривать" данную ситуацию) последует соответствующая реакция программы по выходу из создавшегося состояния.

Именно для логически правильного контроля работы оборудования и были введены дополнительные входные сигналы "Пускатель на открытие (закрытие) отключен по внешней причине (проверено временем)". Реально хватает 100 мс.

Таким образом, только если существует полная уверенность в быстродействии схемы, ситуацию "Отключение привода по внешней причине" можно диагностировать просто по пропаданию сигнала "Привод включен".

3. Входной сигнал "Включено питание привода" может объединять по схеме «И» все нормальные значения параметров привода и механизма, которые разрешают работу привода, включая, например, еще и отсутствие заклинивания. Обычно схемой предусматривается аппаратное отключение, или запрет включения привода при отсутствии необходимых условий. При этом главное предусмотреть в будущем технологическом автомате, который будет указывать приводу, когда ему управляться, проверку этих условий перед выдачей команды.

4. **Некоторые признаки только записываются в автомате.** Для них принято следующее правило – **необходимо предусмотреть безусловное стирание признака при каждом новом вызове автомата.** При разработке схемы автомата это подразумевается, но в программной реализации оно должно обязательно присутствовать. Таким образом, период существования каждого из таких признаков равен одному скану вызова автомата. Этот признак должен записываться и стираться только в данном автомате (Приложение 1).

4.7. Таблица символов автомата управления нереверсивным приводом

Автомат потенциального управления нереверсивным приводом является подпрограммой и оперирует формальными параметрами (табл. 4.2).

Таблица 4.2.

Формальные параметры автомата потенциального управления нереверсивным приводом

ФОРМАЛЬНЫЕ ПАРАМЕТРЫ АВТОМАТА УПРАВЛЕНИЯ НАСОСОМ (АСРх)		
YACPx	VB262	Номер состояния
		ВХОДНЫЕ ДАННЫЕ
X0P	V264.0	Включено питание привода
X1P	V264.1	Привод включен
KonP	V264.2	Команда "Включить"
KoffP	V264.3	Команда "Отключить"
TACP1	V264.4	Окончилось время контроля включения/отключения пускателя
		признаки ВЫХОДНЫХ ПРОЦЕДУР
zZ1P	V268.0	Запись сигнала "Управление работой"
zNZ1P	V268.1	Стирание сигнала "Управление работой"
Zstacp1	V268.2	Начать контроль времени включения/отключения пускателя
zM1P	V268.3	Запись признака "Нет включения пускателя за контрольное время"
zM2P	V268.4	Запись признака "Нет отключения пускателя за контрольное время"
zM3P	V268.5	Запись признака "Отключение пускателя по внешним причинам"

Контроллер S7-200 в большинстве случаев позволяет не экономить при распределении данных по адресам. Поэтому для унификации предлагается в любом автомате-подпрограмме выделять по четыре байта на входные и выходные переменные и по байту на номер состояния.

4.8. Таблица символов автомата управления реверсивным приводом

Табл. 4.3 содержит пример формальных параметров автомата для рассматриваемого привода.

Таблица 4.3.

Формальные параметры автомата потенциального управления задвижкой

ФОРМАЛЬНЫЕ ПАРАМЕТРЫ АВТОМАТА УПРАВЛЕНИЯ ЗАДВИЖКОЙ (ACSx)		
YACSx	VB250	Номер состояния
		ВХОДНЫЕ ДАННЫЕ
X0S	V252.0	Включено питание привода
X1S	V252.1	Задвижка открывается
X2S	V252.2	Задвижка закрывается
X3S	V252.3	Задвижка открыта
X4S	V252.4	Задвижка закрыта
KopS	V252.5	Команда "Открыть"
KclS	V252.6	Команда "Закрыть"
KoffS	V252.7	Команда "Остановить"
TACS1	V253.0	Окончилось время контроля включения/отключения пускателя
TACS2	V253.1	Окончилось время контроля открытия/закрытия задвижки
noneX1S_time	V253.2	Пускатель на открытие отключен по внешней причине (проверено временем)
noneX2S_time	V253.3	Пускатель на закрытие отключен по внешней причине (проверено временем)
		Признаки ВЫХОДНЫХ ПРОЦЕДУР
zZ1S	V256.0	Запись признака "Управление открытием"
zNZ1S	V256.1	Стирание признака "Управление открытием"
zZ2S	V256.2	Запись признака "Управление закрытием"
zNZ2S	V256.3	Стирание признака "Управление закрытием"
zstacs1	V256.4	Начать контроль времени включения/отключения пускателя

Окончание табл. 4.3.

zstacs2	V256.5	Начать контроль времени открытия/закрытия задвижки
zM1S	V256.6	Запись признака "Нет открытия задвижки за контрольное время"
zM2S	V256.7	Запись признака "Нет закрытия задвижки за контрольное время"
zM3S	V257.0	Запись признака "Нет включения пускателя на открытие за контрольное время "
zM4S	V257.1	Запись признака "Нет включения пускателя на закрытие за контрольное время "
zM5S **(не используется)**	V257.2	Запись признака "Нет отключения пускателя на открытие за контрольное время "
zM6S **(не используется)**	V257.3	Запись признака "Нет отключения пускателя на закрытие за контрольное время "
zM7S	V257.4	Запись признака "Отключение пускателя на открытие по внешним причинам"
zM8S	V257.5	Запись признака "Отключение пускателя на закрытие по внешним причинам"
zMopS	V257.6	Запись признака "Ненормальное открытие задвижки"
zMclS	V257.7	Запись признака "Ненормальное закрытие задвижки"

4.9. Таблица символов параметров, формируемых при вызове автомата потенциального управления нереверсивным приводом

Табл. 4.4 содержит пример символов фактических параметров для рассматриваемого привода.

Таблица 4.4.

ФАКТИЧЕСКИЕ ПАРАМЕТРЫ ПРИ ВЫЗОВЕ ACP_WP ДЛЯ УПРАВЛЕНИЯ ПРИВОДОМ ПРОМЫВНОГО НАСОСА (кроме фактических входов/выходов контроллера и таймеров)		
YACP_WP	VB32	Номер состояния
		ПАРАМЕТРЫ, ПЕРЕДАВАЕМЫЕ В АВТОМАТ УПРАВЛЕНИЯ (кроме фактических входов контроллера) Необходимые фактические входы контроллера также передаются в автомат
KonP_WP	V33.0	Команда "Включить"
KoffP_WP	V33.1	Команда "Отключить"
		КОНКРЕТНЫЕ ТАЙМЕРЫ, ИСПОЛЬЗУЕМЫЕ В АВТОМАТЕ (передаются при вызове автомата)
TACP1_WP	T121	Окончилось время контроля включения/отключения пускателя
		ПАРАМЕТРЫ, ФОРМИРУЕМЫЕ ПО ПРИЗНАКАМ ВЫХОДНЫХ ПРОЦЕДУР (С ИСПОЛЬЗОВАНИЕМ ИНСТРУКЦИЙ LD...=) (кроме фактических выходов контроллера и таймеров).Фактические выходы контроллера также формируются по признакам (с использованием инструкций LD...S...R). Таймеры также запускаются по признакам (с использованием инструкций LDN...TON).

Окончание табл. 4.4

Abnormal_WP_B0	VB34	Признаки ненормальной работы привода промывного насоса (нулевое слово)
M1P_WP	V34.0	Нет включения пускателя за контрольное время
M2P_WP	V34.1	Нет отключения пускателя за контрольное время
M3P_WP	V34.2	Отключение пускателя по внешним причинам
		РЕЗУЛЬТАТЫ ВЫЧИСЛЕНИЯ ПАРАМЕТРОВ-ФУНКЦИЙ, ВЫЗЫВАЕМЫЕ НЕПОСРЕДСТВЕННО В ТЕХНОЛОГИЧЕСКОМ АВТОМАТЕ
ConP_WP	V35.0	Есть все условия для автоматического пуска промывного насоса (DDI_ReadyWP - Насосный агрегат к пуску готов)
MNConP_WP	V35.1	Нет условий для автоматического пуска привода промывного насоса

Базовой для проекта является таблица входов/выходов контроллера. Если в автомате (непосредственном или подпрограмме) используется переменная, описанная в таблице входов/выходов, то, естественно, ее не надо описывать еще раз.

Эта и другие таблицы являются Excel-аналогом реальной **Symbol Table** среды разработки **STEP7-MicroWin32**. Поэтому описываемые здесь параметры (они же программные переменные со своими уникальными адресами) должны быть уникальны и не повторяться в разных таблицах (проверка уникальности обозначения и адреса производится автоматически в среде разработки **STEP7-MicroWin32**).

Таблица фактических параметров, используемых при вызове автомата управления, должна содержать логически определенный для рассматриваемого привода набор данных – все фактические

параметры (кроме входов/выходов), которые необходимы при управлении. Это не только данные, используемые в рассмотренном выше автомате управления, но и команды, результаты проверки условий для пуска/останова, признаки сообщений об отсутствии этих условий и т.д. Все эти данные формируются /используются в технологических автоматах, которых может быть несколько. Однако все они относятся к конкретному приводу, и поэтому их логично описывать в одной таблице.

Для минимизации размера кода программы при дальнейших операциях поддержки обмена по сети введена обобщенная переменная Abnormal_WP_B0 с адресом, перекрывающим адреса признаков ненормальной работы привода.

Таймеры, используемые при контроле времени, представлены в таблице как булевы переменные, но в программе, естественно, будут использованы и их текущие численные значения.

4.10. Таблица символов параметров, формируемых при вызове автомата потенциального управления реверсивным приводом

Табл. 4.5 содержит символы фактических параметров для рассматриваемого привода.

Таблица 4.5.

Пример **таблицы фактических параметров** при вызове автомата потенциального управления реверсивным приводом

ФАКТИЧЕСКИЕ ПАРАМЕТРЫ ПРИ ВЫЗОВЕ ACS_FW ДЛЯ УПРАВЛЕНИЯ НАПОРНОЙ ЗАДВИЖКОЙ (кроме фактических входов/выходов контроллера и таймеров)		
YACS_FW	VB18	Номер состояния
		ПАРАМЕТРЫ, ПЕРЕДАВАЕМЫЕ В АВТОМАТ УПРАВЛЕНИЯ (кроме фактических входов контроллера). Необходимые фактические входы контроллера также передаются в автомат.

Продолжение табл. 4.5.

KopS_FW	V19.0	Команда "Открыть"
KclS_FW	V19.1	Команда "Закрыть"
KoffS_FW	V19.2	Команда "Остановить"
		КОНКРЕТНЫЕ ТАЙМЕРЫ, ИСПОЛЬЗУЕМЫЕ В АВТОМАТЕ (передаются при вызове автомата).
TACS1_FW	T111	Окончилось время контроля включения/отключения пускателя
TACS2_FW	T112	Окончилось время контроля открытия/закрытия задвижки
noneX1S_time_FW	T113	Пускатель на открытие отключен по внешней причине (проверено временем)
noneX2S_time_FW	T114	Пускатель на закрытие отключен по внешней причине (проверено временем)
		ПАРАМЕТРЫ, ФОРМИРУЕМЫЕ ПО ПРИЗНАКАМ ВЫХОДНЫХ ПРОЦЕДУР (С ИСПОЛЬЗОВАНИЕМ ИНСТРУКЦИЙ LD...=) (кроме фактических выходов контроллера и таймеров). Фактические выходы контроллера также формируются по признакам (с использованием инструкций LD...S...R). Таймеры также запускаются по признакам (с использованием инструкций LDN...TON).
Abnormal_FW_B0	VB20	Признаки ненормальной работы напорной задвижки (0-й байт)
M1S_FW	V20.0	Нет открытия задвижки за контрольное время
M2S_FW	V20.1	Нет закрытия задвижки за контрольное время

Окончание табл. 4.5.

M3S_FW	V20.2	Нет включения пускателя на открытие за контрольное время
M4S_FW	V20.3	Нет включения пускателя на закрытие за контрольное время
MNCopS_FW	V20.4	Нет условий для автоматического открытия напорной задвижки
MNCclS_FW	V20.5	Нет условий для автоматического закрытия напорной задвижки
M7S_FW	V20.6	Отключение пускателя на открытие по внешним причинам
M8S_FW	V20.7	Отключение пускателя на закрытие по внешним причинам
MopS_FW	V21.0	Ненормальное открытие задвижки
MclS_FW	V21.1	Ненормальное закрытие задвижки
		РЕЗУЛЬТАТЫ ВЫЧИСЛЕНИЯ ПАРАМЕТРОВ-ФУНКЦИЙ, ВЫЗЫВАЕМЫЕ НЕПОСРЕДСТВЕННО В ТЕХНОЛОГИЧЕСКОМ АВТОМАТЕ
CopS_FW	CopS_FW	Есть все условия для автоматического открытия напорной задвижки (X0 - Включено питание привода задвижки; !X2 – Не включен привод задвижки на закрытие; !X5 - Задвижка не заклинена; !HTD - Нет превы-шения температуры двигателя напорной задвижки)
CclS_FW	V22.1	Есть все условия для автоматического закрытия напорной задвижки (X0 - Включено питание привода задвижки; !X1 – Не включен привод задвижки на открытие; !X5 - Задвижка не заклинена; !HTD - Нет превышения температуры двигателя напорной задвижки)

Адреса признаков MNCopS_FW, MNCclS_FW ("Нет условий для автоматического открытия/закрытия") можно выбрать и вне диапазона признаков ненормальной работы задвижки. Однако поскольку их скорее всего потребуется передавать по сети, то лучше эти адреса включить в байт признаков ненормальной работы (Abnormal_FW_B0).

