К. В. Вавилов

Программируемые логические контроллеры *SIMATIC S7-200 (SIEMENS)*

Методика

алгоритмизации и программирования
задач логического управления

Санкт-Петербург

2005

Оглавление

ВВЕДЕНИЕ **3**

ЧАСТЬ 1. СОЗДАНИЕ МАТЕМАТИЧЕСКОГО И ИНФОРМАЦИОННОГО ОБЕСПЕЧЕНИЯ **4**

Глава 1. Первые шаги 4
1.1. Основные термины и обозначения *4*
1.2. Определяем функции локальной системы управления *4*
1.3. Начинаем разработку информационного обеспечения *5*
Глава 2. Базовые алгоритмы 6
2.1. Краткое описание концепции выполнения программы контроллером S7-200 ..... *6*
2.2. Виды и типы разрабатываемых алгоритмов (автоматов) управления *7*
2.3. Создание базовых автоматов *8*
2.3.1. Автомат потенциального управления нереверсивным приводом 8
2.3.1.1. Построение схемы связей автомата 8
2.3.1.2. Построение графа переходов автомата 9
2.3.1.3. Дополнение схемы связей и графа переходов необходимыми функциями контроля 10
2.4. Информационное обеспечение базовых алгоритмов *16*
2.4.1. Таблица символов автомата потенциального управления нереверсивным приводом 16
2.4.2. Таблица символов автомата потенциального управления реверсивным приводом 17
2.4.3. Таблица символов фактических параметров, используемых/формируемых при вызове автомата потенциального управления нереверсивным приводом 18
2.4.4. Таблица символов фактических параметров, используемых/формируемых при вызове автомата потенциального управления реверсивным приводом 19
Глава 3. Технологические алгоритмы 21
3.1. Создание технологического автомата управления насосным агрегатом *21*
3.1.1. Типичное техническое задание 21
3.1.3. Нормальный ход процесса 22
3.1.3. Дополняем автомат “нехорошими” событиями 24
3.1.4. «Главный» технологический автомат 27
3.2. Таблицы символов технологических автоматов *29*

ЧАСТЬ 2. РЕКОМЕНДАЦИИ ПО ПРОГРАММИРОВАНИЮ ОСНОВНЫХ ФУНКЦИЙ ПРОГРАММЫ ПОЛЬЗОВАТЕЛЯ **31**

Глава 4. Основные процедуры программы пользователя 31
Глава 5. Описание основных процедур 32
Процедура 1. Инициализация значений переменных *32*
Процедура 2. Проверка на достоверность кода входного сигнала АЦП. Масштабирование значений параметров, представленных аналоговыми сигналами *32*
Процедура 3. Контроль значений параметров, представленных аналоговыми и дискретными сигналами *32*
Процедура 4. Формирование признаков – результатов вычисления функций, необходимых для выполнения технологического автомата *32*
Процедура 5. Технологический автомат управления *33*
Процедура 6. Контроль состояния и управление приводами *36*
Процедура 7. Поддержка индикатора TD200 ..... *38*
Процедура 8. Поддержка обмена по сети (как правило, сеть ProfiBus) *39*
Глава 6. Особенности применения технологии в контролерах с малым объемом памяти и языками инструкций (типа STL) 41

ЗАКЛЮЧЕНИЕ **41**

Приложение 1. SWITCH-технология® 42
Приложение 2. Психология ответственности и формализация логики 43
Приложение 3. Что плохого в неавтоматном подходе к алгоритмизации управления? 45
Приложение 4. Автоматизация кодирования 48
Приложение 5. Примеры сгенерированного кода автоматов проекта 50
Автомат *ACPx* (рис. 5, 6) 50
Автомат *ACSx* (рис. 7, 8) 52
Автомат *ATWPSFW* (рис. 11, 12) 57
Автомат *ATMainWP* (рис. 13, 14) 62
Приложение 6. Примеры подпрограмм вызова автоматов 65
Вызов автомата *ACPx* 65
Вызов автомата *ACSx* 66
Вызов автомата *ATWPSFW* 68
Вызов автомата *ATMainWP* 68

Введение

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

Математическое обеспечение является основным формализованным представлением требований технического задания в части логики управления. В принципе этот вид обеспечения не требует для своей разработки ничего кроме технического задания.

Информационное обеспечение является своеобразным каталогом данных, используемых при контроле и управлении. Оно создается на основе технического задания и дополнительной информации, содержащейся в математическом обеспечении.

Программное обеспечение является реализацией всех функций, требуемых от системы управления (в том числе и основных, логика реализации которых изложена в математическом обеспечении). Большинство используемых программных переменных создается на основе материалов информационного обеспечения.

Теоретически процесс проектирования должен протекать последовательно.

Предлагаемая **МЕТОДИКА** охватывает все этапы процесса проектирования математического, информационного и программного обеспечения **локальных систем управления**.

Она предназначена для специалистов, имеющих опыт работы с **контроллерами SIMATIC S7-200**, и опирается на опыт разработки систем рассматриваемого класса.

МЕТОДИКА не является панацеей от всех бед и ошибок. Это попытка систематизации процесса проектирования от алгоритмизации до программирования.

В МЕТОДИКЕ в части алгоритмизации используется **SWITCH-технология®** (краткое описание ее положений приведено в Приложении 1, **http://is.ifmo.ru/**). Эта технология, предложенная в 1991 г. профессором А.А. Шалыто, включает, в частности, формальное программирование алгоритмов. Следствием этого является возможность автоматического преобразования алгоритма в код программы. При этом логическая часть программы не требует затрат времени и сил на кодирование (Вавилов К., Программирование за... 1 (одну) минуту, **http://is.ifmo.ru/?i0=progeny&i1=1minute**).

Часть 1. Создание математического и информационного обеспечения

Глава 1. Первые шаги

1.1. Основные термины и обозначения

Определим основные термины. **Некоторые определения являются частными и приняты исключительно для данной МЕТОДИКИ**.

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 данной МЕТОДИКЕ – это последовательность состояний оборудования по истечении определенного времени и/или для достижения определенного значения технологического параметра.

Автомат – это алгоритм, созданный на принципах **SWITCH-технологии**®. Пояснения и ссылки приведены в Приложении 1, кроме того, многое будет разъяснено ниже.

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

Алгоритм управления приводом (алгоритм нижнего уровня для 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 (реакции).

1.2. Определяем функции локальной системы управления

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

Кратко функции любой системы управления можно сформулировать как “принял – обработал – выдал”. Локальная система управления с использованием контроллера *SIMATIC S7-200* не отличается оригинальностью. В подавляющем большинстве случаев предстоит реализовать следующие функции.

- **Первичная обработка аналоговых сигналов (включая проверку на достоверность) и масштабирование значений параметров, представленных аналоговыми сигналами.**
- **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.

1.3. Начинаем разработку информационного обеспечения

Одновременно с определением функций системы возможно создать **основу** информационного обеспечения – **таблицу входных/выходных сигналов контроллера** (сюда не входят данные получаемые/передаваемые по сети). Пример такой таблицы приведен ниже (табл.1).

Таблица входных/выходных сигналов контроллера является первой из таблиц символического представления элементов памяти контроллера (битов, байтов, слов, двойных слов). Эту и последующие таблицы для удобства редактирования 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 1. Пример **таблицы входных/выходных сигналов контроллера**

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"	
		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 привода 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
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 2. Базовые алгоритмы

2.1. Краткое описание концепции выполнения программы контроллером *S7-200*

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

Связь программы с входами и выходами
Основная работа центрального процессора (CPU) контроллера S7-200 достаточно проста:
• 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.

Объяснение цикла 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.

Из документации фирмы *SIEMENS*

Основные положения, необходимые для дальнейшего проектирования:

- **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**.

2.2. Виды и типы разрабатываемых алгоритмов (автоматов) управления

В МЕТОДИКЕ описывается ТОЛЬКО **алгоритмизация и программирование задач логического управления.** Алгоритмы реализации остальных задач представляются более простыми. Примеры с комментариями приведены ниже.

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. Если совместить решение этих задач в одном алгоритме, то это “утяжелит” алгоритм. 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**.**

Необходимость такого разделения станет ясной позже. Данная МЕТОДИКА описывает определенную технологию проектирования, где все этапы взаимосвязаны, и предлагаемые решения являются по сути обязательными.

2.3. Создание базовых автоматов

Как отмечалось выше, 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).

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

2.3.1. Автомат потенциального управления нереверсивным приводом

2.3.1.1. Построение 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 в настоящей работе рекомендуется применять пакет *Visio***.

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

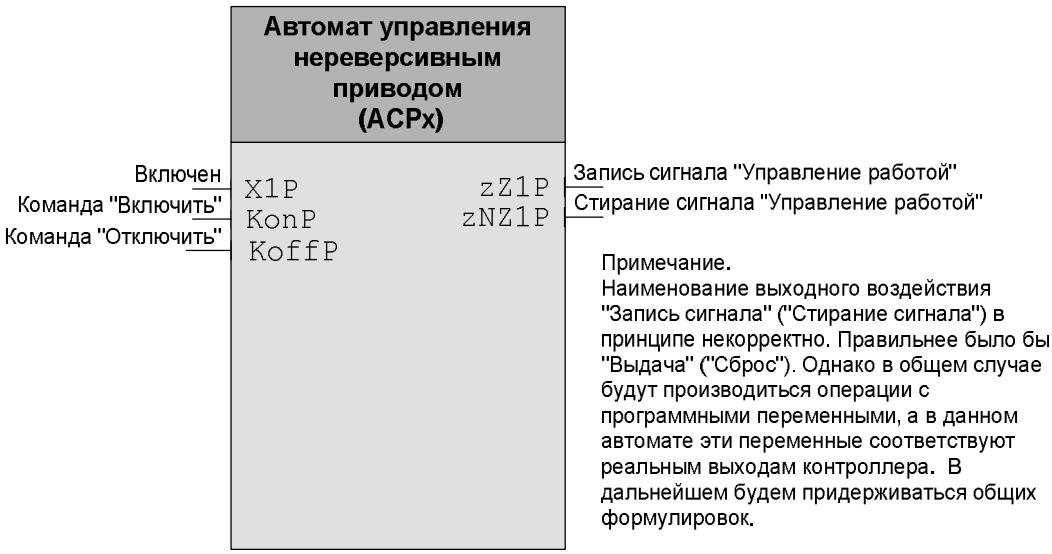


Рис. 1

Важное напоминание
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, так как чаще всего предназначен для управления несколькими аналогичными приводами и создавать отдельный автомат управления каждым из них излишне.

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.

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 одно 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, так как в противном случае значение переменной может быть “затерто” последующими процедурами.

