«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ имени академика С.П. КОРОЛЕВА»

Илюхин В.Н.

Программирование промышленных логических контроллеров «ОВЕН» в системе «CoDeSys»

Конспект лекций по дисциплине

«Средства электроавтоматики пневмо- и гидросистем»

Содержание

Лекция 1

1 Цель курса ……... 4

2 Описание ПЛК ОВЕН …….4

2.1 Введение ……4

2.2 Назначение ……6

2.3 Технические характеристики ……. 6

2.4 Условия эксплуатации …….. 10

2.5 Устройство контроллера ОВЕН ПЛК 150 ……... 10

Лекция 2

3 Программирование ПЛК в CoDeSys 2.3 …… 13

3.1 Введение ……. 13

3.2 Компоненты проекта ……. 14

3.3 Языки программирования …… 26

Лекция 3

3.3.1 Список инструкций (IL) …… …….. 26

3.3.2 Модификаторы и операторы IL …….. 26

3.3.3 Структурированный текст (ST) ……... 28

Лекция 4

3.3.4 Язык последовательных функциональных схем (SFC) … 35

3.3.5 Язык функциональных блоковых диаграмм (FBD) …….. 44

Лекция 5

3.3.6 Непрерывные функциональные схемы (CFC) …….. 45

3.3.7 Язык релейных диаграмм (LD) ……... 45

3.4 Отладка и online функции ……. 49

4 Контрольные вопросы к лабораторной работе …….. 52

5 Пример …….. 52

6 Порядок выполнения работы …….. 59

7 Задание …….. 60

Литература …….. 61

Приложение А. Схемы подключения к ОВЕН ПЛК150 …… 62

Приложение Б. Операторы в CoDeSys …… 64

Лекция 1

1 Цель курса

Целями являются:

1. Изучение программируемого логического контроллера ОВЕН ПЛК 150 и принципа его работы.
2. Программирование ОВЕН ПЛК150 в CoDeSys 2.3.

2 Описание ПЛК ОВЕН

2.1 Введение

Любая автоматизированная система содержит в своём составе элементы, заставляющие её выполнять нужные действия в нужном порядке, иными словами систему управления. Так, для паровых машин был изобретён регулятор Уатта. С увеличением скорости вращения вала его подпружиненные шарики расходятся и через механическую тягу закрывают паровой клапан, уменьшая скорость вращения. Вторым достижением в сфере автоматизации по его исторической значимости можно назвать Жаккардовский ткацкий станок. Он позволял автоматически изготавливать полотна, составленные из нитей разного цвета по заданной программе. В последние десятилетия элементы управления систем автоматизации выполняются в виде электронных устройств. По сравнению с механическими системами они выигрывают по надёжности, цене, габаритным размерам.

Широкое распространение получили программируемые логические контроллеры (ПЛК) – устройства, автоматизирующие работу как отдельных аппаратов, например станков с программным управлением, так и огромных производственных комплексов.

Контроллер ПЛК150 выпускается в различных модификациях, отличающихся типом установленных в него дискретных выходных

элементов, напряжением питания и различными лицензионными ограничениями размера памяти ввода-вывода программы ПЛК.

Условное обозначение контроллера:

Напряжение питания:

220 В переменного тока

24 В постоянного тока;

Тип встроенных аналоговых выходных элементов:

И – цифроаналоговый преобразователь "параметр – ток 4...20 мА"

У– цифроаналоговый преобразователь "параметр – напряжение 0..10 В"

А– цифроаналоговый преобразователь "параметр – ток 4...20 мА или напряжение 0...10 В"

Размер лицензионного ограничения на область ввода-вывода:

L – ограничение в 360 байт

M – без ограничения

Примеры условного обозначения контроллера при заказе:

ПЛК150-24.И-L – контроллер с номинальным напряжением питания 24 В постоянного тока, оснащенный встроенными цифроаналоговыми преобразователями "параметр – ток 4...20 мА" и имеющий лицензионное ограничение на размер области ввода-вывода в 360 байт.

ПЛК150-220.А-М – контроллер с номинальным напряжением питания 220 В переменного тока, оснащенный встроенными цифроаналоговыми преобразователями "параметр – ток 4...20 мА или напряжение 0...10 В" и не имеющий лицензионного ограничения на размер области ввода-вывода.

2.2 Назначение

Программируемый логический контроллер ОВЕН ПЛК150 предназначен для создания систем автоматизированного управления технологическим оборудованием в различных областях промышленности, жилищно-коммунального и сельского хозяйства.

Логика работы ПЛК150 определяется потребителем в процессе программирования контроллера. Программирование осуществляется с помощью системы программирования CoDeSys 2.3.

2.3 Технические характеристики

Основные технические характеристики, характеристики входных сигналов и характеристики встроенных выходных элементов контроллера ПЛК150 приведены в таблицах 1, 2, 3, 4.