ГЛАВА 5
ТЕХНОЛОГИЧЕСКИЕ АЛГОРИТМЫ УПРАВЛЕНИЯ

5.1. Синтез технологического автомата управления насосным агрегатом

Технологические алгоритмы являются, как правило, более сложными по сравнению с базовыми алгоритмами.

Техническое задание обычно описывает необходимые действия системы управления при нормальном ходе процесса. Разработчику алгоритмов придется "додумать" реакцию системы при ненормальной ситуации.

5.2. Типовое техническое задание

Ниже приводится пример фрагмента технического задания – задания на управление насосным агрегатом.

4.7. Режимы работы шкафа автоматического управления насосным агрегатом.

4.7.1. Режим "АВТОМАТИЧЕСКИЙ".

В автоматическом режиме осуществляется:

– штатный пуск насосного агрегата, как по команде с верхнего уровня от мастер - контроллера промывки блока КО-1, так и по командам со шкафа;

– штатная остановка насосного агрегата, как по команде с верхнего уровня от мастер - контроллера, так и по командам со шкафа;

– аварийная остановка насосного агрегата, как по команде с верхнего уровня от мастер - контроллера, так и по командам со шкафа с последующим закрытием напорной задвижки;

– закрытие напорной задвижки после аварийной остановки или при поступлении от подстанции во время промывки сигнала "АГРЕГАТ АВАРИЙНО ОТКЛЮЧЕН";

– индикация состояния элементов автоматики, положения задвижек, сигналов с подстанции и передача их на верхний уровень

Дополнения к заданию:

- штатный пуск (включение насоса, затем открытие напорной задвижки);

- штатный останов (закрытие напорной задвижки, затем останов насоса);

- перед включением насоса необходима предпусковая сигнализация;
- включать насос можно только при закрытой задвижке.

5.3. Нормальный ход процесса

На рис. 9, 10 представлены схема связей и граф переходов автомата управления насосным агрегатом при условии нормальной работы оборудования.

Это - относительно простой алгоритм (за исключением не вполне очевидной логики обработки аварийных сигналов и команд на останов насоса при открытии/закрытии напорной задвижки). При этом **следует обратить внимание на одно обстоятельство.**

Вспомним, как выполняются программа в контроллере S7-200, а именно **"CPU выполняет программу последовательно, начиная с первой команды и продолжая до ее конечной команды".** Этот процесс является только одной из частей цикла сканирования. Остальные части наверное даже более необходимы, особенно **считывание входов.** Таким образом, ни в коем случае **нельзя зацикливать программу логически,** поскольку она просто не будет получать текущую информацию об объекте управления. Следовательно, алгоритм должен учитывать эту особенность выполнения программы контроллера.

Алгоритм необходимо создавать в виде, наиболее близком к реализации. Гораздо проще разобраться в логике, изображенной в виде диаграммы на бумаге, чем изучать программный код.

Алгоритм, созданный на принципах **SWITCH-технологии, и программа, созданная на основе этого алгоритма,** полностью отвечают вышеизложенным требованиям.

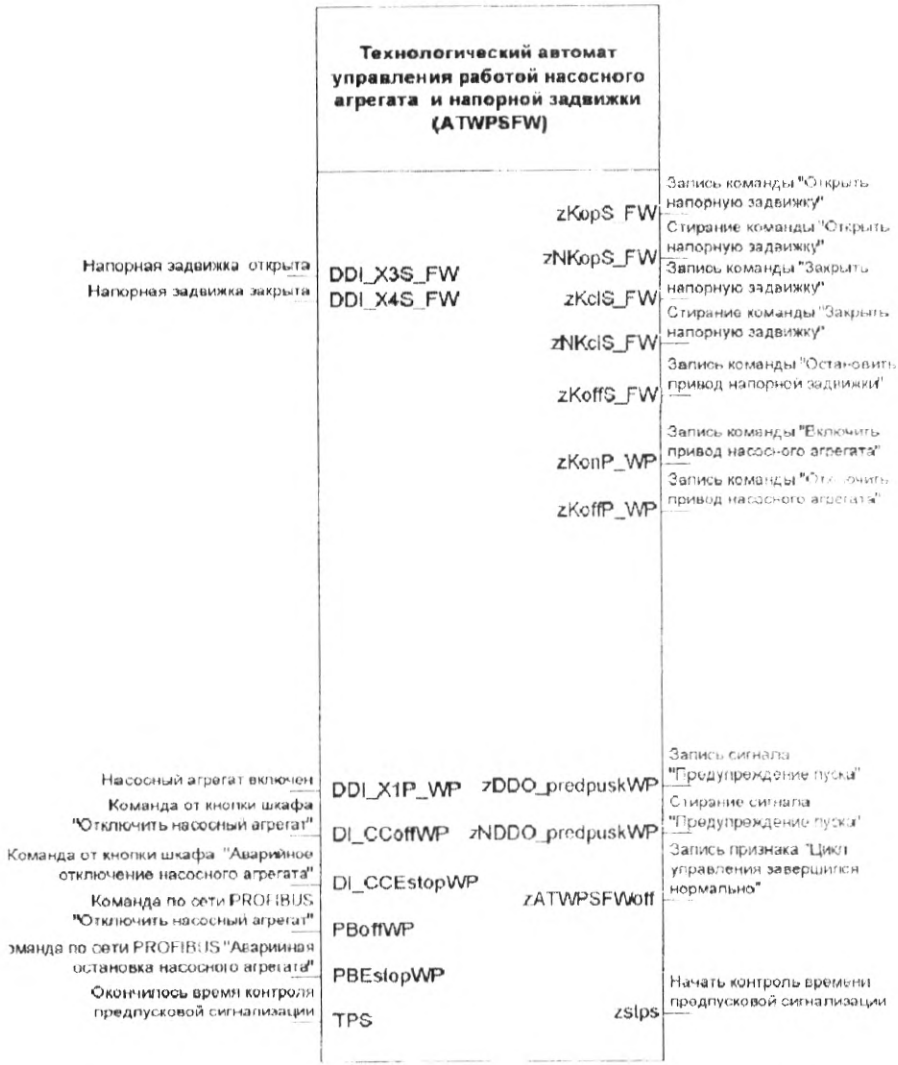


Рис. 9. Схема связей технологического автомата управления работой насосного агрегата и напорной задвижкой

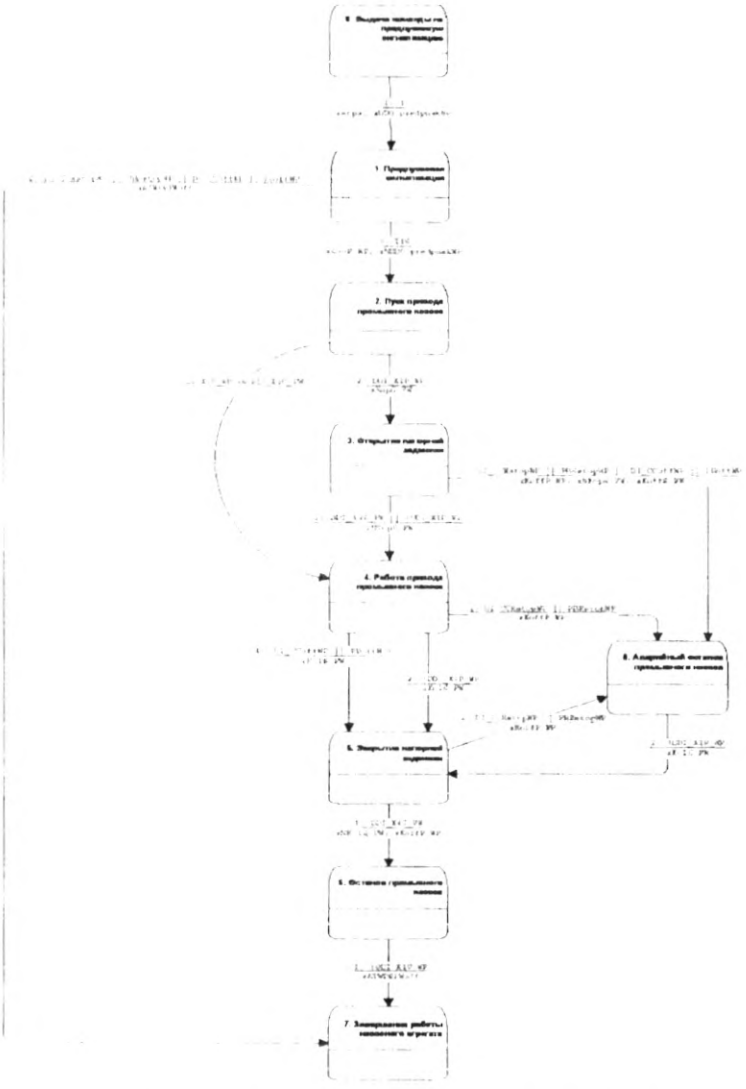


Рис. 10. Граф переходов автомата управления работой насосного агрегата и напорной задвижкой.

5.4. Дополнение функций автомата

На рис. 11 и 12 представлены схема связей и граф переходов рассматриваемого автомата, дополненный событиями, связанными с неготовностью или ненормальной работой оборудования.

Пояснения к этим диаграммам.

Опыт наладки показал, что почти все из того, что дополнительно контролировалось, было использовано.

При этом необходимо обратить внимание на следующее:

· рекомендуется как можно более полно описывать логику формирования входной информации. В данном случае – условия готовности привода к управлению. В дальнейшем это помогает при программировании – нет необходимости разбираться почему и что контролировать;

· под готовностью подразумевается отсутствие любых условий, препятствующих нормальной работе объекта управления;

· при отсутствии готовности оборудования рекомендуется формировать признаки неготовности. Они могут использоваться для информирования оператора или в других алгоритмах;

· применение результатов контроля готовности оборудования производится в момент непосредственно перед выдачей команды управления. Таким образом фиксируется событие "хотели управлять, но в данный момент это невозможно". Готовность обычно контролируется все время, естественно, в другом месте программы. Если использовать результаты такого контроля в любой момент времени, то может возникнуть ситуация, когда неготовность оборудования, которым уже управляли или не будут управлять, не позволит выполнить алгоритм в целом. При этом будет невозможно сформировать признак неготовности именно в требуемый момент времени;

· также как и в базовом автомате, технологический автомат обеспечивает безусловное стирание перед вызовом тех признаков которые только записываются этим автоматом (например, признаки неготовности привода к управлению).

5.5. Головной технологический автомат

Опишем когда должен запускаться вышеприведенный технологический автомат. Делается это в одном из состояний «главного» (головного) автомата. В любом проекте всегда

существуют какие-то основные условия для управления от контроллера, например, автоматический режим управления. При этом только наличие всех основных условий должно разрешать действия контроллера, а отсутствие любого из них должно немедленно прекратить эти действия и/или совершить необходимые действия по выходу из режима управления от контроллера. В системе также имеются команды, инициирующие работу логической части программы контроллера. Они могут поступать со входов контроллера или по сети.

Данный случай – не исключение. На рис. 13 и 14 приведены схема связей и граф переходов «главного» технологического автомата для управления насосным агрегатом.

Главный автомат ожидает появления всех условий для управления от контроллера (режим "Автоматический") и команды, инициирующей начало отработки цикла управления.

В проекте предусматривается запрет на прохождение сигналов от контроллера в схему при отсутствии режима "Автоматический". Поэтому в автомате применяется процедура zStopPS, осуществляющая "подхват" запрета – программное снятие сигналов с выходов контроллера при отсутствии условий работы от контроллера. Для простоты указанный запрет обеспечивается не только при отсутствии режима "Автоматический", но и всех других необходимых условий.