2.3.1.2. Построение графа переходов автомата

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

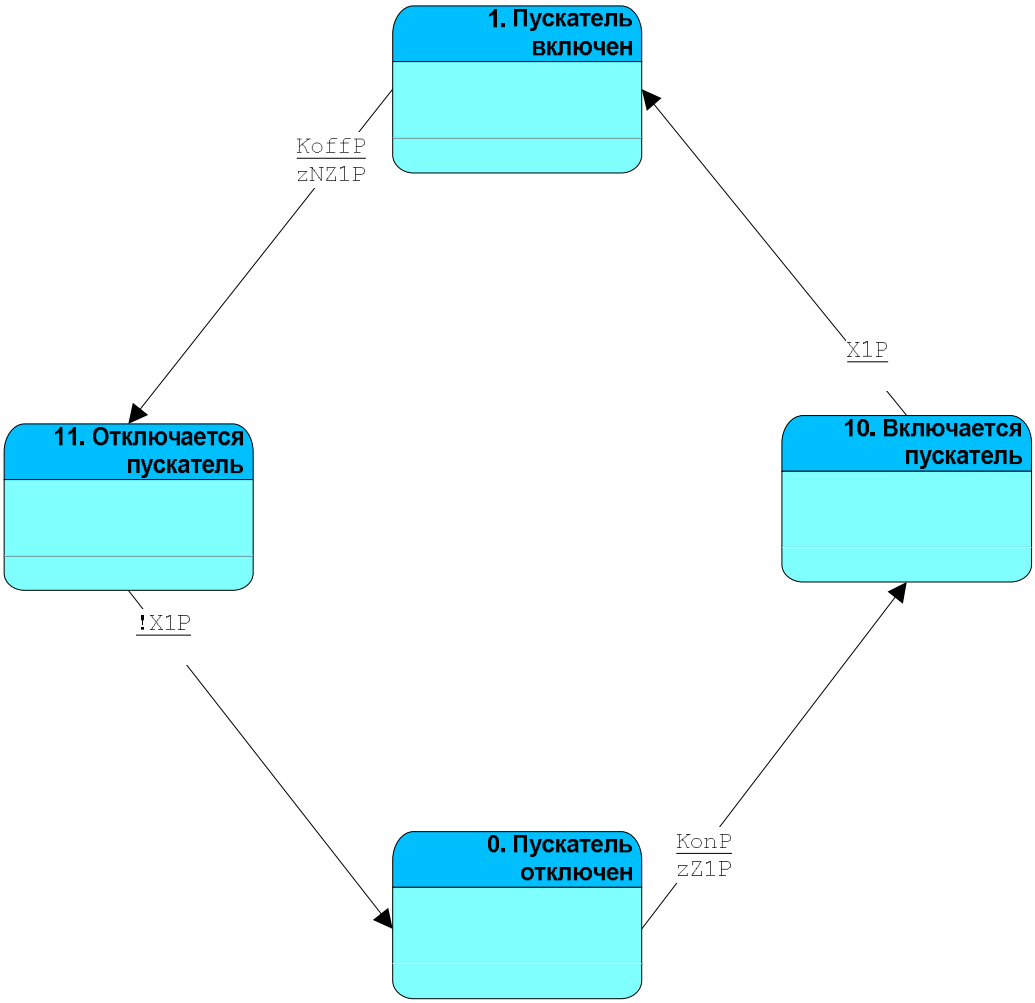


Рис. 2

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

2.3.1.3. Дополнение схемы связей и графа переходов необходимыми функциями контроля

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 интуитивно 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 интуитивно 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, так как 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). Эти дополнения выделены.

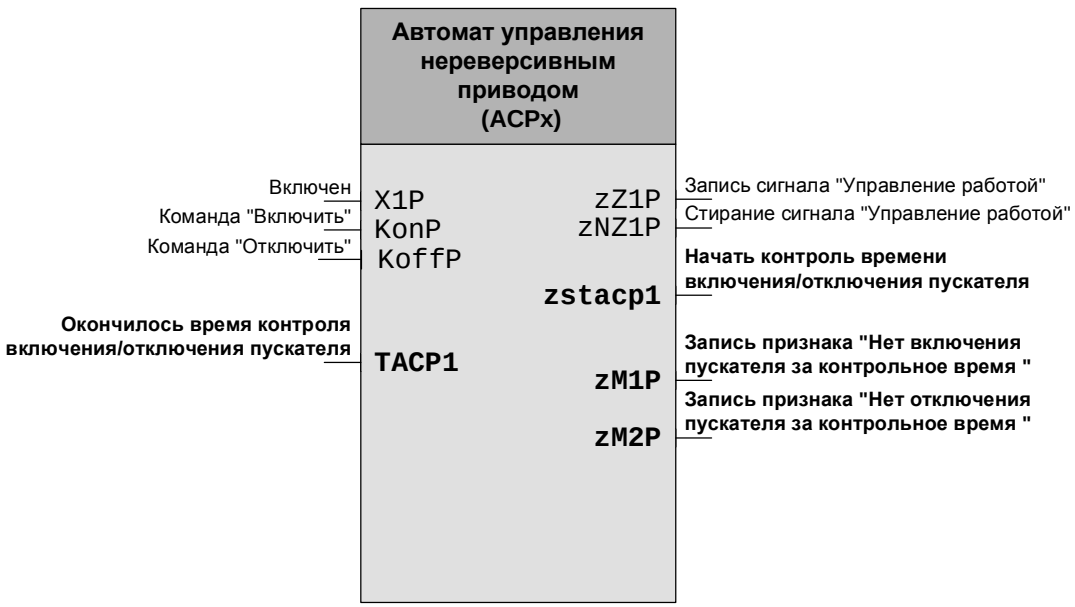


Рис.3

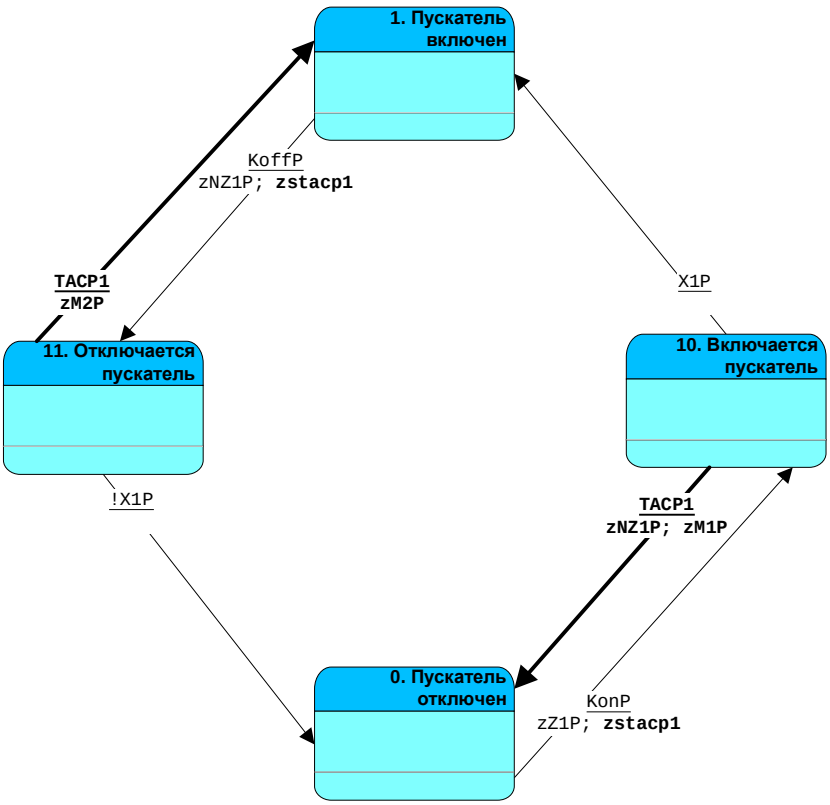


Рис. 4

В графе переходов на рис. 4 в каждом из состояний 10 и 11 переходы могут быть противоречивы – выполняться одновременно. Это может быть устранено, как показано ниже, 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 невключения, вероятность неотключения мала). При этом если пускатель не отключился, то система будет ожидать отключения и ничего более. Решение об упрощении должно быть хорошо продуманным.

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).

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.

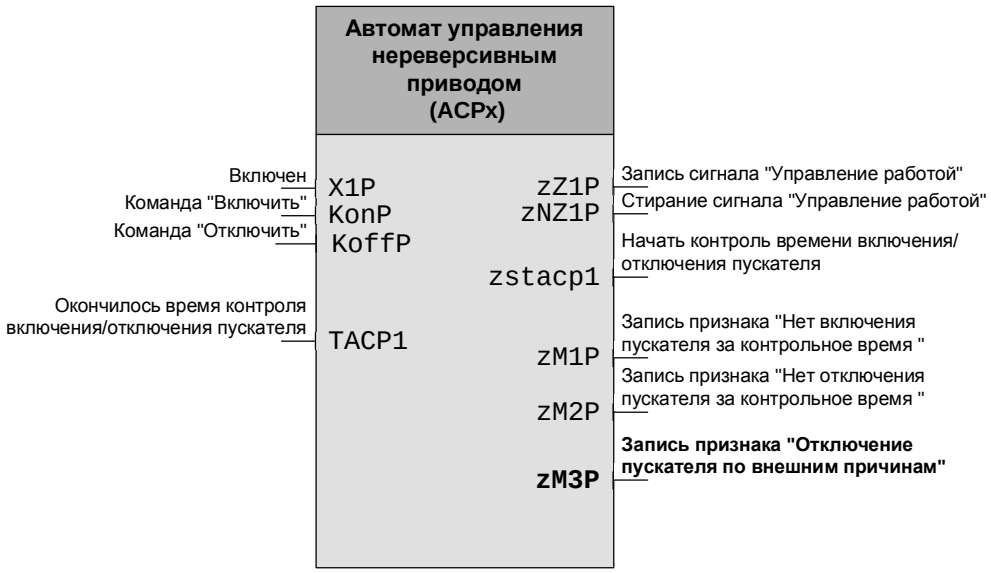


Рис. 5

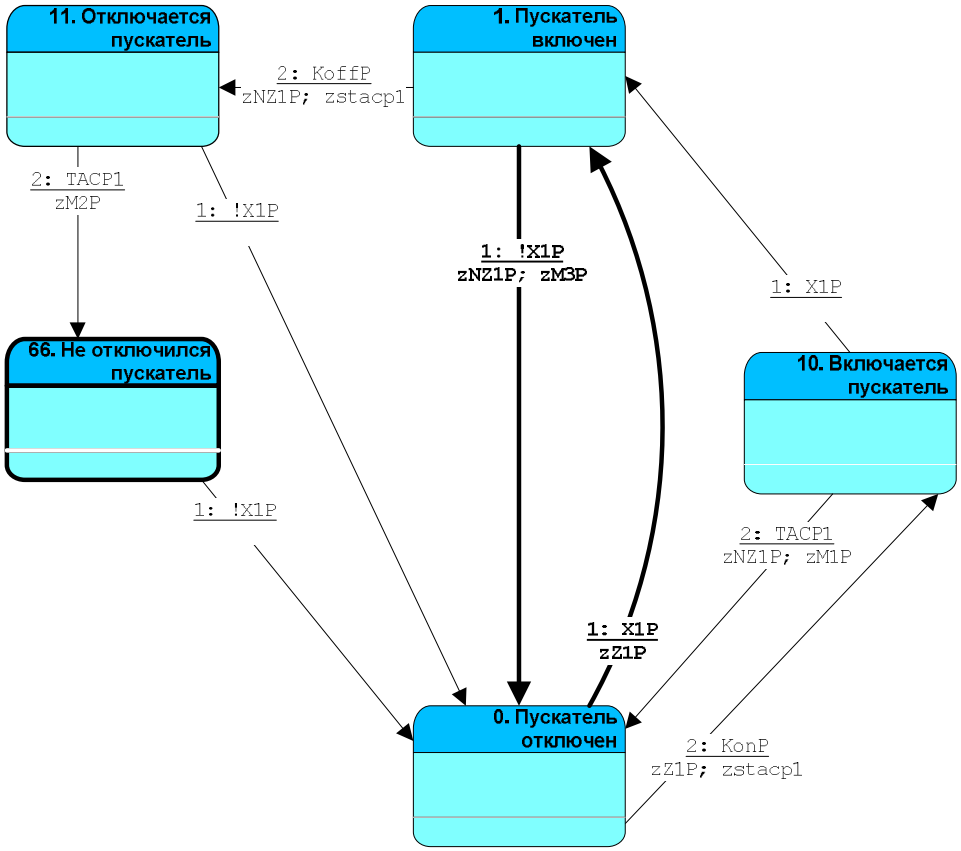


Рис. 6

Этот граф переходов отличается от предыдущего (рис. 4) еще и тем, что в нем используется 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, так как произойдет изменение состояния.

Отметим важность перехода с приоритетом 1 из состояния 0 в состояние1. 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 (естественно, если привод управляется потенциально).

2.3.2. Автомат потенциального управления реверсивным приводом

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

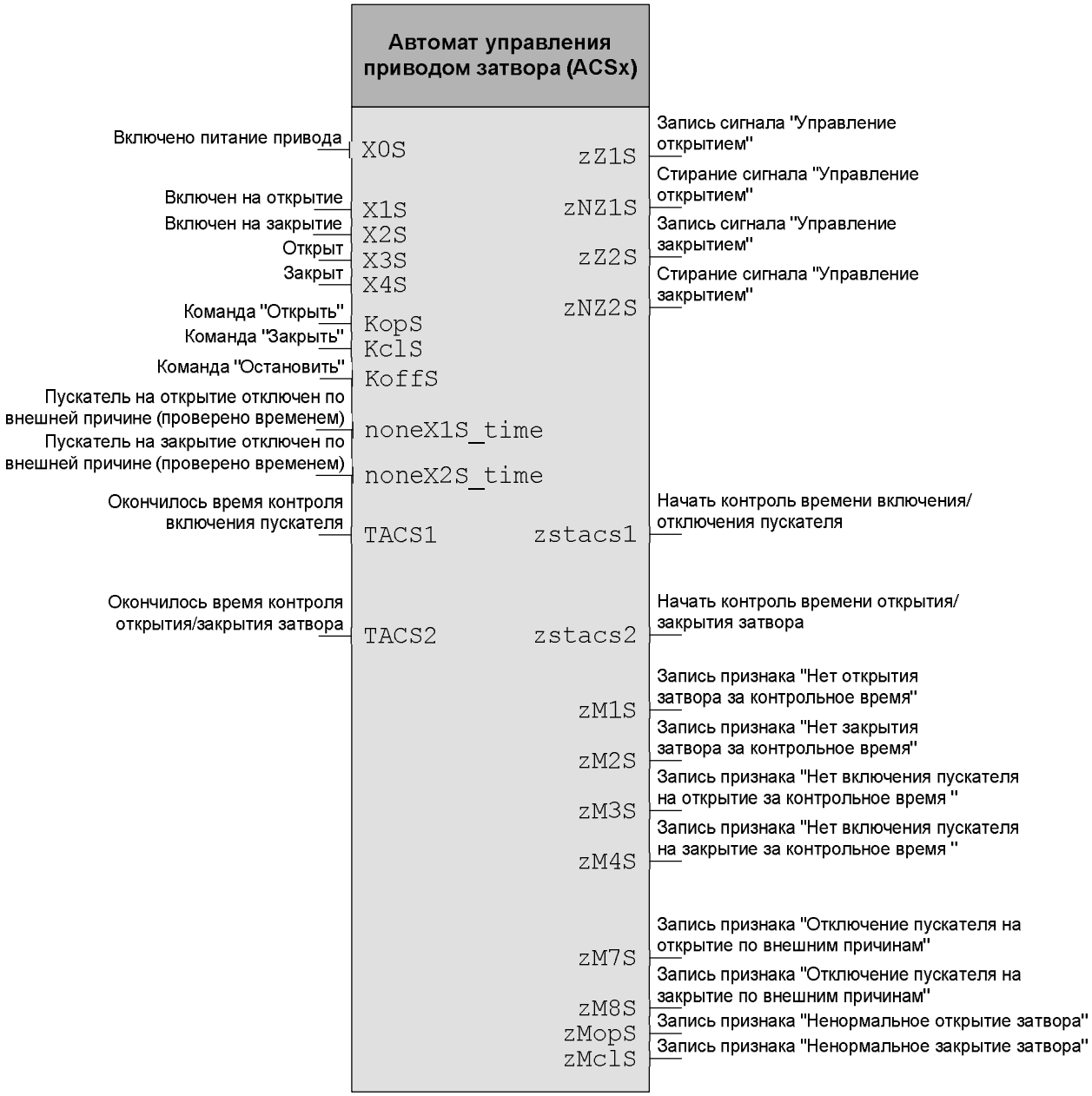


Рис. 7

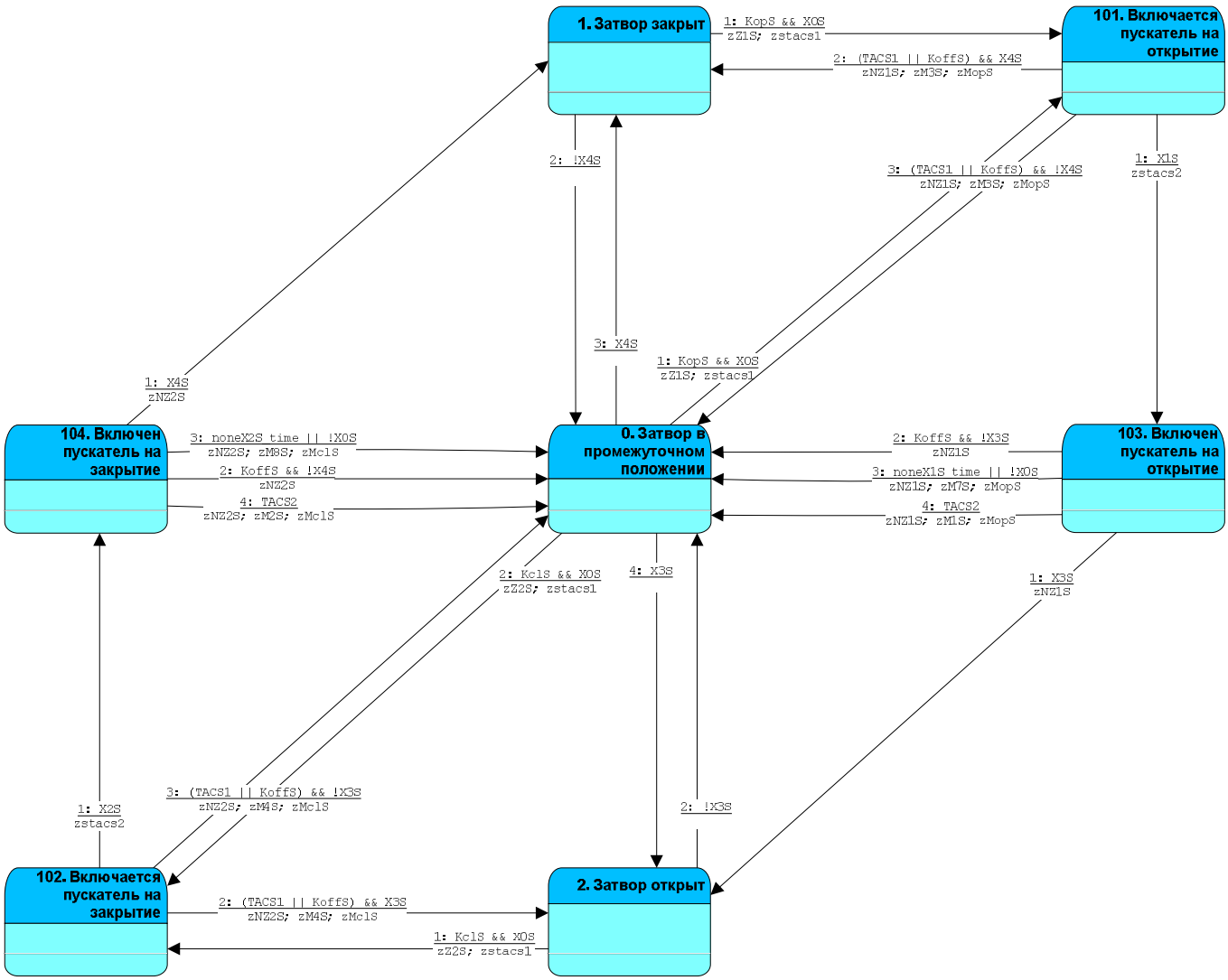


Рис. 8

Необходимо отметить, что создание автомата – процесс творческий. Поэтому автоматы, которые представлены в МЕТОДИКЕ – э**то ОДНА из возможных реализаций управления.**

Поясним некоторые 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. Естественно, 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 пришлось подстраховаться от одной неприятной ситуации. Дело в том, что 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 временны*е* (от слова ВРЕМЯ) характеристики срабатывания реле, на которых построена схема. Чем больше реле содержит цепь, тем позже сработает контакт, сигнализируя о каком-то событии. Конечно, тут же сразу могут последовать

возражения типа “Современные реле срабатывают практически мгновенно и не стоит об этом думать”. И все-таки…

Сигналы, поступающие в контроллер, являются результатом замыкания/размыкания контактов определенных реле схемы. Очень часто это промежуточные реле, применяемые только для передачи сигналов в контроллер. Получается, что сигнал от конечного выключателя (допустим, “Затвор закрыт”), принимаемый контроллером, поступает не непосредственно (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 неприятна 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 все-таки является нормальной, штатной работой схемы управления. Если бы это была только информационная “оплошность” системы, то особых проблем не было бы, но здесь другая ситуация – практически всегда при ненормальной работе оборудования, а именно так система будет “рассматривать” данную ситуацию, последует соответствующая реакция программы по выходу из создавшегося “плохого” 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 есть 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. **Отметим, что 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 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).

2.4. Информационное обеспечение базовых алгоритмов

2.4.1. Таблица символов автомата потенциального управления нереверсивным приводом

Как уже отмечалось, автомат потенциального управления нереверсивным приводом является подпрограммой и оперирует формальными параметрами (табл. 2).

Таблица 2. Пример **таблицы формальных параметров** автомата потенциального управления нереверсивным приводом

		ФОРМАЛЬНЫЕ ПАРАМЕТРЫ АВТОМАТА УПРАВЛЕНИЯ НАСОСОМ (ACPx)
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"

Приведем некоторые пояснения к таблице.

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.

2.4.2. Таблица символов автомата потенциального управления реверсивным приводом

Табл. 3 содержит формальные параметры автомата для рассматриваемого привода.

Таблица 3. Пример **таблицы формальных параметров** автомата потенциального управления реверсивным приводом

		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
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"

2.4.3. Таблица символов фактических параметров, используемых/формируемых при вызове автомата потенциального управления нереверсивным приводом

Табл. 4 содержит символы фактических параметров для рассматриваемого привода.

Таблица 4. Пример **таблицы фактических параметров** при вызове автомата потенциального управления нереверсивным приводом

		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).
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)
		ПРИЗНАКИ СООБЩЕНИЙ, ФОРМИРУЕМЫЕ В ТЕХНОЛОГИЧЕСКОМ АВТОМАТЕ
MNConP_WP	V35.1	Нет условий для автоматического пуска привода промывного насоса

Приведем пояснения к табл. 4.

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 они все 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.

2.4.4. Таблица символов фактических параметров, используемых/формируемых при вызове автомата потенциального управления реверсивным приводом

Табл. 5 содержит символы фактических параметров для рассматриваемого привода.

Таблица 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.
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
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	V22.0	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)
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)

Приведем пояснения к табл. 5.

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

Глава 3. Технологические алгоритмы

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” (сначала самостоятельно, а потом обязательно согласовать с Заказчиком) реакцию системы при ненормальной ситуации.

3.1. Создание технологического автомата управления насосным агрегатом

3.1.1. Типичное техническое задание

Приведем пример фрагмента технического задания – задания на управление насосным агрегатом.

> 3.7. Режимы работы шкафа автоматического управления насосным агрегатом.
> 3.7.1. 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

Приведем также дополнения к заданию (после первого обсуждения с Заказчиком):

- 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.