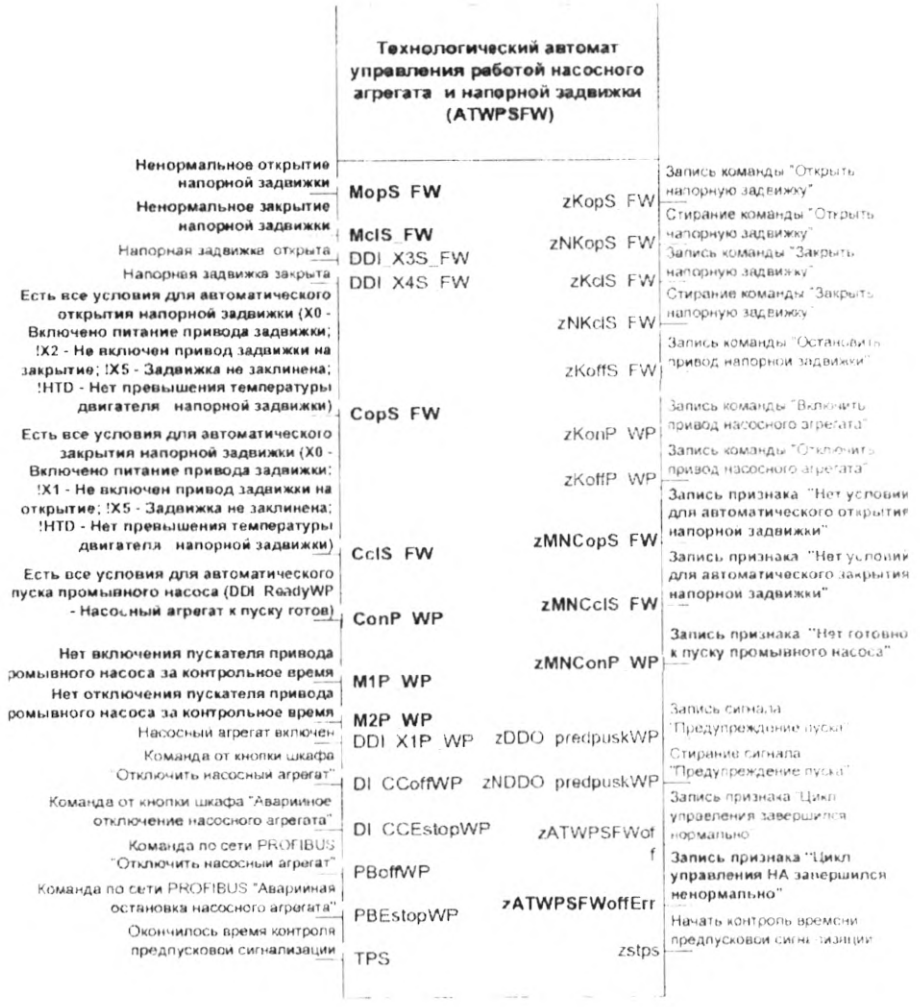


Рис. 11. Схема связей технологического автомата управления работой насосного агрегата и напорной задвижки.

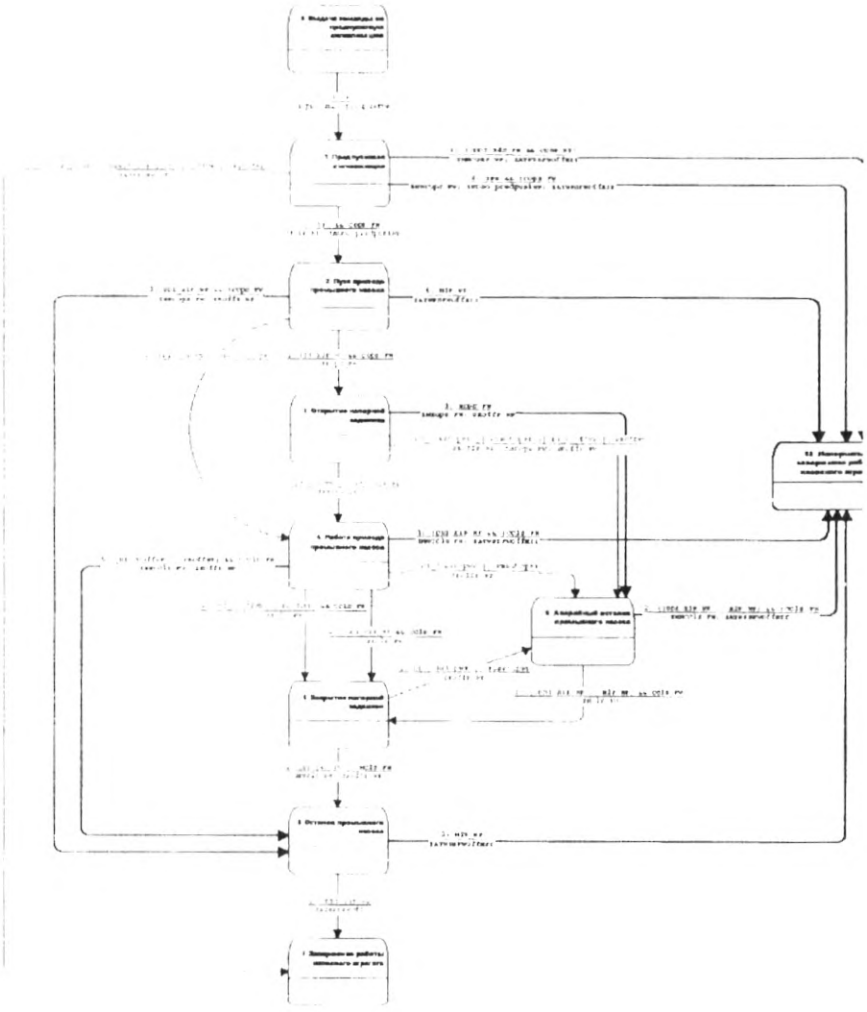


Рис. 12. Граф переходов технологического автомата управления работой насосного агрегата и напорной задвижки.

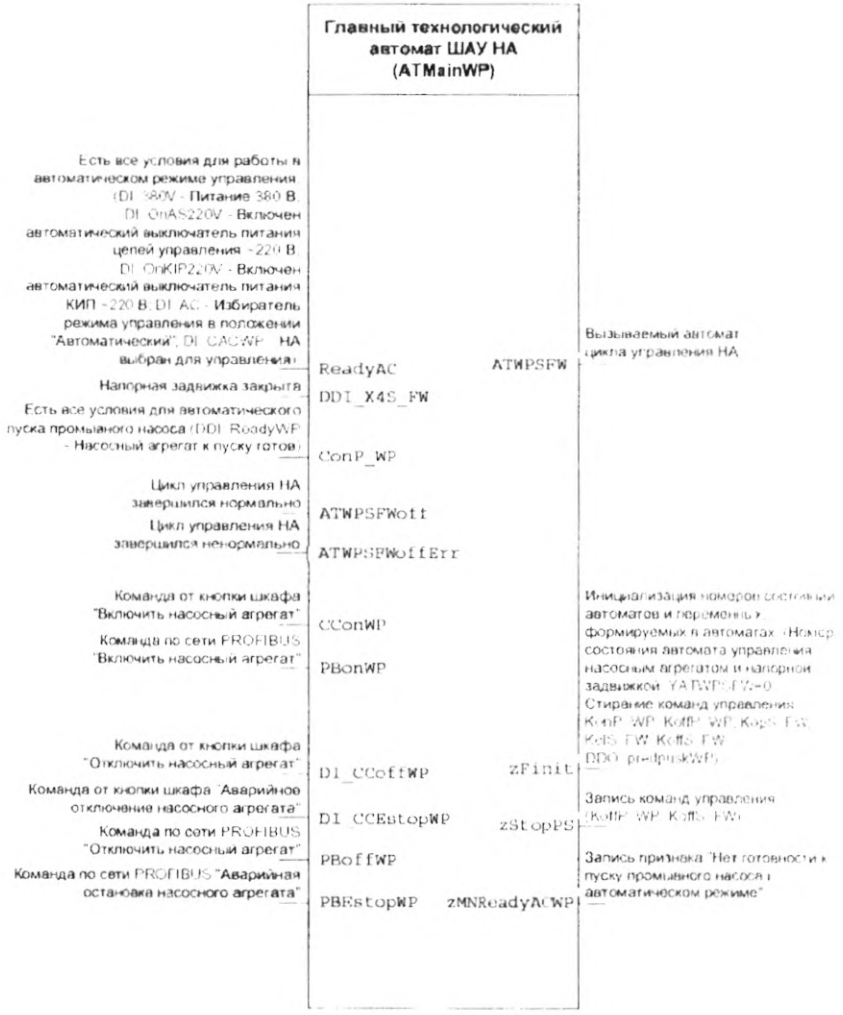


Рис. 13. Схема главного технологического автомата.

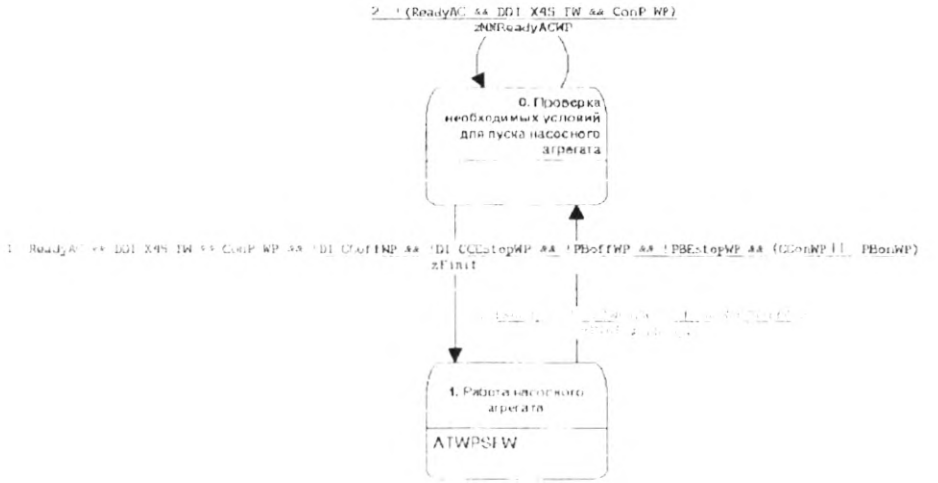


Рис. 14. Граф переходов главного технологического автомата.

5.6. Таблицы символов технологических автоматов

Табл. 4.6 содержит фактические параметры для рассматриваемого технологического автомата.

В табл. 4.6 мало данных. Однако вся остальная информация есть в таблицах вызова базовых автоматов. Технологические автоматы практически всегда используют и формируют данные, уже описанные в таблице входных/выходных сигналов контроллера и в таблицах базовых автоматов. Все остальные данные (чаще всего выходные), необходимы только для реализации логики или для последующего контроля реализации конкретного алгоритма.

Таблица 4.6.

Таблица фактических параметров технологического автомата управления циклом работы насосного агрегата

ФАКТИЧЕСКИЕ ПАРАМЕТРЫ ПРИ ВЫЗОВЕ ATWPSFW для управления циклом работы НА (кроме фактических входов/выходов контроллера, таймеров и других переменных, считываемых непосредственно в автомате)		
YATWPSFW	VB16	Номер состояния
		РЕЗУЛЬТАТЫ ВЫЧИСЛЕНИЯ ПАРАМЕТРОВ-ФУНКЦИЙ, передаваемые в технологический автомат при вызове (фактические входы контроллера и другие переменные считываются непосредственно в автомате).
		КОНКРЕТНЫЕ ТАЙМЕРЫ, используемые непосредственно в технологическом автомате
TPS	T103	Окончилось время контроля предпусковой сигнализации
		ПАРАМЕТРЫ, ФОРМИРУЕМЫЕ НЕПОСРЕДСТВЕННО В АВТОМАТЕ (с использованием инструкций LD...=) (кроме команд управления приводами и признаков сообщений "Нет условий для открытия /закрытия..."). Команды управления приводами формируются также НЕПОСРЕД-СТВЕННО В АВТОМАТЕ (с использованием инструкций LD...S...R). Признаки сообщений "Нет условий для открытия/закрытия..." формируются также НЕПОСРЕДСТВЕННО В АВТОМАТЕ (с использованием инструкций LD...=). Таймеры запускаются по признакам выходных процедур ПОСЛЕ ВЫЗОВА АВТОМАТА (с использованием инструкций LDN...TON).
ATWPSFWoff	V17.0	Цикл управления НА завершился нормально
ATWPSFWoff Err	V17.1	Цикл управления НА завершился ненормально
Stps	V17.2	Начать контроль времени предпусковой сигнализации
		Признаки вызова ПОДАВТОМАТОВ

Табл. 4.7 содержит фактические параметры для главного технологического автомата.

Таблица 4.7.

Таблица фактических параметров «главного» технологического автомата

ФАКТИЧЕСКИЕ ПАРАМЕТРЫ ПРИ ВЫЗОВЕ ATMainWP для управления работой насосного агрегата (кроме фактических входов/выходов контроллера, таймеров и других переменных, считываемых непосредственно в автомате)		
YATMainWP	VB14	Номер состояния
		РЕЗУЛЬТАТЫ ВЫЧИСЛЕНИЯ ПАРАМЕТРОВ-ФУНКЦИЙ, ПЕРЕДАВАЕМЫЕ В ТЕХНОЛОГИЧЕСКИЙ АВТОМАТ при вызове (фактические входы контроллера и другие переменные считываются непосредственно в автомате).
ReadyAC	V15.0	Есть все условия для работы в автоматическом режиме управления
		КОНКРЕТНЫЕ ТАЙМЕРЫ, ИСПОЛЬЗУЕМЫЕ НЕПОСРЕДСТВЕННО В ТЕХНОЛОГИЧЕСКОМ АВТОМАТЕ
		ПАРАМЕТРЫ, ФОРМИРУЕМЫЕ НЕПОСРЕДСТВЕННО В АВТОМАТЕ(С ИСПОЛЬЗОВАНИЕМ ИНСТРУКЦИЙ LD...=) (кроме команд управления приводами и признаков сообщений "Нет условий для открытия/закрытия..."). Команды управления приводами формируются также НЕПОСРЕДСТВЕННО В АВТОМАТЕ (с использованием инструкций LD...S...R).