3.1.3. Нормальный ход процесса

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

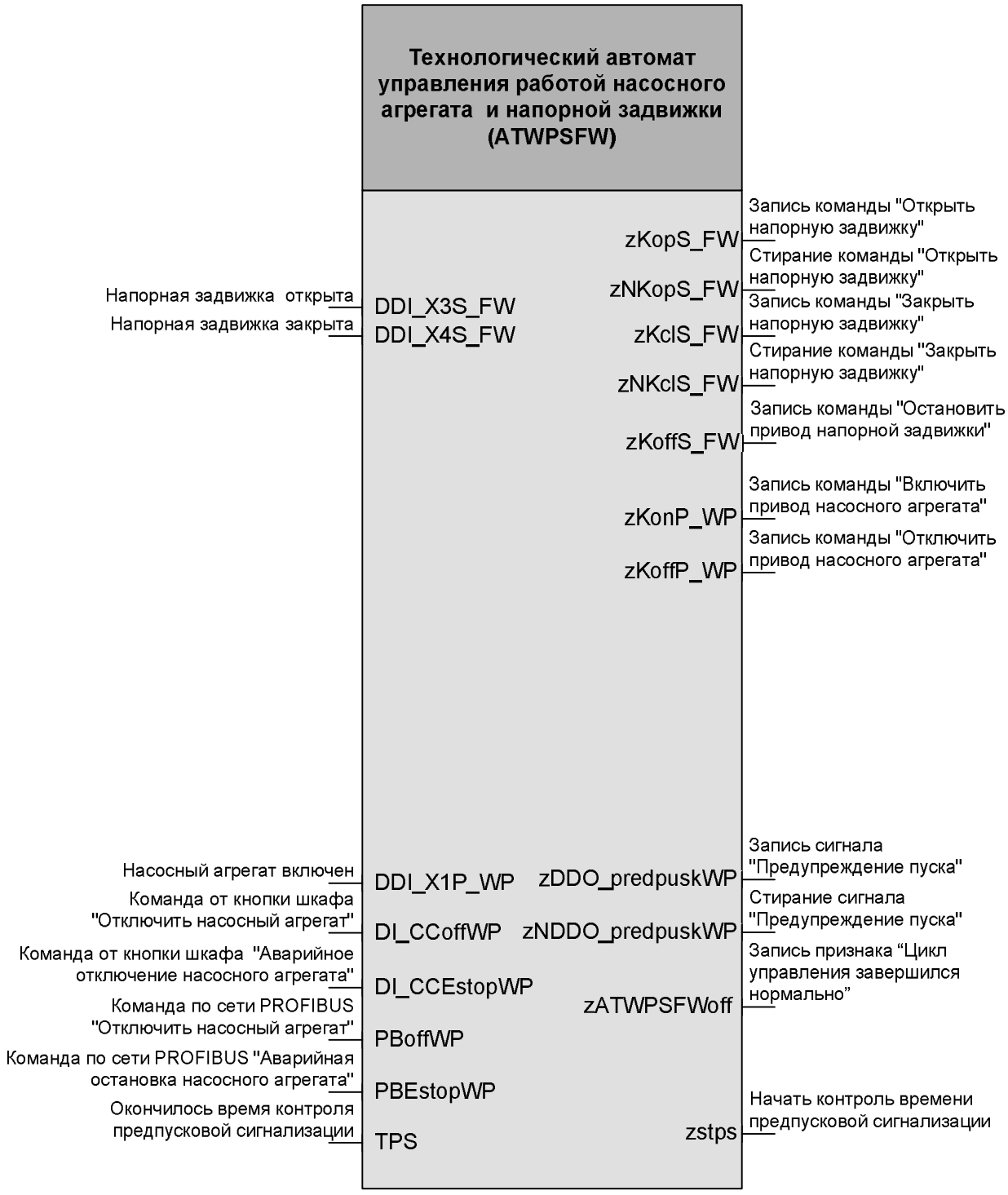


Рис. 9

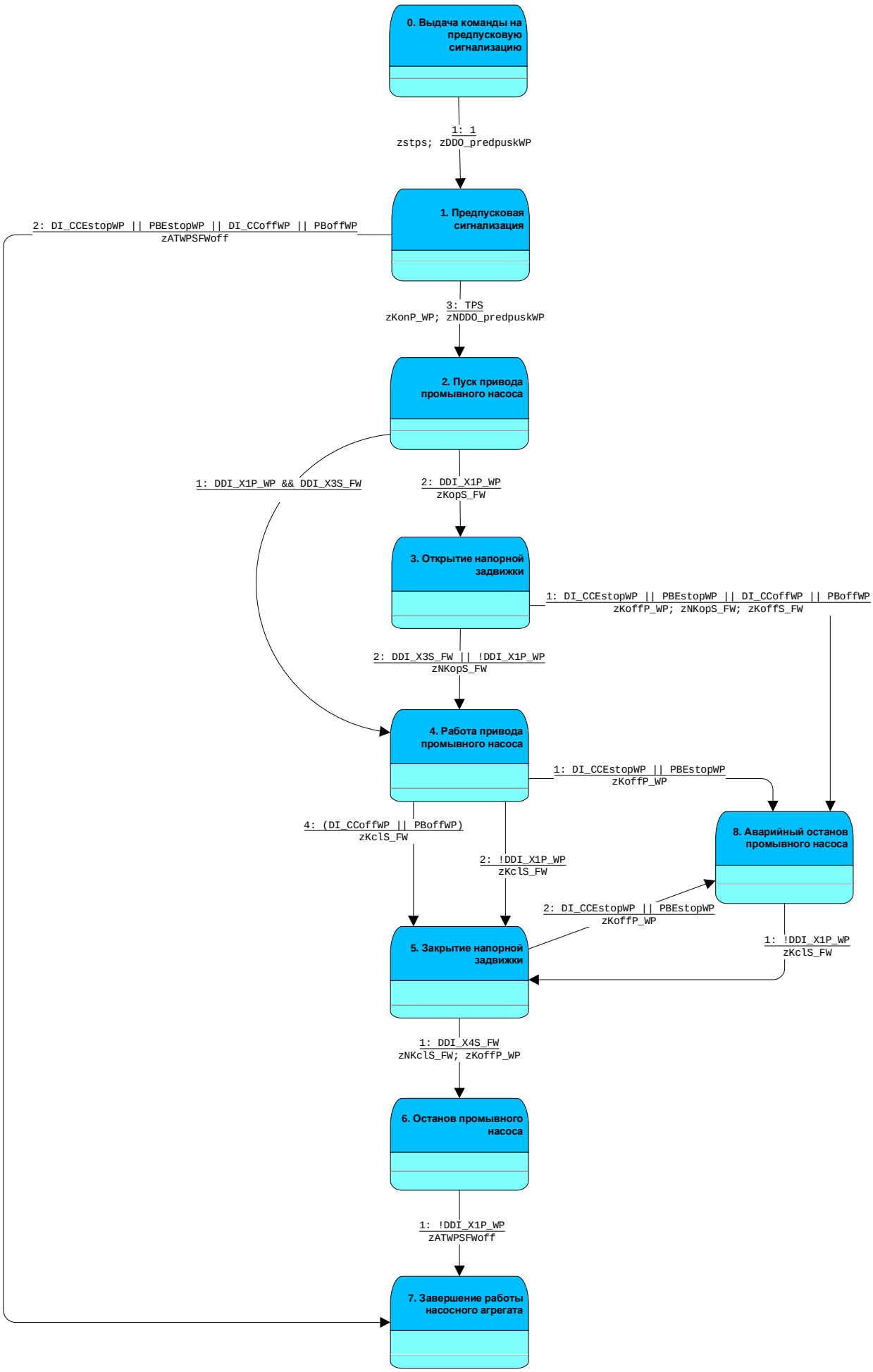

0. Выдача команды на предпусковую сигнализацию
1: 1
zstps; zDDO_predpuskWP
1. Предпусковая сигнализация
2: DI_CCEstopWP || PBEstopWP || DI_CCoffWP || PBoffWP
zATWPSFWoff
3: TPS
zKonP_WP; zNDDO_predpuskWP
2. Пуск привода промывного насоса
1: DDI_X1P_WP && DDI_X3S_FW
2: DDI_X1P_WP
zKopS_FW
3. Открытие напорной задвижки
1: DI_CCEstopWP || PBEstopWP || DI_CCoffWP || PBoffWP
zKoffP_WP; zNKopS_FW; zKoffS_FW
2: DDI_X3S_FW || !DDI_X1P_WP
zNKopS_FW
4. Работа привода промывного насоса
1: DI_CCEstopWP || PBEstopWP
zKoffP_WP
4: (DI_CCoffWP || PBoffWP)
zKclS_FW
2: !DDI_X1P_WP
zKclS_FW
8. Аварийный останов промывного насоса
2: DI_CCEstopWP || PBEstopWP
zKoffP_WP
1: !DDI_X1P_WP
zKclS_FW
5. Закрытие напорной задвижки
1: DDI_X4S_FW
zNKclS_FW; zKoffP_WP
6. Останов промывного насоса
1: !DDI_X1P_WP
zATWPSFWoff
7. Завершение работы насосного агрегата

Рис. 10

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 особое 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**, так как при этом она просто не будет получать текущую информацию об объекте управления.

Из изложенного следует, что алгоритм должен учитывать эту особенность выполнения программы контроллера.

Автор МЕТОДИКИ убежден, что алгоритм необходимо создавать в виде наиболее близком к реализации. Гораздо проще разобраться в логике, изображенной в виде диаграммы на бумаге, чем изучать программный код, особенно такой как у контроллера *S7-200*.

Алгоритм, созданный на принципах **SWITCH-технологии®**, и что самое важное, **и программа, созданная на основе этого алгоритма**, полностью отвечают вышеизложенным требованиям. Кроме этого, такой алгоритм весьма компактно изображается.

3.1.3. Дополняем автомат "нехорошими" событиями

На рис. 11, 12 представлены схема связей и граф переходов рассматриваемого автомата, дополненный событиями, связанными с неготовностью или ненормальной работой оборудования (дополнения выделены жирным шрифтом). **Рекомендуется перед реализацией согласовать с Заказчиком действия системы при ненормальной работе оборудования.**

Приведем пояснения к этим диаграммам.

Опыт наладки показал, что почти все из того, что дополнительно контролировалось, было использовано (вернее сказать, произошло), а контролировался практически каждый "чих" в работе оборудования.

При этом хотелось бы 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 очень помогает при программировании – не надо еще раз 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 будет 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).

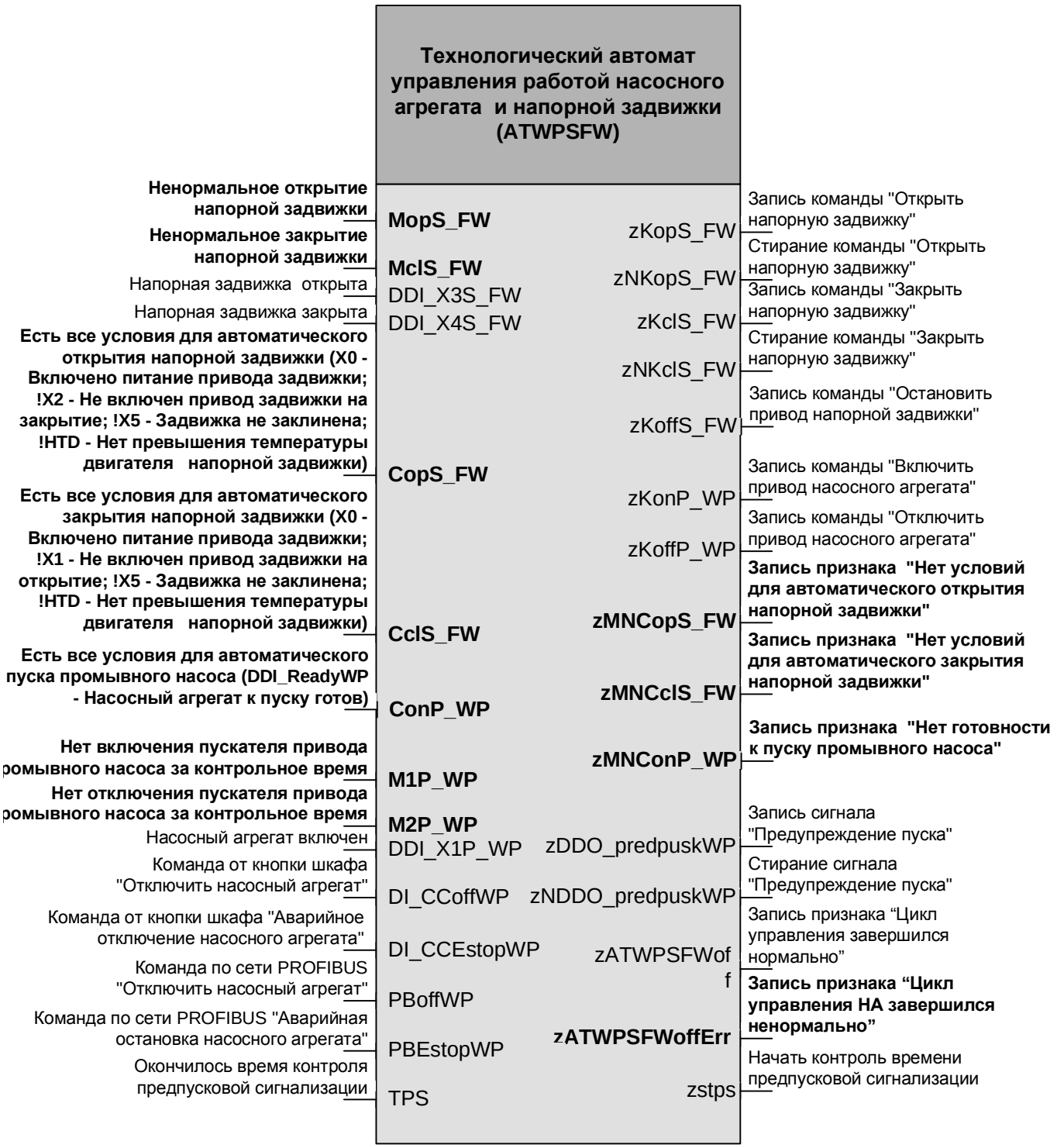

Технологический автомат управления работой насосного агрегата и напорной задвижки (ATWPSFW)
Ненормальное открытие напорной задвижки
MopS_FW
Ненормальное закрытие напорной задвижки
MclS_FW
Напорная задвижка открыта
DDI_X3S_FW
Напорная задвижка закрыта
DDI_X4S_FW
Есть все условия для автоматического открытия 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)
CopS_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)
CclS_FW
Есть все условия для автоматического пуска промывного насоса (DDI_ReadyWP - Насосный агрегат к пуску готов)
ConP_WP
Нет включения пускателя привода промывного насоса за контрольное время
M1P_WP
Нет отключения пускателя привода промывного насоса за контрольное время
M2P_WP
Насосный агрегат включен
DDI_X1P_WP
Команда от кнопки шкафа "Отключить насосный агрегат"
DI_CCoffWP
Команда от кнопки шкафа "Аварийное отключение насосного агрегата"
DI_CCEstopWP
Команда по сети PROFIBUS "Отключить насосный агрегат"
PBoffWP
Команда по сети PROFIBUS "Аварийная остановка насосного агрегата"
PBEstopWP
Окончилось время контроля предпусковой сигнализации
TPS
zKopS_FW
Запись команды "Открыть напорную задвижку"
zNKopS_FW
Стирание команды "Открыть напорную задвижку"
zKclS_FW
Запись команды "Закрыть напорную задвижку"
zNKclS_FW
Стирание команды "Закрыть напорную задвижку"
zKoffS_FW
Запись команды "Остановить привод напорной задвижки"
zKonP_WP
Запись команды "Включить привод насосного агрегата"
zKoffP_WP
Запись команды "Отключить привод насосного агрегата"
zMNCopS_FW
Запись признака "Нет условий для автоматического открытия напорной задвижки"
zMNCclS_FW
Запись признака "Нет условий для автоматического закрытия напорной задвижки"
zMNConP_WP
Запись признака "Нет готовности к пуску промывного насоса"
zDDO_predpuskWP
Запись сигнала "Предупреждение пуска"
zNDDO_predpuskWP
Стирание сигнала "Предупреждение пуска"
zATWPSFWoff
Запись признака “Цикл управления завершился нормально”
zATWPSFWoffErr
Запись признака “Цикл управления НА завершился ненормально”
zstps
Начать контроль времени предпусковой сигнализации

Рис. 11

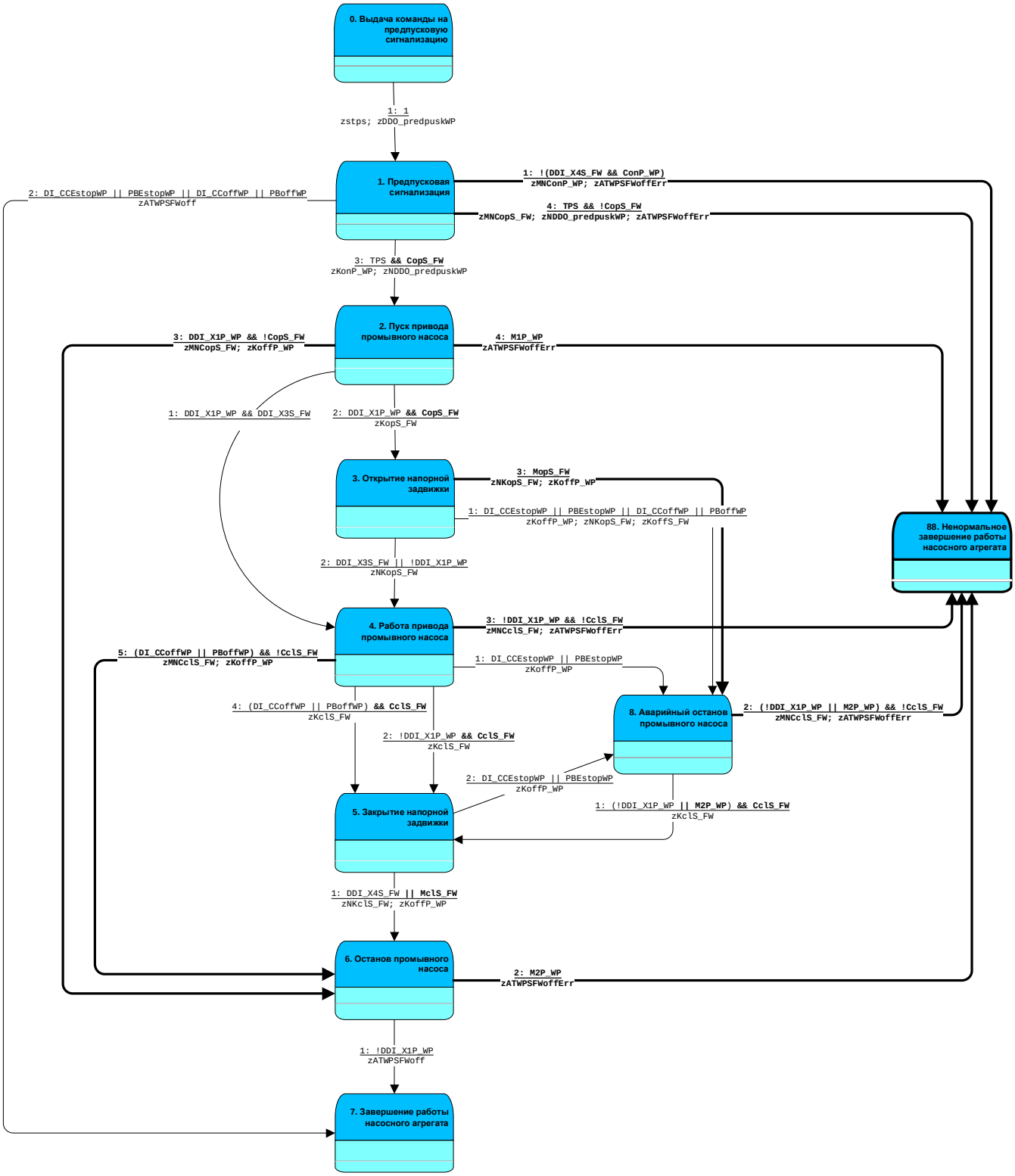

0. Выдача команды на предпусковую сигнализацию
1: 1
zstps; zDDO_predpuskWP
1. Предпусковая сигнализация
2: DI_CCEstopWP || PBEstopWP || DI_CCoffWP || PBoffWP
zATWPSFWoff
1: !(DDI_X4S_FW && ConP_WP)
zMNConP_WP; zATWPSFWoffErr
4: TPS && !CopS_FW
zMNCopS_FW; zNDDO_predpuskWP; zATWPSFWoffErr
3: TPS && CopS_FW
zKonP_WP; zNDDO_predpuskWP
2. Пуск привода промывного насоса
3: DDI_X1P_WP && !CopS_FW
zMNCopS_FW; zKoffP_WP
4: M1P_WP
zATWPSFWoffErr
1: DDI_X1P_WP && DDI_X3S_FW
2: DDI_X1P_WP && CopS_FW
zKopS_FW
3. Открытие напорной задвижки
3: MopS_FW
zNKopS_FW; zKoffP_WP
1: DI_CCEstopWP || PBEstopWP || DI_CCoffWP || PBoffWP
zKoffP_WP; zNKopS_FW; zKoffS_FW
88. Ненормальное завершение работы насосного агрегата
2: DDI_X3S_FW || !DDI_X1P_WP
zNKopS_FW
4. Работа привода промывного насоса
3: !DDI_X1P_WP && !CclS_FW
zMNCclS_FW; zATWPSFWoffErr
5: (DI_CCoffWP || PBoffWP) && !CclS_FW
zMNCclS_FW; zKoffP_WP
1: DI_CCEstopWP || PBEstopWP
zKoffP_WP
8. Аварийный останов промывного насоса
2: (!DDI_X1P_WP || M2P_WP) && !CclS_FW
zMNCclS_FW; zATWPSFWoffErr
4: (DI_CCoffWP || PBoffWP) && CclS_FW
zKclS_FW
2: !DDI_X1P_WP && CclS_FW
zKclS_FW
2: DI_CCEstopWP || PBEstopWP
zKoffP_WP
1: (!DDI_X1P_WP || M2P_WP) && CclS_FW
zKclS_FW
5. Закрытие напорной задвижки
1: DDI_X4S_FW || MclS_FW
zNKclS_FW; zKoffP_WP
6. Останов промывного насоса
2: M2P_WP
zATWPSFWoffErr
1: !DDI_X1P_WP
zATWPSFWoff
7. Завершение работы насосного агрегата

Рис. 12

3.1.4. «Главный» технологический автомат

Теперь осталось совсем немного – описать **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.