Окончание табл. 4.7

		Признаки сообщений "Нет условий для открытия/закрытия..." формируются также НЕПОСРЕДСТВЕННО В АВТОМАТЕ (с использованием инструкций LD...=). Таймеры запускаются по признакам выходных процедур ПОСЛЕ ВЫЗОВА АВТОМАТА (с использованием инструкций LDN...TON).
Finit	V15.1	Инициализация номеров состояний автоматов и переменных, формируемых в автоматах
MNReadyACWP	V15.2	Нет готовности к пуску промывного насоса в автоматическом режиме
StopPS	V15.3	Запись команд управления (KoffP_WP, KoffS_FW)
		Признаки вызова ПОДАВТОМАТОВ
callATWPSFW	V15.4	Вызываемый автомат цикла управления насосным агрегатом

ГЛАВА 6
ПРОГРАММИРОВАНИЕ ОСНОВНЫХ ФУНКЦИЙ ПРОГРАММЫ ПОЛЬЗОВАТЕЛЯ

6.1. Основные процедуры программы пользователя

Функции локальной системы управления реализуются следующими процедурами программы:

1. Инициализация значений переменных (только в первом цикле программы).

2. Проверка на достоверность кода входного сигнала аналого-цифрового преобразователя (АЦП). Масштабирование значений параметров, представленных аналоговыми сигналами.

3. Контроль значений параметров, представленных аналоговыми и дискретными сигналами.

4. Формирование признаков – результатов вычисления функций, необходимых для реализации технологического автомата.

5. Технологический автомат управления (основная функция – формирование команд управления приводами). Как правило, автомат этого типа выполняется при режиме управления от контроллера.

6. Контроль состояния и управление приводами (основная функция – формирование выходных сигналов контроллера для управления приводами).

7. Процедуры поддержки индикатора TD200.

8. Процедуры поддержки обмена по сети (как правило, ProfiBus). Основная функция – формирование выходного буфера обмена для передачи по сети.

Для обеспечения качественного тестирования функций управления рекомендуется применять программные имитаторы работы приводов (основная функция – формирование информации, программно имитирующей значения реальных входных сигналов из схемы
управления приводом). Выполнение данных процедур производится перед формированием указанных выше признаков.

Процедуры 1 – 3, 7, 8 являются достаточно простыми и необходимы для контроля за ходом технологического процесса.

Процедуры 4 – 6 – более сложные процессы по управлению и контролю работы оборудования.

6.2. Описание основных процедур

Процедура 1. Инициализация значений переменных

Выполняется только в первом цикле программы. При этом используется служебный бит SM0.1, значение которого устанавливается в первом цикле обработки программы. В отличие от инициализации, проводимой с применением таблицы **Data Block**, данная процедура выполняется при каждом новом запуске программы.

Инициализации подлежат:

– признаки результатов контроля аналоговых и дискретных сигналов (стираются);

– команды управления приводами (стираются);

– номера состояний всех технологических (ATx, ARx) и управляющих (ACx) автоматов.

Эти номера приводятся к начальным, например, YATx=0; YACx=0;

–команды управления технологическими операциями (и/или приводами), получаемые по сети (стираются);

– номера состояний всех имитационных (AI) автоматов (приводятся к начальным);

– номера имитируемых ситуаций (Hx) (приводятся к нормальным).

Реализация инициализации достаточна проста. Для этих целей рекомендуется использовать служебные биты SM0.0 для собственно инициализации и SM0.1 при вызове процедуры.

Процедура 2. Проверка на достоверность кода входного сигнала АЦП. Масштабирование значений параметров, представленных аналоговыми сигналами

Здесь не должно быть сложностей логического плана (хотя обычно приходится повозиться с наладкой ввода аналоговых сигналов, но, это, по большей мере, техническая проблема).

Процедура 3. Контроль значений параметров, представленных аналоговыми и дискретными сигналами

Это достаточно простая процедура с точки зрения логики.

Процедура 4. Формирование признаков – результатов вычисления функций, необходимых для выполнения технологического автомата

Этой процедурой начинается часть программы, непосредственно относящаяся и влияющая на управление. Приведенные в качестве примеров технологические автоматы используют непосредственные данные (описанные в различных таблицах символов). При программировании будет использовано только то, что изображено на графе переходов автомата. Поэтому необходимо подготовить (вычислить) где-то в отдельной подпрограмме те данные, которые являются результатом логической обработки, например, признаки готовности оборудования для автомата управления циклом работы насосного агрегата.

Вся логика формирования таких данных полностью описана в соответствующих автоматах и является достаточно простой. Ее реализация не должна вызывать особых трудностей при программировании.

Может возникнуть вопрос — зачем нужна отдельная подпрограмма? Ведь можно подготавливать данные в процессе вызова автомата, передавая их в качестве параметров. Ответ прост – чаще всего эти данные используются одновременно в разных технологических автоматах, и поэтому незачем вычислять их заново.

Процедура 5. Технологический автомат управления

Эта и следующая процедуры предполагают **формальное программирование алгоритмов.** Возможность такого подхода предоставляется самой технологией – **SWITCH-технологией®.**Ведь при ее использовании алгоритм реализуется весьма просто: "вызвал проверил – если есть необходимые условия, сделал что надо (включая переход) – вышел". Такие однородные действия позволяют не только выполнять программирование формально, но и **автоматически создавать код программы по графам переходов автоматов** (Приложение 4).

Программирование (вручную или автоматически) проще всего выполнять с использованием языка **STL** среды **STEP7-MicroWin32**. В языке **STL** нет оператора, аналогичного switch языка C, поэтому придется воспользоваться аналогом оператора goto – инструкцией JMP. Структура программы при применяемом подходе фиксирована.

Код программы, реализующей произвольный автомат, состоит из нескольких частей:

· обнуление (сброс) команд, признаков и т.п., используемых в автомате, но не имеющих обнуления по условию;

· определение текущего состояния;

· проверка условий и действий в состоянии 0;

· …

· проверка условий и действий в состоянии N;

· обнуление (сброс) признаков вызова вложенных автоматов (если они используются);

· вызов вложенных автоматов, если текущее состояние равно 0;

· …

· вызов вложенных автоматов, если текущее состояние равно N.

Используемые инструкции языка:

LD - **загрузка значения первого элемента (аргумента) логической функции;**

LDN - **загрузка обратного значения первого элемента (аргумента) логической функции;**

A - AND **для последующего аргумента;**

AN - NOT AND

O - OR

= - **присвоение результата логической функции;**

S - **установка значения переменной в** TRUE **(выполняется по условию);**

R - **сброс значения переменной в** FALSE **(выполняется по условию);**

LDB= - **загрузка и сравнение численных значений первых двух байтовых аргументов (вариации** - LDB>, LDB<, LDB>=, LDB<= **и т.д.) логической функции;**

JMP - **переход на метку (выполняется по условию);**

LBL - **метка;**

MOVB - **“передача” численного значения одного байтового**

аргумента другому (выполняется по условию);

TON **- одна из инструкций вызова таймера (выполняется по условию);**

На языке STL шаблон для реализации произвольного автомата имеет следующий вид.

Network 1

// Обнуление (сброс) команд, признаков и т.п., формируемых в

// автомате, но не имеющих обнуления по условию

//

LDN SM0.0

команда (признак) 1

...

команда (признак) n

Network 2

// Определение текущего состояния (аналог "case 0" на языке C)

//

LDB **номер состояния автомата,** 0

JMP 0

...

Network 3

// Определение текущего состояния (аналог "case 1" на языке C)

//

LDB **номер состояния автомата,** 1

JMP 1

Network 4

// Переход по умолчанию (необходим для выхода из автомата,

// если используется несуществующий номер состояния)

//

LD SM0.0

JMP 255

Network 5

// Проверка условий и действий в состоянии 0

//

LBL 0

Network 6

// Переход с приоритетом 1 из состояния 0 в состояние 1

//

```
LD      условие 1
A       условие 2
...
AN      условие k
MOVB    1, номер состояния автомата
=       признак выполнения какого-либо действия
JMP     255
Network 7
//  Переход с приоритетом 2 из состояния 0 в состояние 0
//
LD      условие 1
A       условие 2
JMP     255
Network 8
//  Переход по умолчанию
//
LD      SM0.0
JMP     255

Network 9
//  Проверка условий и действий в состоянии 1
//
LBL     1
Network 10
//  Переход с приоритетом 1 из состояния 1 в состояние 0
//
LDN     условие 1
O       условие 2
MOVB    0, номер состояния автомата
=       признак выполнения какого-либо действия
=       признак выполнения какого-либо действия
JMP     255
Network 11
LBL     255
Network 12
//  Обнуление (сброс) признаков вызова вложенных автоматов
//
LDN     SM0.0
```

= **признак вызова вложенного автомата**

Network 13

// Вызов вложенных автоматов, если текущее состояние первое

//

LDB= **номер состояния автомата,** 1

= **признак вызова вложенного автомата**

JMP 254

Network 14

LBL 254

Как следует из приведенного шаблона, структура кода строгая и позволяет реализовать произвольный автомат. Программирование выполняется последовательно – состояние за состоянием, условие за условием и т.д.

В итоге получается **код программы, ПОЛНОСТЬЮ соответствующий графу переходов автомата (алгоритму)!**

Для получения наилучшего эффекта **РЕКОМЕНДУЕТСЯ автоматизировать построение кода программы по графу переходов.** Объектная модель редактора Visio позволяет это сделать. Тогда при тестировании **не будет необходимости вообще заглядывать в текст программы.** Надо будет только исправить граф переходов автомата и сгенерировать новый код.

Примеры сгенерированного кода приведены в приложениях 5, 6.

Для завершения построения программы осталось проделать несложные операции над выходными переменными по сформированным в автомате признакам. Для того, чтобы унифицировать общую структуру программы рекомендуется производить вызов автомата в отдельной подпрограмме. Там же можно выполнить операции над выходными переменными, операции с таймерами и вызов вложенных автоматов.

На языке STL это выглядит так:

Network 1

// Собственно вызов автомата

//

LD SM0.0

CALL **подпрограмма автомата**

Network 2
LD **признак выполнения операции над выходной переменной**
S **выходная переменная,** 1
Network 3
LD **признак выполнения операции над выходной переменной**
R **выходная переменная,** 1
Network 4
LD **признак выполнения операции над выходной переменной**
= **выходная переменная**
Network 5
LDN **признак выполнения операции с таймером**
TON **таймер, уставка таймера**
Network 6
LD **признак вызова вложенного автомата**
CALL **программа, реализующая вложенный автомат**

По поводу операций с выходными данными: здесь действует следующее правило.

Если переменная записывается и стирается, то следует использовать инструкции S и R.

Если переменная только записывается, то применяется инструкция =. Переменная безусловно стирается в начале процедуры, реализующей автомат.

Процедура 6. Контроль состояния и управление приводами

Базовый автомат управления является подпрограммой, оперирующей формальными параметрами. Структура кода программы для этого автомата ничем не отличается от рассмотренной выше для технологического автомата. При этом, правда, скорее всего не будет вызова вложенных автоматов.

Рекомендуется отказаться от передачи параметров при вызове подпрограммы. Во-первых, понадобится вручную формировать список входных/выходных параметров. Во-вторых, в пакете **STEP7-MicroWin32** существует ограничение на общее количество входных/выходных параметров подпрограммы и подстраивать под

это ограничение логику автомата (количество входных/выходных данных) по меньшей мере странно. В-третьих, входные параметры являются по большей части результатами вычисления булевых формул и поэтому их надо вычислять и где-то хранить промежуточные результаты.

На языке STL вызов базового автомата выглядит следующим образом:

```
Network 1
//      Формирование формальных параметров
//
LD      конкретная непосредственная программная переменная
        формальный входной параметр
...
Network k
//      Формирование формальных параметров
//
LD      конкретная непосредственная программная переменная
        формальный входной параметр
Network m
//      Перезапись непосредственного номера состояния автомата в
//      формальный номер.
//      Собственно вызов автомата.
//      Перезапись формального номера состояния в
//      непосредственный номер.
//
LD      SM0.0
MOVB
CALL    подпрограмма автомата
MOVB    формальный номера состояния, непосредственный
номер
Network x
LD      признак выполнения операции над выходной
переменной
S       выходная переменная, 1
...
Network y
```

LD **признак выполнения операции над выходной переменной**
R **выходная переменная,** 1
Network z
LDN **признак выполнения операции с таймером**
TON **таймер, уставка таймера**

И здесь унифицированный подход к реализации логической части программы.

Процедура 7. Поддержка индикатора TD200

Рекомендации по этой и следующей процедуре могут показаться избыточными. Однако лучше заложить более основательный механизм и поддержать его также и на верхнем уровне в системе SCADA с тем, чтобы информация о ненормальной работе оборудования (чаще всего оно то там, то тут "сбоит", гарантированно доходила до оператора. То же самое можно сказать и о задании уставок и режимов – должна быть полная уверенность, что необходимые значения принимаются при передаче в любом направлении.