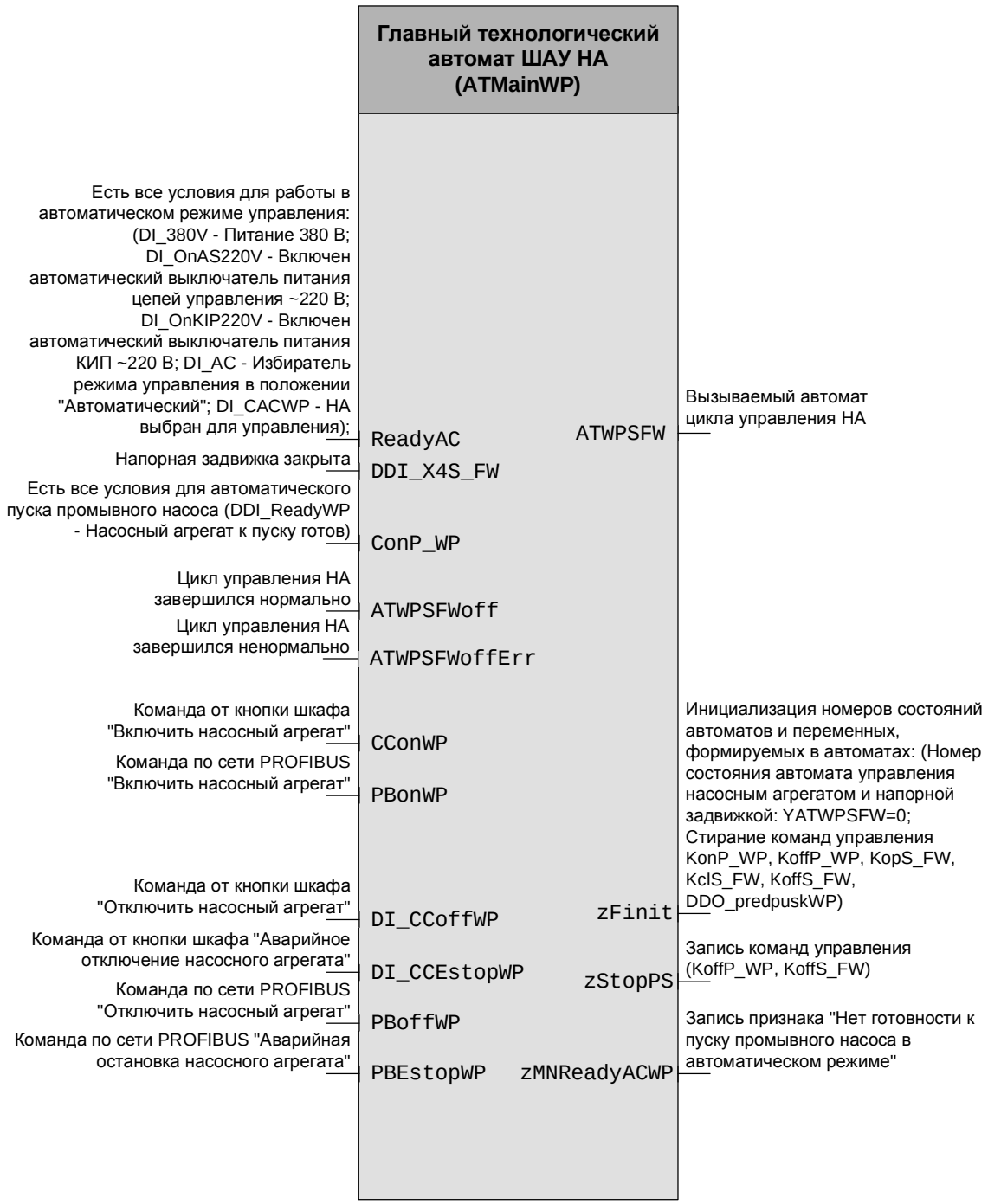


Рис. 13

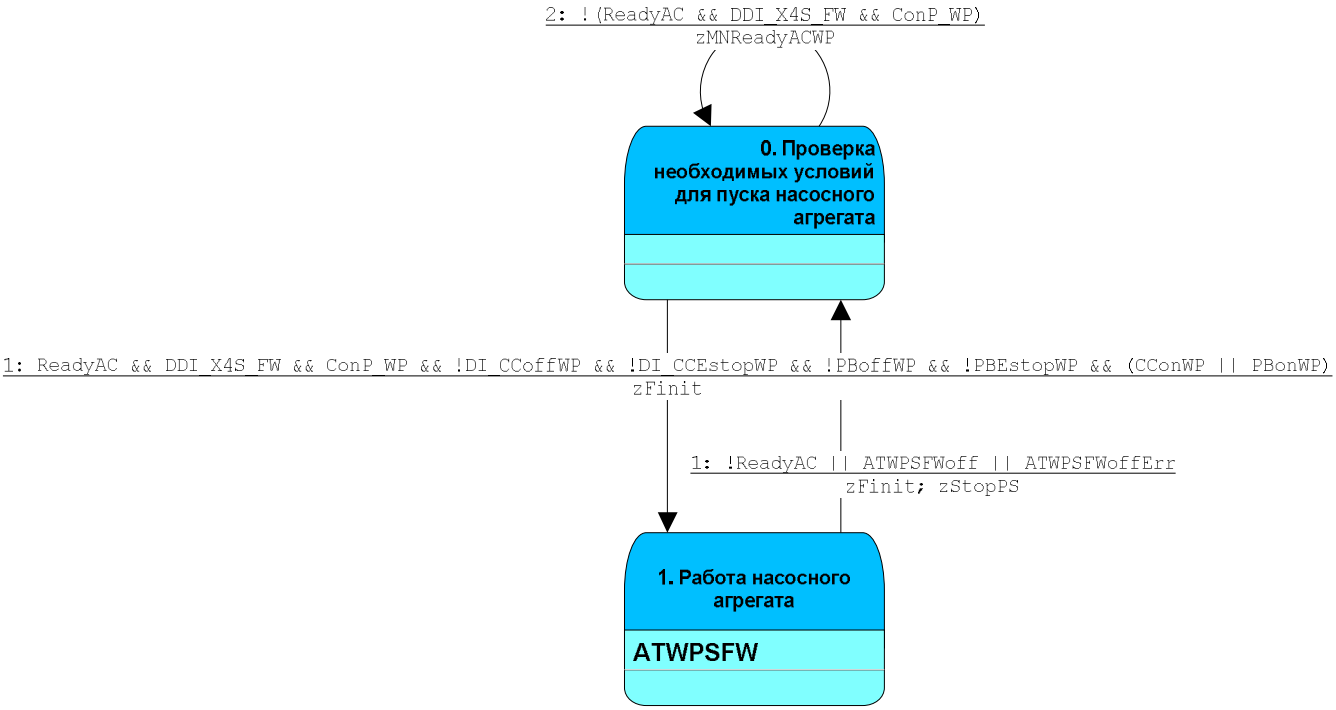


Рис. 14

Приведем пояснения к рис. 13, 14.

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

В проекте, откуда были взяты этот и другие примеры, предусматривается запрет прохождения 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.

3.2. Таблицы символов технологических автоматов

Табл. 6 содержит фактические параметры для рассматриваемого технологического автомата.

Таблица 6. Пример **таблицы 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 НЕПОСРЕДСТВЕННО 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
ATWPSFWoffErr	V17.1	Цикл управления НА завершился ненормально
Stps	V17.2	Начать контроль времени предпусковой сигнализации
		Признаки вызова ПОДАВТОМАТОВ

В табл. 6 неожиданно мало данных. Однако это нормально, так как 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.

Табл. 7 содержит фактические параметры для главного технологического автомата.

Таблица 7. Пример **таблицы 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).**
		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

Часть 2. Рекомендации по программированию основных функций программы пользователя

Глава 4. 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.

Глава 5. 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 тоже 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. Как отмечалось выше, 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 единственный вопрос – зачем нужна отдельная подпрограмма? Ведь можно подготавливать данные в процессе вызова автомата, передавая их в качестве параметров. Ответ достаточно прост – чаще всего эти данные используются одновременно в разных технологических автоматах, и поэтому незачем вычислять их заново. Впрочем, это не аксиома. Но в данной МЕТОДИКЕ передача параметров в подпрограмму не выполняется.

Процедура 5. Технологический автомат управления

Эта и следующая процедуры предполагает **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**, что и было реализовано (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`. Это не должно никого печалить, так как 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 **НАСТОЯТЕЛЬНО РЕКОМЕНДУЕТСЯ автоматизировать построение кода программы по графу переходов**, благо 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, сгенерировать новый код – и **ВСЕ**!

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
```

Несколько слов 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.

В данной МЕТОДИКЕ 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, как показал опыт, 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. Тоже 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 следующее:

- 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 (как правило, сеть *ProfiBus*)

Предлагается следующая концепция (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 6. Особенности применения технологии в контролерах 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 нереверсивных 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.

Попробуйте организовать такую наблюдаемость и контроль выполнения кода программы (уже программы, а не алгоритма), применяя другие подходы!

ЗАКЛЮЧЕНИЕ

Данная технология и МЕТОДИКА, основанная на ее применении, позволяют эффективно разрабатывать и реализовывать алгоритмы логического управления **локальным оборудованием** с использованием **контроллера *S7-200***. Естественно, **это может быть и любой другой контроллер**, позволяющий писать программы **в текстовом виде**.

Предлагаемая МЕТОДИКА является не столько пособием по алгоритмизации и программированию задач логического управления, сколько **руководством по ОРГАНИЗАЦИИ ПРОЕКТИРОВАНИЯ**.

Приложение 1. SWITCH-технология®

SWITCH-технология® предложена А.А. Шалыто **в 1991 г.** (http://is.ifmo.ru/works/switch_prr/). Эта 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. И не только из-за упрощения процесса программирования, но и в большей степени благодаря предлагаемому подходу к алгоритмизации.

Подробные материалы по автоматному программированию размещены на сайте **http://is.ifmo.ru**.

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).

В данной МЕТОДИКЕ 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 в данной МЕТОДИКЕ для обозначения входных и выходных воздействий применяются не символические обозначения, а идентификаторы.

Приложение 2. Психология ответственности и формализация логики

"... **ФОРМАЛИЗАЦИЯ**..., осуществляемая на базе определенных абстракций, идеализаций и искусственных символических языков, используется прежде всего в математике ... Ф. предполагает усиление роли **формальной логики**... Ф. позволяет систематизировать, уточнить и методологически прояснить содержание теории, выяснить характер взаимосвязи между собой различных ее положений, выявить и сформулировать еще не решенные проблемы...
... ФОРМАЛЬНАЯ ЛОГИКА, наука о мышлении, предметом которой является исследование умозаключений и доказательств с точки зрения их формы и в отвлечении от их конкретного содержания. Ф. л. – базисная наука. Ее идеи и методы используются как в повседневной практике, например в качестве средства предотвращения логических ошибок, так и в особенности в теории для логического анализа научного знания... "

Большая советская энциклопедия

Любой серьезный проект (подчас не столько сложный, сколько грозящий возможными негативными последствиями от некорректного проектирования, реализации или простого невыполнения к сроку) требует **ответственных исполнителей**. Автор МЕТОДИКИ вкладывает в это понятие не просто **готовность** человека к добросовестному выполнению своих производственных обязанностей. Это конечно необходимо, но не достаточно. Гораздо важнее представляется **способность и готовность к формализации логики поставленной задачи** во всем диапазоне от осмысления технического задания до тестирования программной реализации. Формализация помогает исполнителю не только корректно и своевременно реализовать необходимую функцию, но и дает **возможность легко проконтролировать выбранный путь (алгоритм) реализации**.

Формализации боятся и, что особенно непонятно, противятся некоторые исполнители. Чаще всего из-за нежелания или кажущейся сложности изложения (при этом они могут тут же начать программную реализацию, что не вызывает такого трепета) или, наоборот, из-за кажущейся легкости реализации, а то и из-за эгоистических побуждений – как бы никто не узнал тайну их “супермыслей”.

С другой стороны, формализация тоже может быть недостаточной и неэффективной, ибо существующие методы описания логики решения (алгоритмы в виде схем алгоритмов или

логических схем на элементах И, ИЛИ, НЕ) как бы “оторваны” от методов конкретной реализации (в частности, в контроллерах *S7-200*).

Наверное давней мечтой всех постановщиков задач и разработчиков программного обеспечения является полное соответствие задуманного решения задачи (алгоритма решения) и программной реализации этого решения. Однако обычно что-то не склеивается у постановщиков и программистов. 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, мягко говоря, 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. Что плохого в неавтоматном подходе 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 (или предложенный заказчиком алгоритм решения задачи) легко изображается в виде схемы алгоритма, представленной на рис. 15.

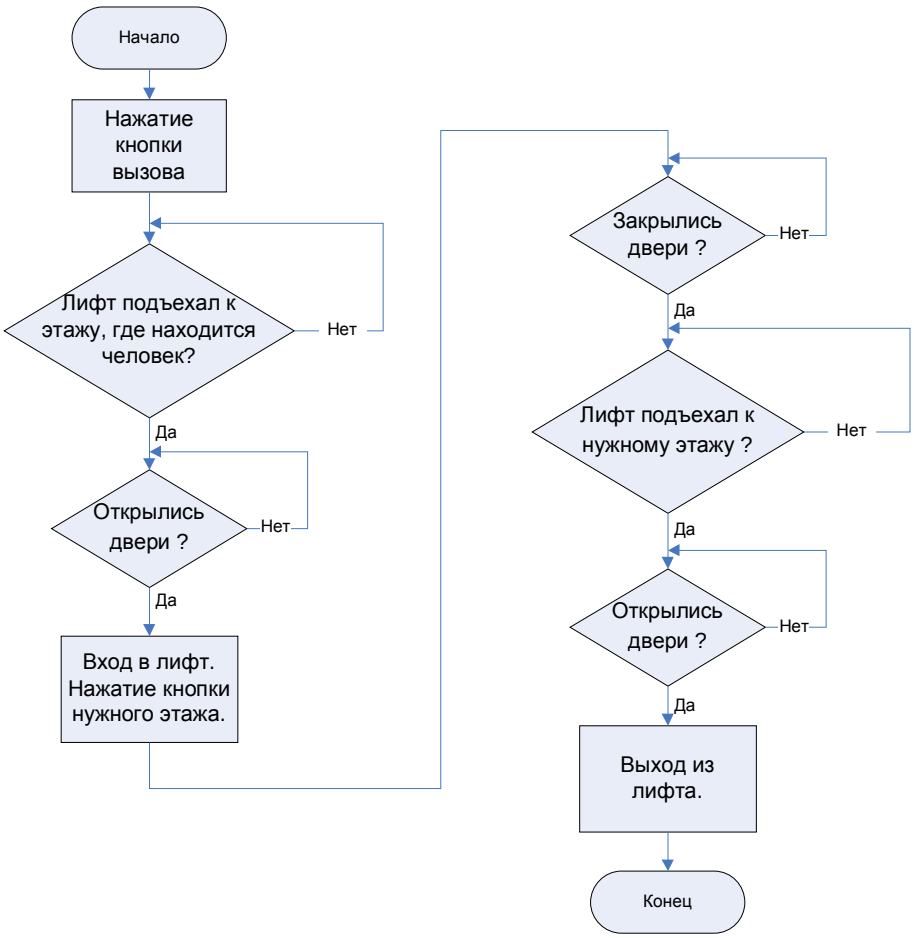


Рис. 15

Это плохое решение. Поясним это.

Достаточно вспомнить концепцию выполнения цикла CPU *S7-200*, 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.

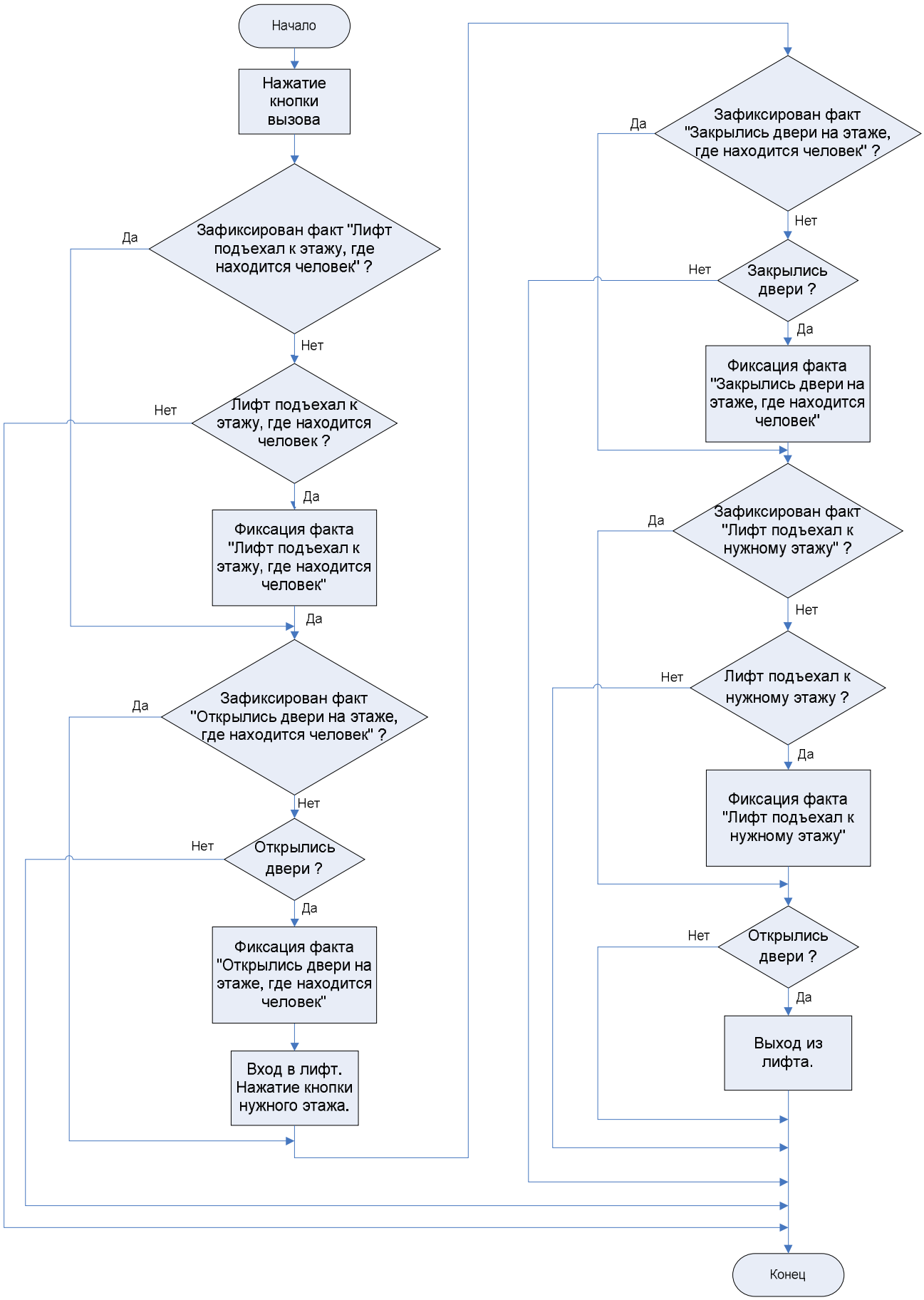

Начало
Нажатие кнопки вызова
Зафиксирован факт "Лифт подъехал к этажу, где находится человек" ?
Да
Нет
Лифт подъехал к этажу, где находится человек ?
Нет
Да
Фиксация факта "Лифт подъехал к этажу, где находится человек"
Да
Зафиксирован факт "Открылись двери на этаже, где находится человек" ?
Да
Нет
Открылись двери ?
Нет
Да
Фиксация факта "Открылись двери на этаже, где находится человек"
Вход в лифт. Нажатие кнопки нужного этажа.
Зафиксирован факт "Закрылись двери на этаже, где находится человек" ?
Да
Нет
Закрылись двери ?
Нет
Да
Фиксация факта "Закрылись двери на этаже, где находится человек"
Зафиксирован факт "Лифт подъехал к нужному этажу" ?
Да
Нет
Лифт подъехал к нужному этажу ?
Нет
Да
Фиксация факта "Лифт подъехал к нужному этажу"
Открылись двери ?
Нет
Да
Выход из лифта.
Конец

Рис. 16

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

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.

А вот теперь начинается самое интересное.

В техническом задании практически никогда не описываются временны*е* (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.

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 (Шалыто А.А. *SWITCH*-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука,1998. – 628 с. http://is.ifmo.ru/books/switch/1).

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 *Cu* (*Visio2Switch*). 2002, http://is.ifmo.ru/progeny/visio2switch). После этой публикации автор решил перейти на этот более удобным редактор (***Visio***).

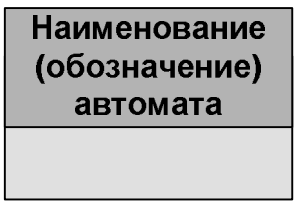
Разработка велась в среде графического программирования *LabVIEW 6.1*, так как автор МЕТОДИКИ не является профессиональным (квалифицированным) программистом, и не владеет такими средствами как *Visual C* или *Visual Basic*. Однако это обстоятельство нисколько не повлияло на результат, ибо пакет *LabVIEW* 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.).

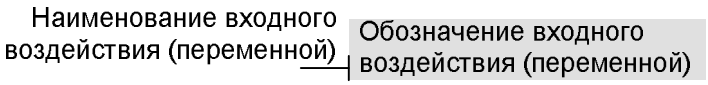
Объекты **masters** легко создаются вручную, *Visio* содержит в себе соответствующий редактор. Творчески переработав и дополнив сделанное Головешиным А., автор МЕТОДИКИ сформировал следующий необходимый набор объектов.

Объекты схемы связей:

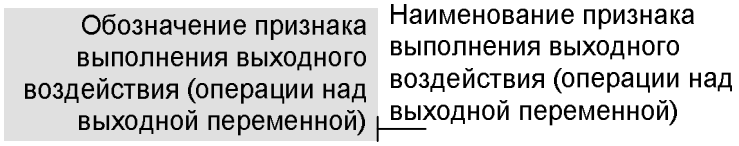
Объект “Наименование и обозначение автомата”



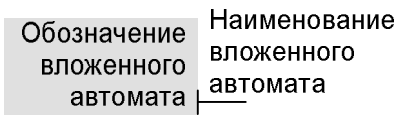
Объект “Наименование и обозначение входного воздействия”



Объект “Наименование и обозначение признака выходного воздействия”

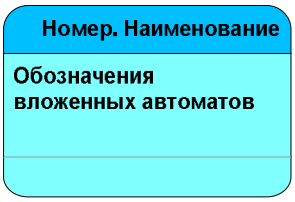


Объект “Наименование и обозначение вложенного подавтомата”

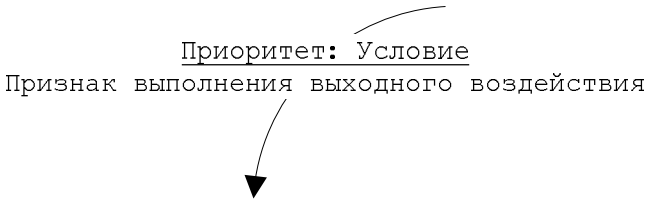


Объекты графа переходов:

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
//  ПОСЛЕДОВАТЕЛЬНОСТЬ ПРОВЕРКИ УСЛОВИЙ И ДЕЙСТВИЙ в состоянии 3
LBL     3

Network 29
//  Переход с приоритетом 1 из состояния 3 в состояние 8
LD      DI_CCEstopWP
O       PBEstopWP
O       DI_CCoffWP
O       PBoffWP
MOVB    8 , YATWPSFW
=       KoffP_WP
R       KopS_FW , 1
=       KoffS_FW
JMP     255

Network 30
//  Переход с приоритетом 2 из состояния 3 в состояние 4
LD      DDI_X3S_FW
ON      DDI_X1P_WP
MOVB    4 , YATWPSFW
R       KopS_FW , 1
JMP     255

Network 31
//  Переход с приоритетом 3 из состояния 3 в состояние 8
LD      MopS_FW
MOVB    8 , YATWPSFW
R       KopS_FW , 1
=       KoffP_WP
JMP     255

Network 32
//  Переход  по умолчанию
LD      SM0.0
JMP     255

Network 33
//  ПОСЛЕДОВАТЕЛЬНОСТЬ ПРОВЕРКИ УСЛОВИЙ И ДЕЙСТВИЙ в состоянии 4
LBL     4

Network 34
//  Переход с приоритетом 1 из состояния 4 в состояние 8
LD      DI_CCEstopWP
O       PBEstopWP
MOVB    8 , YATWPSFW
=       KoffP_WP
JMP     255

Network 35
//  Переход с приоритетом 2 из состояния 4 в состояние 5
LDN     DDI_X1P_WP
A       CclS_FW
MOVB    5 , YATWPSFW
S       KclS_FW , 1
JMP     255

Network 36
//  Переход с приоритетом 3 из состояния 4 в состояние 88
```

```
LDN    DDI_X1P_WP
AN     CclS_FW
MOVB   88 , YATWPSFW
=      MNCclS_FW
=      ATWPSFWoffErr
JMP    255



Network 37
//  Переход с приоритетом 4 из состояния 4 в состояние 5
LD     DI_CCoffWP
O      PBoffWP
A      CclS_FW
MOVB   5 , YATWPSFW
S      KclS_FW , 1
JMP    255

Network 38
//  Переход с приоритетом 5 из состояния 4 в состояние 6
LD     DI_CCoffWP
O      PBoffWP
AN     CclS_FW
MOVB   6 , YATWPSFW
=      MNCclS_FW
=      KoffP_WP
JMP    255

Network 39
//  Переход  по умолчанию
LD     SM0.0
JMP    255

Network 40
//  ПОСЛЕДОВАТЕЛЬНОСТЬ ПРОВЕРКИ УСЛОВИЙ И ДЕЙСТВИЙ в состоянии 5
LBL    5

Network 41
//  Переход с приоритетом 1 из состояния 5 в состояние 6
LD     DDI_X4S_FW
O      MclS_FW
MOVB   6 , YATWPSFW
R      KclS_FW , 1
=      KoffP_WP
JMP    255

Network 42
//  Переход с приоритетом 2 из состояния 5 в состояние 8
LD     DI_CCEstopWP
O      PBEstopWP
MOVB   8 , YATWPSFW
=      KoffP_WP
JMP    255

Network 43
//  Переход  по умолчанию
LD     SM0.0
JMP    255

Network 44
//  ПОСЛЕДОВАТЕЛЬНОСТЬ ПРОВЕРКИ УСЛОВИЙ И ДЕЙСТВИЙ в состоянии 6
LBL    6

Network 45
//  Переход с приоритетом 1 из состояния 6 в состояние 7
LDN    DDI_X1P_WP
MOVB   7 , YATWPSFW
=      ATWPSFWoff
JMP    255

Network 46
//  Переход с приоритетом 2 из состояния 6 в состояние 88
LD      M2P_WP
```

```
MOVB   88 , YATWPSFW
=      ATWPSFWoffErr
JMP    255

Network 47
//  Переход  по умолчанию
LD     SM0.0
JMP    255

Network 48
//  ПОСЛЕДОВАТЕЛЬНОСТЬ ПРОВЕРКИ УСЛОВИЙ И ДЕЙСТВИЙ в состоянии 8
LBL    8

Network 49
//  Переход с приоритетом 1 из состояния 8 в состояние 5
LDN    DDI_X1P_WP
O      M2P_WP
A      CclS_FW
MOVB   5 , YATWPSFW
S      KclS_FW , 1
JMP    255

Network 50
//  Переход с приоритетом 2 из состояния 8 в состояние 88
LDN    DDI_X1P_WP
O      M2P_WP
AN     CclS_FW
MOVB   88 , YATWPSFW
=      MNCclS_FW
=      ATWPSFWoffErr
JMP    255

Network 51
LBL    255

END_SUBROUTINE_BLOCK
```