Процедуры поддержки обеспечивают:
- запись значения уставки, измененного в индикаторе TD200, в значение уставки контроллера (при установленном флаге изменения (бита редактирования) значения уставки);
- запись значения уставки контроллера в значение уставки, выводимой в индикатор TD200 (при установленном флаге изменения (бита редактирования) значения уставки);
- установку флагов разрешения вывода сообщений на индикатор TD200 в зависимости от состояния флага нажатия функциональной клавиши F1 (вывод сообщений с масштабированными значениями аналоговых параметров);
- сброс флага нажатия функциональной клавиши F4 (вывод сообщений со значениями уставок) по переднему фронту флага нажатия функциональной клавиши F1;
- установку флагов разрешения вывода сообщений на индикатор TD200 в зависимости от состояния флага нажатия функциональной клавиши F4 (вывод сообщений со значениями уставок);

- сброс флага нажатия функциональной клавиши F1 (вывод сообщений с масштабированными значениями аналоговых параметров) по переднему фронту флага нажатия функциональной клавиши F1.

Рекомендуется хранить все уставки в контроллерах Slave (практически всегда предполагается, что контроллеры будут управлять оборудованием в автономном режиме, не взирая на состояние связи с Master-контроллером или системой SCADA).

Поэтому при первой связи между контроллерами S7-300 (Master) и S7-200 (Slave) необходимо обеспечить передачу значений уставок из контроллера Slave (S7-200) в контроллер Master (S7-300).

Одновременно с этим необходимо учитывать, что значения уставок могут изменяться с помощью индикатора TD200. Это является приоритетной операцией по сравнению с изменением уставок на верхнем уровне посредством какой-либо системы SCADA. Значения реальных уставок контроллера считываются из области памяти, определенной при конфигурации индикатора TD200 (Embedded Data Value), при установленном бите редактирования уставки. Это бит устанавливается автоматически из индикатора TD200 при изменении значения уставки. При сброшенном бите редактирования производится обратная операция записи в Embedded Data Value значения реальной уставки контроллера.

Значение уставки, присылаемой из контроллера Master (S7-300), переписывается в реальную уставку контроллера при сброшенном бите редактирования уставки. Это обеспечивает приоритетную передачу уставок, измененных в индикаторе TD200, в контроллер Master (S7-300), и, соответственно, синхронизацию уставок верхнего уровня с уставками контроллера. При успешной передаче значений уставок контроллера в контроллер Master (S7-300), биты редактирования сбрасываются.

Для обеспечения вышеизложенного предлагается в первом скане программы контроллера (Main) устанавливать биты редактирования уставок, а при операции считывания из области памяти Embedded Data Value в реальную уставку контроллера учитывать то, что эта операция НЕ должна производиться в первом скане программы.

Процедура 8. Поддержка обмена по сети ProfiBus

Предлагается следующая организация связи.

Функции связи:

1. Основная – передача текущей информации о состоянии объектов управления от контроллера S7-200 (Slave) к контроллеру S7-300 (Master). Подавляющая часть этой информации – признаки ненормальной работы оборудования.

2. Передача команд управления от контроллера S7-300 (Master) к контроллеру S7-200 (Slave).

3. Передача значений уставок, измененных в контроллере S7-200 (изменения производятся посредством индикатора TD-200) от контроллера S7-200 (Slave) к контроллеру S7-300 (Master).

4. Передача значений уставок, измененных на верхнем уровне (изменения производятся посредством какой-либо системы SCADA) от контроллера S7-300 (Master) к контроллеру S7-200 (Slave).

Информация (принимаемая, посылаемая) при обмене по сети ProfiBus:

1. Масштабированные значения параметров, представленных аналоговыми сигналами.

2. Значения дискретных входов/выходов контроллера.

3. Информация, передаваемая о состоянии, передаваемая от контроллера S7-200 (Slave) к контроллеру S7-300 (Master) и возвращаемая обратно:

- признаки, формируемые по результатам первичного контроля входных аналоговых сигналов (признаки выхода значения входного кода за пределы достоверных и реальных);
- зафиксированные по результатам контроля ("защита временем") входные дискретные сигналы;
- признаки ненормальной работы приводов и механизмов (формируемые в автоматах низового управления приводами);
- значения уставок.

4. Команды, полученные с верхнего уровня или формируемые в контроллере S7-300.

Опишем процедуры, производимые в контроллерах Master (S7-300) и Slave (S7-200) для обеспечения гарантированной передачи информации:

1. Контроллер Master (S7-300) посылает всю полученную от контроллера Slave (S7-200) информацию обратно (в первый раз

посылаются нули). Он также посылает дополнительную информацию (новые значения уставок и команды, полученные с верхнего уровня или формируемые в контроллере S7-300).

2. Контроллер Slave (S7-200) сравнивает полученную информацию (кроме команд и уставок, если соответствующие биты редактирования сброшены, или кроме только команд, если соответствующие биты редактирования установлены) с отосланной ранее и если РЕЗУЛЬТАТЫ ИДЕНТИЧНЫ, то считается, что передача прошла успешно.

При этом формируется новая информация (весь выходной буфер) и ПОСЛЕ ЭТОГО стираются биты редактирования уставок (записываемые из индикатора TD200 в ПЕРВОМ СКАНЕ программы).

Выходной буфер формируется полностью также и в первом скане программы.

3. Если полученная информация НЕ ИДЕНТИЧНА посланной ранее, то:

a) Биты, у которых значимым значением является только TRUE (например, признаки предупредительной или аварийной сигнализации), записываются в выходной буфер ТОЛЬКО если их текущее значение равно TRUE. Перезапись производится с помощью инструкции S. Таким образом, когда передача будет успешной, Мастером будут гарантированно получены именно значимые значения битов.

b) Значение уставки, измененной в контроллере S7-200, записывается в выходной буфер контроллера S7-200. Об этом сигнализирует установленный бит редактирования этой уставки. Этот бит стирается ТОЛЬКО при успешной передаче от контроллера S7-200 к контроллеру S7-300.

c) Остальная информация не переписывается (кроме масштабированных значений параметров и входных дискретных сигналов).

4. Значение уставки, полученной из контроллера S7-300, считывается из входного буфера контроллера S7-200 (и соответственно переписывается в реальную уставку) ТОЛЬКО когда сброшен бит редактирования этой уставки. Этот бит стирается ТОЛЬКО при успешной передаче от контроллера S7-200 к

контроллеру S7-300. Рекомендации к этой процедуре полностью совпадают с приведенными в процедуре 7.

Команды, присылаемые по сети (формируемые в контроллере Master), отсылаются обратно (повторяются).

Программа контроллера Master должна стирать отсылаемую команду (для контроллера Slave входную), если получено повторение от контроллера Slave. До тех пор контроллер Master будет отсылать команду со значением TRUE.

В контроллере Slave биты команд, присылаемых из сети, стираются в первом скане программы. В подпрограммах управления приводами воспринимается только передний фронт команды из сети.

ГЛАВА 7
ОСОБЕННОСТИ ПРИМЕНЕНИЯ ТЕХНОЛОГИИ В КОНТРОЛЛЕРАХ С МАЛЫМ ОБЪЕМОМ ПАМЯТИ И ЯЗЫКАМИ ИНСТРУКЦИЙ ТИПА STL

Необходимо отметить следующее.

1. Технология предположительно является кодоемкой. Поэтому, например, надо стремиться к минимизации числа состояний в автомате, так как используемые при определении текущего номера состояния инструкции JMP "съедают" много байт программы, да и инструкции S и R весьма «прожорливы». Правда, автор МЕТОДИКИ не пользовался при разработке программ другими методами, и ему не с чем сравнивать. Поэтому этот недостаток может быть только кажущимся.

Пока самый большой набор оборудования, которым автор управлял с помощью контроллера S7-200 с использованием данной технологии – это система из двух насосов и пяти задвижек и система из трех переверсивных приводов и четырех задвижек. Код программы, реализующий автомат управления задвижки занимал 878 байт, а вместе с вызовом этой подпрограммы для четырех задвижек – 2,5 Кбайт. Память для хранения кода всей программы была использована примерно на 60–80 % – применялся контроллер S7-224 с 8 Кбайт программной памяти. Естественно, это код всех реализованных функций, включая примерно 2–3 Кбайт на обеспечение поддержки обмена по сети.

В этом кажущемся большим коде программы **заложено всё необходимое для эффективного контроля и управления.**

2. Рекомендуется автоматизировать построения кода программы по графу переходов автомата. Это может повысить «комфортность» применения контроллеров, так как языки, используемые в них, не очень "высокого уровня", да и рутинные операции никогда не вызывают энтузиазма у исполнителей.

Хочется добавить, что **сама возможность формального (в идеале полностью автоматизированного) программирования обеспечивает SWITCH-технологии огромное преимущество перед другими методами программирования последовательностной логики.**

3. К сожалению, ограниченный размер памяти и выходного буфера обмена по сети, а также большой скан опроса контроллера Master не позволяют проводить полноценное протоколирование работы автоматов, как это предлагается делать в рамках SWITCH-технологии. Вернее сказать, организовать хранение нормального качественного протокола (с зафиксированными до миллисекунд моментами смены состояния и полным набором входных/выходных данных при заранее неизвестном числе переходов) при использовании контроллера S7-200 достаточно трудоемко. Однако в контроллере S7-300 (конкретно S7-317-2DP) такой проблемы не было и сохраняемые протоколы очень помогли при наладке. Однако даже наблюдение за изменением номера состояния автомата в реальном времени дает очень многое, и это, пожалуй, самый большой плюс рассматриваемой технологии при наладке.

ЗАКЛЮЧЕНИЕ

Данная технология позволяет эффективно разрабатывать и реализовать алгоритмы логического управления **локальным оборудованием** с использованием **контроллера S7-200.** Естественно, **это может быть и любой другой контроллер**, позволяющий писать программы **в текстовом виде.**

ПРИЛОЖЕНИЯ

Приложение 1
SWITCH-технология®

SWITCH-технология® позволяет проводить формальную реализацию алгоритмов. Следствием этого является возможность **автоматического преобразования графа переходов автомата в код программы**, что применительно к контроллеру S7-200 реализовано К.В. Вавиловым с использованием интерфейса редактора Visio. Таким образом, получение логической части программы, реализующей основную задачу управления оборудованием, не требует затрат времени и сил на программирование. **Главное преимущество этой технологии – текст логической части программы полностью соответствует алгоритму.**

Эффективность SWITCH-технологии® возрастает со сложностью решаемой логической задачи.

Краткое описание принципов автоматного программирования (SWITCH-технологии)

Основное понятие – "состояние". В состоянии автомат ожидает определенный набор входных воздействий (признаков, значений или событий). Состояния бывают зависящими от времени (когда появление входного воздействия ожидают в течение определенного периода времени) и не зависящими от времени.

При появлении необходимого входного воздействия производится переход в другое состояние. Если требуется, то при переходе выполняется выходное воздействие (например, запись/стирание признака). Такое описание соответствует автомату Мили. При соответствующем входном воздействии может выполняться переход в то же состояние с выполнением выходного воздействия.

Алгоритм задается двумя диаграммами – схемой связей и графом переходов.

Схема связей задает интерфейс автомата и содержит перечисление наименований и обозначений входных и выходных воздействий. Обозначения входных и выходных воздействий в схеме связей и графе переходов строго соответствуют.

Граф переходов представляет собой набор вершин, соответствующих определенным состояниям, соединенных между собой направленными дугами — переходами. Каждой дуге соответствует условие перехода с определенным уровнем приоритета проверки и перечисление (для автомата Мили) необходимых выходных воздействий при переходе. Важной особенностью реализации автомата и, что чрезвычайно важно, программы, также являющейся автоматом, построенных на основе SWITCH-технологии, является то, что пребывание в некотором состоянии не означает, что выполнение программы, реализующей автомат, "останавливается" до появления какого-либо входного воздействия. Наоборот, если нет ожидаемого входного воздействия, производится "выход из автомата" (окончание работы с автоматом), а затем, если необходимо, новый вход в автомат, проверка появления входного воздействия, снова "выход" и т.д. Таким образом, вызов (выполнение) автомата сводится к проверке нескольких условий и выполнению выходного воздействия, если необходимое условие выполняется. После этого осуществляется соответствующий переход. Исходя из изложенного, понятно, что прекратить вызов (выполнение) автомата можно в любой момент времени (он не зациклен логически).

В системах, как правило, имеется один главный автомат, в который вложены другие автоматы, которые, в свою очередь, также могут иметь вложенные автоматы.

Существенным также является то, что при использовании этой технологии в принципе можно обойтись без так называемых "промежуточных переменных", так как номер текущего состояния, а также номер следующего состояния, являются исчерпывающей информацией для выполнения перехода.

В общем случае, выходные воздействия могут формироваться в состояниях (автомат Мура), а также в состояниях и на переходах (смешанный автомат).

Из-за низкой информативности номера состояния (это просто число) рекомендуется применять промежуточные

признаки (переменные). При этом необходимо выполнять следующие требования.

1. Рекомендуется производить процедуры записи/стирания применяемого промежуточного признака в одном и том же автомате и только в нем.