Автомат *ATMainWP* (рис. 13, 14)

```
SUBROUTINE_BLOCK ATMainWP:SBR12
TITLE=SUBROUTINE COMMENTS
//  СОЗДАНО с использованием КОНВЕРТОРА
//  графа переходов алгоритма (изображенного в Visio) в текст программы
//
// ********************************
// **** Входные переменные
// ********************************
// ATWPSFWoff - Цикл управления НА завершился нормально
// ATWPSFWoffErr - Цикл управления НА завершился ненормально
// ConP_WP - Есть все условия для автоматического пуска промывного насоса (DDI_ReadyWP -
Насосный агрегат к пуску готов)
// DDI_X4S_FW - Напорная задвижка закрыта
// DI_CCEstopWP - Команда от ШКО "Аварийное отключение насосного агрегата"
// DI_CCoffWP - Команда от ШКО "Отключить насосный агрегат"
// CConWP - Команда от ШКО "Включить насосный агрегат"
// PBEstopWP - Команда по сети PROFIBUS "Аварийная остановка насосного агрегата"
// PBoffWP - Команда по сети PROFIBUS "Отключить насосный агрегат"
// PBonWP - Команда по сети PROFIBUS "Включить насосный агрегат"
// ReadyAC - Есть все условия для работы в автоматическом режиме управления: (DI_380V -
Питание 380 В DI_OnAS220V - Включен автоматический выключатель питания цепей управления ~220
В DI_OnKIP220V - Включен автоматический выключатель питания КИП ~220 В DI_AC - Избиратель
режима управления в положении "Автоматический" DI_CACWP - НА выбран для управления)
// ********************************
// **** Выходные процедуры
// ********************************
// zFinit - Инициализация номеров состояний автоматов и переменных , формируемых в автоматах:
(Номер состояния автомата управления насосным агрегатом и напорной задвижкой: YATWPSFW=0
Стирание команд управления KonP_WP , KoffP_WP , KopS_FW , KclS_FW , KoffS_FW ,
DDO_predpuskWP)
// zMNReadyACWP - Запись признака "Нет готовности к пуску промывного насоса в автоматическом
режиме"
```

```
// zStopPS - Запись команд управления (KoffP_WP , KoffS_FW)
// ***************************************
// **** Вызываемые подАвтоматы
// ***************************************
// ATWPSFW - Вызываемый автомат цикла управления НА
//
BEGIN

Network 1
//  Обнуление (сброс) команд , признаков и т.п. , записываемых в автомате ,
//  но не имеющих обнуления по условию
LDN    SM0.0
=      Finit
=      MNReadyACWP
=      StopPS

Network 2
//  ОПРЕДЕЛЕНИЕ ТЕКУЩЕГО СОСТОЯНИЯ
//  АНАЛОГ --- case: 0
LDB=   YATMainWP , 0
JMP    0

Network 3
//  ОПРЕДЕЛЕНИЕ ТЕКУЩЕГО СОСТОЯНИЯ
//  АНАЛОГ --- case: 1
LDB=   YATMainWP , 1
JMP    1

Network 4
//  Переход  по умолчанию
LD     SM0.0
JMP    255

Network 5
//  ПОСЛЕДОВАТЕЛЬНОСТЬ ПРОВЕРКИ УСЛОВИЙ И ДЕЙСТВИЙ в состоянии 0
LBL    0

Network 6
//  Переход с приоритетом 1 из состояния 0 в состояние 1
LD     ReadyAC
A      DDI_X4S_FW
A      ConP_WP
AN     DI_CCoffWP
AN     DI_CCEstopWP
AN     PBoffWP
AN     PBEstopWP
LD     CConWP
O      PBonWP
ALD
MOVB   1 , YATMainWP
=      Finit
JMP    255

Network 7
//  Переход с приоритетом 2 из состояния 0 в состояние 0
LD     ReadyAC
A      DDI_X4S_FW
A      ConP_WP
LPS
NOT
=      MNReadyACWP
LPP
NOT
JMP    255

Network 8
//  Переход  по умолчанию
LD     SM0.0
JMP    255

Network 9
//  ПОСЛЕДОВАТЕЛЬНОСТЬ ПРОВЕРКИ УСЛОВИЙ И ДЕЙСТВИЙ в состоянии 1
LBL    1
```

```
Network 10
//  Переход с приоритетом 1 из состояния 1 в состояние 0
LDN     ReadyAC
O       ATWPSFWoff
O       ATWPSFWoffErr
MOVB    0 , YATMainWP
=       Finit
=       StopPS
JMP     255

Network 11
LBL     255

Network 12
//  Обнуление (сброс) признаков вызова ПОДАВТОМАТОВ
LDN     SM0.0
=       callATWPSFW

Network 13
//  ВЫЗОВ ПОДАВТОМАТОВ , ЕСЛИ ТЕКУЩЕЕ СОСТОЯНИЕ - 1
LDB=    YATMainWP , 1
=       callATWPSFW
JMP     254

Network 14
LBL     254

END_SUBROUTINE_BLOCK
```

Приложение 6. Примеры подпрограмм вызова автоматов

Вызов автомата *ACPx*

```
SUBROUTINE_BLOCK CALL_ACP_WP:SBR18
TITLE=
//
BEGIN

Network 1
//   Формирование формальных параметров DATA_ACPx --- Запись KoffP
LD      KoffP_WP
=      KoffP

Network 2
//   Формирование формальных параметров DATA_ACPx --- Запись KonP
LD     KonP_WP
=      KonP

Network 3
//   Формирование формальных параметров DATA_ACPx --- Запись TACP1
LD     TACP1_WP
=      TACP1

Network 4
//   Формирование формальных параметров DATA_ACPx --- Запись X0P
LD     SM0.0
=      X0P

Network 5
//   Формирование формальных параметров DATA_ACPx --- Запись X1P
LD     DDI_X1P_WP
=      X1P

Network 6
//   Запись YACPx
//   Собственно вызов автомата
//   Перезапись YACP_WP
LD     SM0.0
MOVB   YACP_WP , YACPx
CALL   ACPx
MOVB   YACPx , YACP_WP

Network 8
//   Формирование значения фактической переменной после вызова автомата
LD     zM1P
A      ReadyAC
=      M1P_WP

Network 9
//   Формирование значения фактической переменной после вызова автомата
LD     zM2P
A      ReadyAC
=      M2P_WP

Network 10
//   Формирование значения фактической переменной после вызова автомата
LD     zM3P
A      ReadyAC
=      M3P_WP

Network 11
//   Формирование значения фактической переменной после вызова автомата
LD     zNZ1P
R      DDO_Z1WP , 1

Network 12
//   Формирование значения фактической переменной после вызова автомата
LD     zZ1P
S      DDO_Z1WP , 1
```

```
Network 13
//   Формирование значения фактической переменной после вызова автомата
LDN    zstacp1
TON    TACP1_WP , STG_TAC1

END_SUBROUTINE_BLOCK
```

Вызов автомата *ACSx*

```
SUBROUTINE_BLOCK CALL_ACS_FW:SBR16
TITLE=
//
BEGIN

Network 1
//   Формирование формальных параметров DATA_ACSx --- Запись KclS
LD     KclS_FW
LD     PBKclFW
EU
AB=    YATMainWP , 0
OLD
=      KclS

Network 2
//   Формирование формальных параметров DATA_ACSx --- Запись KoffS
LD     KoffS_FW
LD     PBKoffFW
EU
AB=    YATMainWP , 0
OLD
=      KoffS

Network 3
//   Формирование формальных параметров DATA_ACSx --- Запись KopS
LD     KopS_FW
LD     PBKopFW
EU
AB=    YATMainWP , 0
OLD
=      KopS

Network 4
//   Формирование формальных параметров DATA_ACSx --- Запись TACS1
LD      TACS1_FW
=      TACS1

Network 5
//   Формирование формальных параметров DATA_ACSx --- Запись TACS2
LD     TACS2_FW
=      TACS2

Network 6
//   Формирование формальных параметров DATA_ACSx --- Запись X0S
LD     DDI_X0S_FW
=      X0S

Network 7
//   Формирование формальных параметров DATA_ACSx --- Запись X1S
LD     DDI_X1S_FW
=      X1S

Network 8
LD     noneX1S_time_FW
=      noneX1S_time

Network 9
LD     noneX2S_time_FW
=      noneX2S_time

Network 10
//   Формирование формальных параметров DATA_ACSx --- Запись X2S
LD     DDI_X2S_FW
```

```
=      X2S

Network 11
//   Формирование формальных параметров DATA_ACSx --- Запись X3S
LD     DDI_X3S_FW
=      X3S

Network 12
//   Формирование формальных параметров DATA_ACSx --- Запись X4S
LD     DDI_X4S_FW
=      X4S

Network 13
//   Запись YACSx
//   Собственно вызов автомата
//   Перезапись YACS_FW
LD     SM0.0
MOVB   YACS_FW , YACSx
CALL   ACSx
MOVB   YACSx , YACS_FW

Network 15
//   Формирование значения фактической переменной после вызова автомата
LD     zM1S
=      M1S_FW

Network 16
//   Формирование значения фактической переменной после вызова автомата
LD     zM2S
=      M2S_FW

Network 17
//   Формирование значения фактической переменной после вызова автомата
LD     zM3S
=      M3S_FW

Network 18
//   Формирование значения фактической переменной после вызова автомата
LD     zM4S
=      M4S_FW

Network 19
//   Формирование значения фактической переменной после вызова автомата
LD     zM7S
=      M7S_FW

Network 20
//   Формирование значения фактической переменной после вызова автомата
LD     zM8S
=      M8S_FW

Network 21
//   Формирование значения фактической переменной после вызова автомата
LD     zMclS
=      MclS_FW

Network 22
//   Формирование значения фактической переменной после вызова автомата
LD     zMopS
=      MopS_FW

Network 23
//   Формирование значения фактической переменной после вызова автомата
LD     zNZ1S
R      DDO_Z1S_FW , 1

Network 24
//   Формирование значения фактической переменной после вызова автомата
LD     zNZ2S
R      DDO_Z2S_FW , 1

Network 25
//   Формирование значения фактической переменной после вызова автомата
LD     zZ1S
```

```
S      DDO_Z1S_FW , 1

Network 26
//   Формирование значения фактической переменной после вызова автомата
LD     zZ2S
S      DDO_Z2S_FW , 1

Network 27
//   Формирование значения фактической переменной после вызова автомата
LDN    zstacs1
TON    TACS1_FW , STG_TAC1

Network 28
//   Формирование значения фактической переменной после вызова автомата
LDN    zstacs2
TON    TACS2_FW , STG_TAC2


END_SUBROUTINE_BLOCK
```

Вызов автомата *ATWPSFW*

```
SUBROUTINE_BLOCK CALL_ATWPSFW:SBR14
TITLE=
//
BEGIN

Network 1
//   Собственно вызов автомата
LD     SM0.0
CALL   ATWPSFW


Network 2
//   Формирование значения фактической переменной после вызова автомата
LD      stps
TON    TPS , STG_TPS

END_SUBROUTINE_BLOCK
```

Вызов автомата *ATMainWP*

```
SUBROUTINE_BLOCK CALL_ATMainWP:SBR11
TITLE=
//
BEGIN

Network 1
//   Собственно вызов автомата
LD     SM0.0
CALL   ATMainWP

Network 2
LD      Finit
CALL   SBR13

Network 3
LD     callATWPSFW
CALL   CALL_ATWPSFW

Network 4
LD     StopPS
S      KoffP_WP , 1
S      KoffS_FW , 1

END_SUBROUTINE_BLOCK
```