2. Перед вызовом главного автомата или при переходе в начальное состояние главного автомата производится стирание всех промежуточных признаков.

3. Если в графе переходов автомата применяется логическая (условная) запись признака, необходимо предусмотреть безусловное стирание признака при каждом новом вызове автомата. Таким образом, период существования такого признака равен одному скану вызова автомата. Такой признак должен записываться и стираться только в данном автомате.

4. Если в разных автоматах производится запись и стирание признака (и то и другое в каждом автомате, и в одном из автоматов применяется безусловное стирание), то одновременный (реально последовательный) или вложенный вызов этих автоматов должен быть исключен.

5. Если производится запись и стирание признака в одном автомате (и то и другое условные – применяются в графе переходов) и только стирание в другой процедуре (другом автомате), то вызов этого автомата и процедуры должны быть строго последовательными.

Для повышения информативности для обозначения входных и выходных воздействий применяются не символические обозначения, а идентификаторы.

Приложение 2
Психология ответственности и формализация логики

В алгоритмах постоянно не учитывается то, что необходимо программистам для реализации, а текст программы мало похож на алгоритм. Таким образом, существуют два алгоритма: один на бумаге (для отчетности и документирования проектных решений), содержащий обычно некоторый результат проектирования, а не путь для его получения, а второй – в голове программиста (сохраняемый, правда, еще и в текстовом виде программы).После написания окончательного текста программы зачастую предпринимаются попытки изменения документации, но опять учитывается не все. При этом логическая часть программы скорее всего отличается от логики алгоритма – нет полного соответствия. Практически **никто и никогда не проверяет текст программы**, управляющей технологическим оборудованием. Если
программа большая, то при традиционном подходе проверить по тексту соответствие алгоритму невозможно. Для проверки правильности реализации используется некая процедура под названием "Тестирование". При этом по существу проверяется как программист понял алгоритм (тот, который на бумаге) и как преобразовал его в алгоритм в своей голове, а далее в текст программы. В итоге программист является единственным держателем важнейшей логической информации и становится абсолютно не важным, что было придумано до программной реализации. Дело в том, что **каждый программист по-своему "строит" текст программы,** в зависимости от интеллекта и знания языка программирования. В любом случае используется множество промежуточных переменных, которые программист вводит и применяет по своему усмотрению и которые, естественно, "влияют" на логику. И если программа большая и сложная по поведению, то для того, чтобы понять почему что-то логически неправильно выполняется, **необходим квалифицированный специалист, которому придется разбираться уже в тексте программы.**

Большинство программистов, не любят документировать алгоритмы перед программированием (и даже просто рисовать их на бумаге), скорее всего потому, что все равно придется выдумывать что-то свое по ходу программирования. И правда, зачем терять

время на рисование каких-то прямоугольников, ромбиков и стрелочек, если лучше сначала сразу "попрограммировать", а потом изобразить примерно похожий или весьма общий алгоритм в документации. Все привыкли к этому: программисты, потому, что так легче, и постановщики задач, так как не всегда владеют знаниями по программированию в требуемом объеме, а если и владеют, то уж никак не могут влиять своевременно на то, что "вытворяют" программисты. Удобные среды программирования (даже в DOS, не говоря о визуальных под Windows) также способствуют именно такой последовательности разработки, так как все развитые средства отладки и наблюдения за значениями переменных позволяют программистам надеяться, что, якобы, можно выявить любую ошибку в логике.

Но время уходит, проект надо закончить к заданному сроку, а исполнитель сидит и придумывает, как ему разрешить ту или иную логическую часть задачи, да еще и успеть ее реализовать. А о логических ошибках, не выявленных при тестировании, и вспоминать не хочется, так как тестирование выполняется не лучше, чем программирование – так же хаотично.

Из изложенного следует, что ответственный подход к созданию ПО связан с переходом от хаоса к применению эффективных методик (в частности, SWITCH-технологии), поддерживающих все этапы жизненного цикла программ.

Приложение 3
Недостатки неавтоматного подхода к алгоритмизации управления

Следующий пример поможет понять минусы неавтоматного подхода к алгоритмизации управления для контроллеров.

Человек, находясь на одном этаже, должен добраться на лифте на другой. Техническое задание человеку (будем считать его условной системой управления лифтом):

- человек нажимает кнопку вызова лифта;
- когда двери подъехавшего лифта открылись, человек входит в лифт;
- человек нажимает кнопку нужного этажа;
- когда лифт остановился на нужном этаже и двери открылись, человек выходит из лифта.

Приведенная в техническом задании последовательность действий (алгоритм решения задачи) легко изображается в виде схемы представленной на рис. 15.

Это плохое решение. Поясним это.

Достаточно вспомнить концепцию выполнения цикла CPU S 7-200, и станет ясно, что **нельзя логически зацикливать программу**, хотя бы потому, что тогда контроллером **не будет производится** ввод информации, а без новой информации нельзя выйти из логического цикла. Кто-то возразит – есть же прерывания! На всю вводимую информацию прерываний не хватит. Для того чтобы предотвратить логическое зацикливание, вероятнее всего (и похоже это единственный вариант) придется фиксировать некие стадии (этапы), которые программа уже прошла, и «опустить» все связи вниз. Преобразованная схема алгоритма приведена на рис. 16.

Алгоритм на рис.15 представляет собой формализованное описание логики, соответствующее техническому заданию, и не учитывает никаких особенностей реализации.

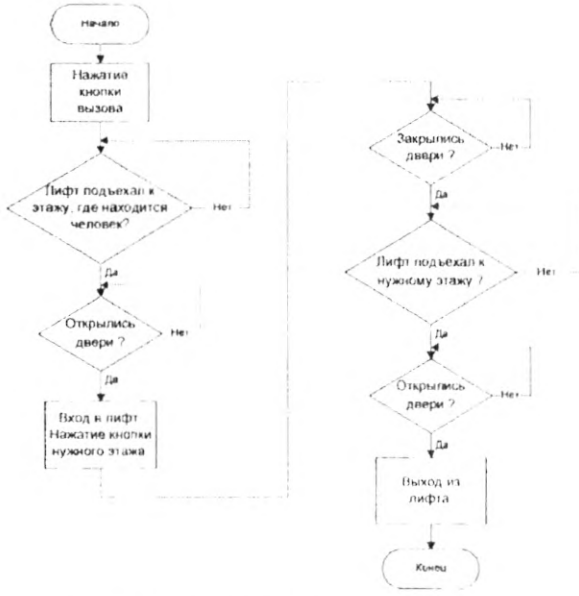


Рис. 15. Алгоритм решения задачи.

Алгоритм на рис.16 по существу является алгоритмом реализации в конкретной среде выполнения с учетом ее особенностей. Он также соответствует техническому заданию, но дополнен необходимой информацией для реализации алгоритма на рис.15 в указанной среде.

В техническом задании практически никогда не описываются временные (от слова "время") особенности процесса управления. В частности, в техническом задании описывается нормальная работа лифта (нормальная реакция на действия человека). Подразумевается, что когда-то (за конечное время) лифт подъедет к этажу, когда-то откроются двери, когда-то лифт доедет до нужного этажа и т.д. Первое, что придется сделать дополнительно к алгоритму на рис.16 – это **добавить контроль времени** к каждому пока "бесконечному" ожиданию события. Если после истечения времени контроля, событие не произошло, необходимо определить (и согласовать с заказчиком) действия

системы (в данном случае, человека) при этом. Однако и это далеко не все.

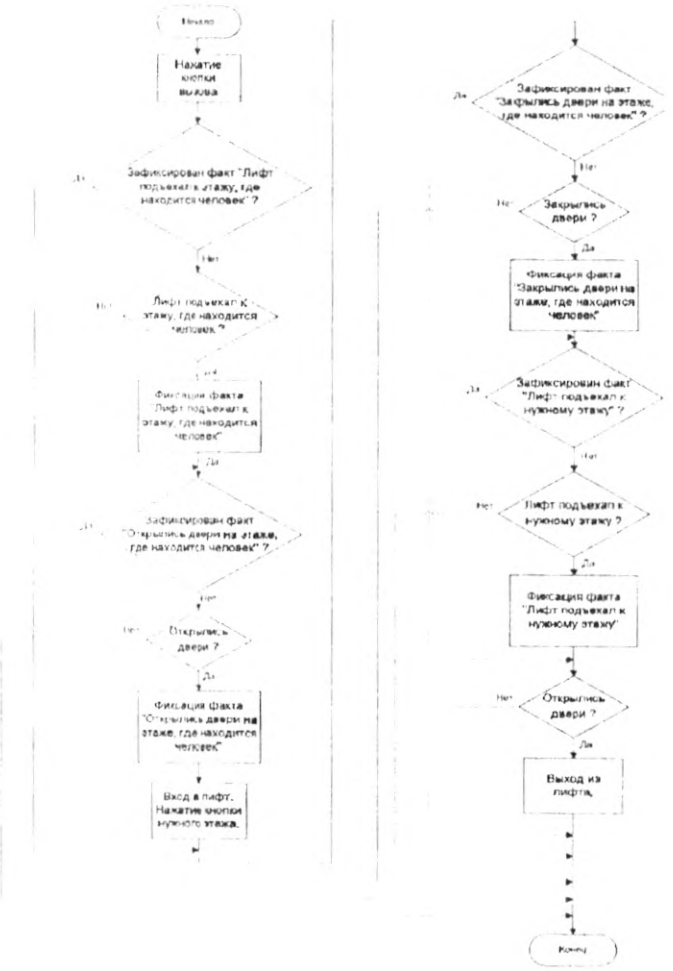


Рис. 16. Алгоритм реализации в конкретной среде выполнения с учетом ее особенностей.

Во время ожидания могут произойти события, нарушающие нормальный ход процесса: после нажатия кнопки лифт не поехал, лифт остановился не доехав, заклинило двери и т.д.

Выход из этих ситуаций на разных стадиях алгоритма будет производиться по-разному. Таким образом, алгоритм должен быть дополнен "нехорошими" условиями и соответственно, действиями для выхода из "нехороших" ситуаций, что отразить на схеме алгоритма весьма непросто, сильно не переделав его.

Теперь вроде бы все учтено. Осталось нарисовать этот новый алгоритм и заняться написанием программы по нему, благо в алгоритме заложено практически все необходимое для реализации.

Такая схема алгоритма будет весьма сложна. Она обычно не рисуется ввиду громоздкости. Серьезным недостатком такой схемы является не только плохая "читабельность" (из-за размеров и, например, наличия огромного числа пересекающихся линий), но и сложность ее логического анализа. Приходится учитывать промежуточную информацию, существующую только для того, чтобы обеспечить корректность реализации конкретным методом (в данном алгоритме это признаки фиксации прошедшей стадии). И это еще без учета собственно программирования, которое придется делать на не очень удобном ассемблероподобном языке, которые только и есть у контроллера S7-200.

Однако, если вдуматься, то можно легко сообразить, что **фиксация прошедшей стадии алгоритма есть ничто иное как введение неких состояний!**

Остается применить SWITCH-технологию®, перерисовав алгоритм в виде графа переходов, который может быть изображен более компактно и изоморфно реализован в виде текста программы даже на языке низкого уровня.

Приложение 4
Автоматизация кодирования

Создание конвертора схемы (диаграммы) автомата, выполненного в виде графа переходов в код программы требует прежде всего четкое понимание принципов SWITCH-технологии и четкое представление о структуре кода будущей программы. Все остальное технически просто.

Первый конвертор был создан автором МЕТОДИКИ весной 2002 г. с применением программного интерфейса редактора **Word .**

В конце лета 2002 года вышла версия конвертора, использующего интерфейс редактора Visio (Головешин А. Конвертор для преобразования графов переходов в текст программы на языке Си Visio2Switch. 2002, http://is.ifmo.ru/progeny/visio2switch). После этой публикации автор решил перейти на этот более удобный редактор (**Visio**).

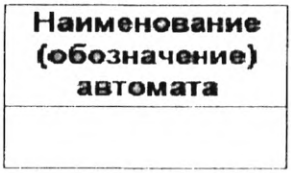
Разработка велась в среде графического программирования LabVIEW 6.1, так как автор МЕТОДИКИ не является профессиональным (квалифицированным) программистом, и не владеет такими средствами как Visual C или Visual Basic. Однако это обстоятельство нисколько не повлияло на результат, ибо пакет LabVIEW содержит все необходимые инструменты для доступа к программному интерфейсу редактора Visio.

Естественным условием перед началом конвертации является соответствующая подготовка документа Visio. Для этой цели служит шаблон (Visio Stencil), содержащий объекты (masters), с помощью которых пользователь конструирует автоматные графы (Головешин А.).

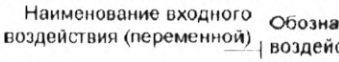
Объекты **masters** легко создаются вручную, Visio содержит в себе соответствующий редактор.

Объекты схемы связей:

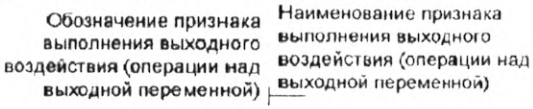
Объект "Наименование и обозначение автомата"



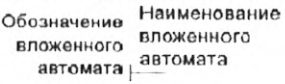
Объект "Наименование и обозначение входного воздействия"



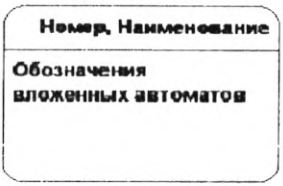
Объект "Наименование и обозначение признака выходного воздействия"



Объект "Наименование и обозначение вложенного подавтомата"



Объекты графа переходов:
Объект "Состояние"



Один из Объектов "Дуга перехода"

Каждый **объект** является группой **подобъектов**.

Для того чтобы осуществить "расшифровку" рисунка, необходимо просто поименовать произвольными обозначениями (но однозначными и неповторяющимися) шаблоны объектов и подобъектов в специальных полях окна их свойств. Автомат строится ("изображается"), используя эти шаблоны. Далее программно собирается (например, в массивы) "содержимое" каждого подобъекта, и формируется код программы.

Самым сложным является следующее.

1. Условия переходов в синтаксисе языка Си необходимо "перевести" в синтаксис языка STL программы контроллера.

2. Необходимо соотнести каждую дугу переходов с начальным и конечным для нее состояниями. Одним из вариантов (который применяет автор МЕТОДИКИ) является определение "попадания" начала и конца дуги в область, занимаемую объектами "Состояние".

3. Необходимо программно проверять правильность подготовки документа перед конвертацией и информировать пользователя о допущенных ошибках. Наиболее часто возникают следующие ошибки редактирования:

- иногда при копировании объектов пропадают их обозначения или обозначения подобъектов ("глюки" редактора Visio);
- для входных данных используются объекты, предназначенные для выходных данных и наоборот;
- некорректно набраны условия.

Даже если есть уверенность в корректности конвертации, необходимо проверить сгенерированный код визуально (особенно, если что-то не получается при тестировании).

Приложение 5
Примеры сгенерированного кода автоматов проекта

Автомат *ACPx* (рис. 5, 6)

```
SUBROUTINE_BLOCK ACPx:SBR19
TITLE=SUBROUTINE COMMENTS
//  СОЗДАНО с использованием КОНВЕРТОРА
//  графа переходов (изображенного в Visio) в текст программы
//
//  ********************************
//  **** Входные переменные
//  ********************************
//  KoffP - Команда "Отключить"
//  KonP - Команда "Включить"
//  TACP1 - Окончилось время контроля включения/отключения
пускателя
//  X0P - Включено питание привода
//  X1P - Включен
//  ********************************
//  **** Выходные процедуры
//  ********************************
//  zM1P - Запись признака "Нет включения пускателя за
контрольное время "
//  zM2P - Запись признака "Нет отключения пускателя за
контрольное время "
//  zM3P - Запись признака "Отключение пускателя по внешним
причинам"
//  zNZ1P - Стирание признака "Управление работой"
//  zZ1P - Запись признака "Управление работой"
//  zstacp1 - Начать контроль времени включения/отключения
пускателя
//  ****************************************
//  **** Вызываемые автоматы
//  ****************************************
//
BEGIN
```

```
Network 1 // Обнуление (сброс) команд , признаков и т.п. ,
записываемых в автомате
LDN     SM0.0
=       zM1P
=       zM2P
=       zM3P
=       zNZ1P
=       zZ1P
=       zstacp1
Network 2 //  ОПРЕДЕЛЕНИЕ ТЕКУЩЕГО СОСТОЯНИЯ АНАЛОГ
--- case: 0
LDB=    YACPx , 0
JMP     0
Network 3
//  ОПРЕДЕЛЕНИЕ ТЕКУЩЕГО СОСТОЯНИЯ АНАЛОГ --- case: 1
LDB=    YACPx , 1
JMP     1
Network 4
//  ОПРЕДЕЛЕНИЕ ТЕКУЩЕГО СОСТОЯНИЯ АНАЛОГ --- case:
10
LDB=    YACPx , 10
JMP     10
Network 5
//  ОПРЕДЕЛЕНИЕ ТЕКУЩЕГО СОСТОЯНИЯ АНАЛОГ --- case:
11
LDB=    YACPx , 11
JMP     11
Network 6
//  Переход по умолчанию
LD      SM0.0
JMP     255
Network 7
//  ПОСЛЕДОВАТЕЛЬНОСТЬ ПРОВЕРКИ УСЛОВИЙ И
ДЕЙСТВИЙ в состоянии 0
LBL     0
Network 8
//  Переход с приоритетом 1 из состояния 0 в состояние 1
```

```
LD          X1P
MOVB        1 , YACPx
JMP         255
Network 9
//  Переход с приоритетом 2 из состояния 0 в состояние 10
LD          KonP
A           X0P
MOVB        10 , YACPx
=           zZ1P
=           zstacp1
JMP         255
Network 10
//  Переход по умолчанию
LD          SM0.0
JMP         255
Network 11
//  ПОСЛЕДОВАТЕЛЬНОСТЬ ПРОВЕРКИ УСЛОВИЙ И
ДЕЙСТВИЙ в состоянии 1
LBL 1
Network 12
//  Переход с приоритетом 1 из состояния 1 в состояние 0
LDN         X1P
ON          X0P
MOVB        0 , YACPx
=           zNZ1P
=           zM3P
JMP         255
Network 13
//  Переход с приоритетом 2 из состояния 1 в состояние 11
LD          KoffP
MOVB        11 , YACPx
=           zNZ1P
=           zstacp1
JMP         255
Network 14
//  Переход по умолчанию
LD          SM0.0
```

```
JMP     255
Network 15
// ПОСЛЕДОВАТЕЛЬНОСТЬ ПРОВЕРКИ УСЛОВИЙ И
ДЕЙСТВИЙ в состоянии 10
LBL     10
Network 16
// Переход с приоритетом 1 из состояния 10 в состояние 1
LD      X1P
MOVB    1 , YACPx
JMP     255
Network 17
// Переход с приоритетом 2 из состояния 10 в состояние 0
LD      TACP1
ON      X0P
MOVB    0 , YACPx
=       zNZ1P
=       zM1P
JMP     255
Network 18
// Переход по умолчанию
LD      SM0.0
JMP     255
Network 19
// ПОСЛЕДОВАТЕЛЬНОСТЬ ПРОВЕРКИ УСЛОВИЙ И
ДЕЙСТВИЙ в состоянии 11
LBL     11
Network 20
// Переход с приоритетом 1 из состояния 11 в состояние 0

LDN     X1P
MOVB    0 , YACPx
JMP     255
Network 21
// Переход с приоритетом 2 из состояния 11 в состояние 1
LD      TACP1
MOVB    1 , YACPx
=       zM2P
JMP     255
```

```
Network 22
LBL          255

END_SUBROUTINE_BLOCK
```

Автомат ACSx (рис. 7, 8)

```
SUBROUTINE_BLOCK ACSx:SBR17
TITLE=SUBROUTINE COMMENTS
//  СОЗДАНО с использованием КОНВЕРТОРА
//  графа переходов алгоритма (изображенного в Visio) в текст программы
//
//  ******************************
//  **** Входные переменные
//  ******************************
//  KclS - Команда "Закрыть"
//  KoffS - Команда "Остановить"
//  KopS - Команда "Открыть"
//  TACS1 - Окончилось время контроля включения пускателя
//  TACS2 - Окончилось время контроля открытия/закрытия затвора
//  X0S - Включено питание привода
//  X1S - Включен на открытие
//  X2S - Включен на закрытие
//  X3S - Открыт
//  X4S - Закрыт
//  noneX1S_time - Пускатель на открытие отключен по внешней причине (проверено временем)
//  noneX2S_time - Пускатель на закрытие отключен по внешней причине (проверено временем)
//  ******************************
//  **** Выходные процедуры
//  ******************************
//  zM1S - Запись признака "Нет открытия затвора за контрольное время"
//  zM2S - Запись признака "Нет закрытия затвора за контрольное время"
```

```
//   zM3S - Запись признака "Нет включения пускателя на открытие
за контрольное время "
//   zM4S - Запись признака "Нет включения пускателя на закрытие
за контрольное время "
//   zM7S - Запись признака "Отключение пускателя на открытие по
внешним причинам"
//   zM8S - Запись признака "Отключение пускателя на закрытие по
внешним причинам"
//   zMclS - Запись признака "Ненормальное закрытие затвора"
//   zMopS - Запись признака "Ненормальное открытие затвора"
//   zNZ1S - Стирание признака "Управление открытием"
//   zNZ2S - Стирание признака "Управление закрытием"
//   zZ1S - Запись признака "Управление открытием"
//   zZ2S - Запись признака "Управление закрытием"
//   zstacs1 - Начать контроль времени включения/отключения
пускателя
//   zstacs2 - Начать контроль времени открытия/закрытия затвора
//   ***************************************
//   **** Вызываемые подАвтоматы
//   ***************************************
//
BEGIN
Network 1
//   Обнуление (сброс) команд , признаков и т.п. , записываемых в
автомате
LDN          SM0.0
=            zM1S
=            zM2S
=            zM3S
=            zM4S
=            zM7S
=            zM8S
=            zMclS
=            zMopS
=            zNZ1S
=            zNZ2S
=            zZ1S
=            zZ2S
```

```
=               zstacs1
=               zstacs2
Network 2
// ОПРЕДЕЛЕНИЕ ТЕКУЩЕГО СОСТОЯНИЯ АНАЛОГ --- case: 0
LDB=            YACSx , 0
JMP             0
Network 3
// ОПРЕДЕЛЕНИЕ ТЕКУЩЕГО СОСТОЯНИЯ АНАЛОГ --- case: 1
LDB=            YACSx , 1
JMP             1
Network 4
// ОПРЕДЕЛЕНИЕ ТЕКУЩЕГО СОСТОЯНИЯ АНАЛОГ --- case: 2
LDB=            YACSx , 2
JMP             2
Network 5
// ОПРЕДЕЛЕНИЕ ТЕКУЩЕГО СОСТОЯНИЯ АНАЛОГ --- case:
101
LDB=            YACSx , 101
JMP             101
Network 6
// ОПРЕДЕЛЕНИЕ ТЕКУЩЕГО СОСТОЯНИЯ АНАЛОГ --- case:
102
LDB=            YACSx , 102
JMP             102
Network 7
// ОПРЕДЕЛЕНИЕ ТЕКУЩЕГО СОСТОЯНИЯ АНАЛОГ --- case:
103
LDB= YACSx , 103
JMP 103
Network 8
// ОПРЕДЕЛЕНИЕ ТЕКУЩЕГО СОСТОЯНИЯ АНАЛОГ --- case:
104
LDB=            YACSx , 104
JMP             104
Network 9
// Переход по умолчанию
LD              SM0.0
JMP             255
```

```
Network 10
// ПОСЛЕДОВАТЕЛЬНОСТЬ ПРОВЕРКИ УСЛОВИЙ И
ДЕЙСТВИЙ в состоянии 0
LBL         0
Network 11
// Переход с приоритетом 1 из состояния 0 в состояние 101
LD          KopS
A           X0S
MOVB        101 , YACSx
=           zZ1S
=           zstacs1
JMP         255
Network 12
// Переход с приоритетом 2 из состояния 0 в состояние 102
LD          KclS
A           X0S
MOVB        102 , YACSx
=           zZ2S
=           zstacs1
JMP         255
Network 13
// Переход с приоритетом 3 из состояния 0 в состояние 1
LD          X4S
MOVB        1 , YACSx
JMP         255
Network 14
// Переход с приоритетом 4 из состояния 0 в состояние 2
LD          X3S
MOVB        2 , YACSx
JMP         255
Network 15
// Переход по умолчанию
LD          SM0.0
JMP         255
Network 16
// ПОСЛЕДОВАТЕЛЬНОСТЬ ПРОВЕРКИ УСЛОВИЙ И
ДЕЙСТВИЙ в состоянии 1
LBL         1
```

```
Network 17
// Переход с приоритетом 1 из состояния 1 в состояние 101
LD      KopS
A       X0S
MOVB    101 , YACSx
=       zZ1S
=       zstacs1
JMP     255
Network 18
// Переход с приоритетом 2 из состояния 1 в состояние 0
LDN     X4S
MOVB    0 , YACSx
JMP     255
Network 19
// Переход по умолчанию
LD      SM0.0
JMP     255
Network 20
// ПОСЛЕДОВАТЕЛЬНОСТЬ ПРОВЕРКИ УСЛОВИЙ И
ДЕЙСТВИЙ в состоянии 2
LBL     2
Network 21
// Переход с приоритетом 1 из состояния 2 в состояние 102
LD      KclS
A       X0S
MOVB    102 , YACSx
=       zZ2S
=       zstacs1
JMP     255
Network 22
// Переход с приоритетом 2 из состояния 2 в состояние 0
LDN     X3S
MOVB    0 , YACSx
JMP     255
Network 23
// Переход по умолчанию
LD      SM0.0
JMP     255
```

```
Network 24
// ПОСЛЕДОВАТЕЛЬНОСТЬ ПРОВЕРКИ УСЛОВИЙ И
ДЕЙСТВИЙ в состоянии 101
LBL             101
Network 25
// Переход с приоритетом 1 из состояния 101 в состояние 103
LD              X1S
MOVB            103 , YACSx
=               zstacs2
JMP             255
Network 26
// Переход с приоритетом 2 из состояния 101 в состояние 1
LD              TACS1
O                 KoffS
A               X4S
MOVB            1 , YACSx
=               zNZ1S
=               zM3S
=               zMopS
JMP             255
Network 27
// Переход с приоритетом 3 из состояния 101 в состояние 0
LD              TACS1
O               KoffS
AN              X4S
MOVB            0 , YACSx
=               zNZ1S
=               zM3S
=               zMopS
JMP             255
Network 28
// Переход по умолчанию
LD              SM0.0
JMP             255
Network 29
// ПОСЛЕДОВАТЕЛЬНОСТЬ ПРОВЕРКИ УСЛОВИЙ И
ДЕЙСТВИЙ в состоянии 102
LBL               102
```

```
Network 30
// Переход с приоритетом 1 из состояния 102 в состояние 104
LD          X2S
MOVB        104 , YACSx
=           zstacs2
JMP         255
Network 31
// Переход с приоритетом 2 из состояния 102 в состояние 2
LD          TACS1
O           KoffS
A           X3S
MOVB        2 , YACSx
=           zNZ2S
=           zM4S
=           zMclS
JMP 255
Network 32
// Переход с приоритетом 3 из состояния 102 в состояние 0
LD          TACS1
O           KoffS
AN          X3S
MOVB        0 , YACSx
=           zNZ2S
=           zM4S
=           zMclS
JMP         255
Network 33
// Переход по умолчанию
LD          SM0.0
JMP         255
Network 34
// ПОСЛЕДОВАТЕЛЬНОСТЬ ПРОВЕРКИ УСЛОВИЙ И
ДЕЙСТВИЙ в состоянии 103
LBL          103
Network 35
// Переход с приоритетом 1 из состояния 103 в состояние 2
LD           X3S
MOVB         2 , YACSx
```

```
=                zNZ1S
JMP              255
Network 36
// Переход с приоритетом 2 из состояния 103 в состояние 0
LD               KoffS
AN               X3S
MOVB             0 , YACSx
=                zNZ1S
JMP              255
Network 37
// Переход с приоритетом 3 из состояния 103 в состояние 0
LD               noneX1S_time
ON               X0S
MOVB             0 , YACSx
=                zNZ1S
=                zM7S
=                zMopS
JMP 255
Network 38
// Переход с приоритетом 4 из состояния 103 в состояние 103
LD               TACS2
=                zM1S
=                zMopS
=                zstacs2
JMP              255
Network 39
// Переход по умолчанию
LD               SM0.0
JMP              255
Network 40
// ПОСЛЕДОВАТЕЛЬНОСТЬ ПРОВЕРКИ УСЛОВИЙ И
ДЕЙСТВИЙ в состоянии 104
LBL              104
Network 41
// Переход с приоритетом 1 из состояния 104 в состояние 1
LD               X4S
MOVB             1 , YACSx
=                zNZ2S
```

```
JMP             255
Network 42
//  Переход с приоритетом 2 из состояния 104 в состояние 0
LD              KoffS
AN              X4S
MOVB            0 , YACSx
=               zNZ2S
JMP             255
Network 43
//  Переход с приоритетом 3 из состояния 104 в состояние 0
LD              noneX2S_time
ON              X0S
MOVB            0 , YACSx
=               zNZ2S
=               zM8S
=               zMclS
JMP             255
Network 44
//  Переход с приоритетом 4 из состояния 104 в состояние 104
LD              TACS2
=               zM2S
=               zMclS
=               zstacs2
JMP             255
Network 45
LBL             255

END_SUBROUTINE_BLOCK
```

Автомат ATWPSFW (рис. 11, 12)

```
SUBROUTINE_BLOCK ATWPSFW:SBR15
TITLE=SUBROUTINE COMMENTS
//  СОЗДАНО с использованием КОНВЕРТОРА
//  графа переходов алгоритма (изображенного в Visio) в текст
программы
//
//  ******************************
```

```
//  **** Входные переменные
//  ********************************
//  CclS_FW - Есть все условия для автоматического закрытия
напорной задвижки (X0 - Включено
питание привода задвижки !X1 - Не включен привод задвижки на
открытие !X5 - Задвижка не
заклинена !HTD - Нет превышения температуры двигателя напорной
задвижки)
//  ConP_WP - Есть все условия для автоматического пуска
промывного насоса (DDI_ReadyWP -
Насосный агрегат к пуску готов)
//  CopS_FW - Есть все условия для автоматического открытия
напорной задвижки (X0 - Включено
питание привода задвижки !X2 - Не включен привод задвижки на
закрытие !X5 - Задвижка не
заклинена !HTD - Нет превышения температуры двигателя напорной
задвижки)
//  DDI_X1P_WP - Насосный агрегат включен
//  DDI_X3S_FW - Напорная задвижка открыта
//  DDI_X4S_FW - Напорная задвижка закрыта
//  DI_CCEstopWP - Команда от ШКО "Аварийное отключение
насосного агрегата"
//  DI_CCoffWP - Команда от ШКО "Отключить насосный агрегат"
//  M1P_WP - Нет включения пускателя привода промывного насоса
за контрольное время
//  M2P_WP - Нет отключения пускателя привода промывного
насоса за контрольное время
//  MclS_FW - Ненормальное закрытие напорной задвижки
//  MopS_FW - Ненормальное открытие напорной задвижки
//  PBEstopWP - Команда по сети PROFIBUS "Аварийная остановка
насосного агрегата"
//  PBoffWP - Команда по сети PROFIBUS "Отключить насосный
агрегат"
//  TPS - Окончилось время контроля предпусковой сигнализации
//  ********************************
//  **** Выходные процедуры
//  ********************************
```

```
//   zATWPSFWoff - Запись признака "Цикл управления НА
завершился нормально"
//   zATWPSFWoffErr - Запись признака "Цикл управления НА
завершился ненормально"
//   zDDO_predpuskWP - Запись команды "Предупреждение пуска"
//   zKclS_FW - Запись команды "Закрыть напорную задвижку"
//   zKoffP_WP - Запись команды "Отключить привод насосного
агрегата"
//   zKoffS_FW - Запись команды "Остановить привод напорной
задвижки"
//   zKonP_WP - Запись команды "Включить привод насосного
агрегата"
//   zKopS_FW - Запись команды "Открыть напорную задвижку"
//   zMNCclS_FW - Запись признака "Нет условий для
автоматического закрытия напорной задвижки"
//   zMNConP_WP - Запись признака "Нет готовности к пуску
промывного насоса"
//   zMNCopS_FW - Запись признака "Нет условий для
автоматического открытия напорной задвижки"
//   zNDDO_predpuskWP - Стирание команды "Предупреждение
пуска"
//   zNKclS_FW - Стирание команды "Закрыть напорную задвижку"
//   zNKopS_FW - Стирание команды "Открыть напорную задвижку"
//   zstps - Начать контроль времени предпусковой сигнализации
//   ***************************************
//   **** Вызываемые подАвтоматы
//   ***************************************
//
BEGIN
Network 1
//   Обнуление (сброс) команд , признаков и т.п. , записываемых в
автомате ,
//   но не имеющих обнуления по условию
LDN          SM0.0
=            ATWPSFWoff
=            ATWPSFWoffErr
=            KoffP_WP
=            KoffS_FW
```

```
=           KonP_WP
=           MNCclS_FW
=           MNConP_WP
=           MNCopS_FW
=           stps
Network 2
// ОПРЕДЕЛЕНИЕ ТЕКУЩЕГО СОСТОЯНИЯ АНАЛОГ --- case: 0
LDB=        YATWPSFW , 0
JMP         0
Network 3
// ОПРЕДЕЛЕНИЕ ТЕКУЩЕГО СОСТОЯНИЯ АНАЛОГ --- case: 1
LDB=        YATWPSFW , 1
JMP         1
Network 4
// ОПРЕДЕЛЕНИЕ ТЕКУЩЕГО СОСТОЯНИЯ АНАЛОГ --- case: 2
LDB=        YATWPSFW , 2
JMP         2
Network 5
// ОПРЕДЕЛЕНИЕ ТЕКУЩЕГО СОСТОЯНИЯ АНАЛОГ --- case: 3
LDB=        YATWPSFW , 3
JMP         3
Network 6
// ОПРЕДЕЛЕНИЕ ТЕКУЩЕГО СОСТОЯНИЯ АНАЛОГ --- case: 4
LDB=        YATWPSFW , 4
JMP         4
Network 7
// ОПРЕДЕЛЕНИЕ ТЕКУЩЕГО СОСТОЯНИЯ АНАЛОГ --- case: 5
LDB=        YATWPSFW , 5
JMP         5
Network 8
// ОПРЕДЕЛЕНИЕ ТЕКУЩЕГО СОСТОЯНИЯ АНАЛОГ --- case: 6
LDB=        YATWPSFW , 6
JMP         6
Network 9
// ОПРЕДЕЛЕНИЕ ТЕКУЩЕГО СОСТОЯНИЯ АНАЛОГ --- case: 7
LDB=        YATWPSFW , 7
JMP         255                    // Из состояния нет переходов
Network 10
```

```
// ОПРЕДЕЛЕНИЕ ТЕКУЩЕГО СОСТОЯНИЯ АНАЛОГ --- case: 8
LDB==           YATWPSFW , 8
JMP 8
Network 11
// ОПРЕДЕЛЕНИЕ ТЕКУЩЕГО СОСТОЯНИЯ АНАЛОГ --- case:
88
LDB=            YATWPSFW , 88
JMP             255                     // Из состояния нет переходов
Network 12
// Переход по умолчанию
LD              SM0.0
JMP             255
Network 13
// ПОСЛЕДОВАТЕЛЬНОСТЬ ПРОВЕРКИ УСЛОВИЙ И
ДЕЙСТВИЙ в состоянии 0
LBL             0
Network 14
// Переход с приоритетом 1 из состояния 0 в состояние 1
LD              SM0.0
MOVB            1 , YATWPSFW
= stps
S               DDO_predpuskWP , 1
JMP             255
Network 15
// Переход по умолчанию
LD              SM0.0
JMP             255
Network 16
// ПОСЛЕДОВАТЕЛЬНОСТЬ ПРОВЕРКИ УСЛОВИЙ И
ДЕЙСТВИЙ в состоянии 1
LBL             1
Network 17
// Переход с приоритетом 1 из состояния 1 в состояние 88
LD              DDI_X4S_FW
A               ConP_WP
LPS
NOT
MOVB            88 , YATWPSFW
```

```
LRD
NOT
=                MNConP_WP
LRD
NOT
=                ATWPSFWoffErr
LPP
NOT
JMP              255
Network 18
// Переход с приоритетом 2 из состояния 1 в состояние 7
LD               DI_CCEstopWP
O                PBEstopWP
O                DI_CCoffWP
O                PBoffWP
MOVB             7 , YATWPSFW
=                ATWPSFWoff
JMP              255
Network 19
// Переход с приоритетом 3 из состояния 1 в состояние 2
LD               TPS
A                CopS_FW
MOVB             2 , YATWPSFW
=                KonP_WP
R                DDO_predpuskWP , 1
JMP              255
Network 20
// Переход с приоритетом 4 из состояния 1 в состояние 88
LD               TPS
AN               CopS_FW
MOVB             88 , YATWPSFW
=                MNCopS_FW
R                DDO_predpuskWP , 1
                 ATWPSFWoffErr
JMP              255
Network 21
// Переход по умолчанию
LD               SM0.0
```

```
JMP             255
Network 22
// ПОСЛЕДОВАТЕЛЬНОСТЬ ПРОВЕРКИ УСЛОВИЙ И
ДЕЙСТВИЙ в состоянии 2
LBL             2
Network 23
// Переход с приоритетом 1 из состояния 2 в состояние 4
LD              DDI_X1P_WP
A               DDI_X3S_FW
MOVB            4 , YATWPSFW
JMP             255
Network 24
// Переход с приоритетом 2 из состояния 2 в состояние 3
LD              DDI_X1P_WP
A               CopS_FW
MOVB            3 , YATWPSFW
S               KopS_FW , 1
JMP             255
Network 25
// Переход с приоритетом 3 из состояния 2 в состояние 6
LD              DDI_X1P_WP
AN              CopS_FW
MOVB            6 , YATWPSFW
=               MNCopS_FW
=               KoffP_WP
JMP             255
Network 26
// Переход с приоритетом 4 из состояния 2 в состояние 88
LD              M1P_WP
MOVB            88 , YATWPSFW
=               ATWPSFWoffErr
JMP             255
Network 27
// Переход по умолчанию
LD              SM0.0
JMP             255
Network 28
```

Содержание

Редактор Ахметжанова Г.М.

Подписано к печати 30.06.2008 г. Формат 60x84 1/16.
Усл.п.л. 9,3. Тираж 50 экз. Заказ № 418.

Отпечатано в типографии ТГТУ. г.Ташкент,
ул.Талабалар 54. тел: 246-63-64.