Таблица 1 - Основные технические характеристики контроллера ПЛК150

Параметр	Значение
Общие сведения	
Конструктивное исполнение	Унифицированный корпус для крепления на DIN-рейку, длина 105 мм (6U), шаг клемм 7,5 мм
Степень защиты корпуса	IP20
Напряжение питания: ПЛК150-24 ПЛК150-220	 18...29 В постоянного тока (номинальное 24 В) 90...264 В переменного тока (номинальное 220 В) частотой 47...63 Гц
Потребляемая мощность	6 Вт
Индикация передней панели	1 индикатор питания 6 индикаторов состояний дискретных входов 4 индикатора состояний выходов 1 индикатор наличия связи с CoDeSys 1 индикатор работы программы пользователя
Ресурсы	
Центральный процессор	32-х разрядный RISC-процессор 200 МГц на базе ядра ARM9
Объем оперативной памяти	8 МВ

Продолжение табл. 1

Параметр	Значение
Объем энергонезависимой памяти хранения программ и архивов*	4 МВ
Размер Retain-памяти**	4 кВ
Дискретные Входы	
Количество дискретных входов	6
Гальваническая изоляция дискретных входов	есть, групповая
Электрическая прочность изоляции дискретных входов	1,5 кВ
Максимальная частота сигнала, подаваемого на дискретный вход	10 кГц
Дискретные Выходы	
Количество дискретных выходов	4 э/м реле
Характеристики дискретных выходов	Ток коммутации до 2 А при напряжении не более 220 В 50 Гц и cos φ> 0,4
Гальваническая изоляция дискретных выходов	есть, индивидуальная
Электрическая прочность изоляции дискретных выходов	1,5 кВ
Аналоговые Входы	
Количество аналоговых входов	4
Типы поддерживаемых унифицированных входных сигналов	Напряжение 0...1 В, 0...10 В Ток 0...5 мА, 0(4)...20 мА Сопротивление 0...5 кОм

Параметр	Значение
Типы поддерживаемых датчиков	Термосопротивления: ТСМ50М, ТСП50П, ТСМ100М, ТСП100П, ТСН100Н, ТСМ500М, ТСП500П, ТСН500Н, ТСП1000П, ТСН1000Н Термопары: ТХК (L), ТЖК (J), ТНН (N), ТХА (K), ТПП (S), ТПП (R), ТПР (B), ТВР (A-1), ТВР (A-2)
Время опроса одного аналогового входа	0,5 с
Предел основной приведенной погрешности измерения аналоговыми входами	около 0,5 %***
Гальваническая изоляция аналоговых входов	отсутствует
Аналоговые Выходы	
Количество аналоговых выходов	2
Разрядность ЦАП	10 бит
Тип выходного сигнала: ПЛК150-И ПЛК150-У ПЛК150-А	 Ток 4...20 мА Напряжение 0...10 В Ток 4...20 мА или напряжение 0...10 В
Питание аналоговых выходов	встроенное, общее на все выходы
Гальваническая изоляция аналоговых выходов	есть, групповая
Электрическая прочность изоляции аналоговых выходов	1,5 кВ

Продолжение табл. 1

Параметр	Значение
Интерфейсы связи	
Интерфейсы	Ethernet 100 Base-T RS-232 RS-485
Скорость обмена по протоколам RS	до 115200 bps
Протоколы	ОВЕН ModBus-RTU, ModBus-ASCII DCON ModBus-TCP GateWay (протокол CoDeSys)
Программирование	
Среда программирования	CoDeSys 2.3
Интерфейс для программирования и отладки	RS-232 или Ethernet

* Для хранения программ и архивов используется Flash&память, специализированная файловая система

** Настраивается пользователем

*** Величина справочная и приводится для приблизительной оценки величины основной приведенной погрешности измерения

Таблица 2 - Характеристики дискретных входных сигналов

Модификация контроллера	Сигнал, подаваемый на дискретный вход	Комментарий
ПЛК150-24	29...15 В* – логическое значение 1 5...0 В* – логическое значение 0	Вход срабатывает при протекающем через него токе не менее 3 мА
ПЛК150-220	С помощью сухого контакта или ключа, коммутирующего общую клемму дискретных входов и клемму конкретного входа	Суммарное сопротивление контакта и линии подключения должно быть не более 100 Ом

* Напряжение относительно минусовой клеммы питания

Таблица 3 - Характеристики встроенных аналоговых выходных элементов

Обозначение при заказе	Наименование	Характеристики
И	Цифроаналоговый преобразователь "параметр – ток 4...20 мА"	Сопротивление нагрузки от 0 до 900 Ом
У	Цифроаналоговый преобразователь "параметр – напряжение 0...10 В"	Сопротивление нагрузки от 2 кОм
А	Цифроаналоговый преобразователь "параметр – ток 4...20 мА или напряжение 0...10 В"	Сопротивление нагрузки от 150 до 900 Ом для токового сигнала и свыше 10 кОм для сигнала напряжения

Таблица 4 - Типы датчиков и унифицированных сигналов, подключаемых к аналоговым входам

Наименование	Диапазон измерений
Термопреобразователи сопротивления (по ГОСТ 6651- 94)	
ТСМ (Cu 50) W_{100} = 1,4260	–50 °С...+200 °С
ТСМ (50М) W_{100} = 1,4280	–190 °С...+200 °С
ТСП (Pt 50) W_{100} = 1,3850	–200 °С...+750 °С
ТСП (50П) W_{100} =1,3910	–200 °С...+750 °С
ТСМ (Cu 100) W_{100} =1,4260	–50 °С...+200 °С
ТСМ (100М) W_{100} =1,4280	–190 °С...+200 °С
ТСП (Pt 100) W_{100} =1,3850	–200 °С...+750 °С
ТСП (100П) W_{100} =1,3910	–200 °С...+750 °С
ТСН (100Н) W_{100}=1,6170	–60 °С...+180 °С
ТСМ (Cu 500) W_{100} =1,4260	–50 °С...+200 °С
ТСМ (500М) W_{100} =1,4280	–190 °С...+200 °С
ТСП (Pt 500) W_{100} =1,3850	200 °С...+750 °С
ТСП (500П) W_{100} =1,3910	–200 °С...+750 °С
ТСН (500Н) W_{100} =1,6170	–60 °С...+180 °С
ТСМ (Cu 1000) W_{100} =1,4260	–50 °С...+200 °С
ТСМ (1000М) W_{100} =1,4280	–190 °С...+200 °С
ТСП (Pt 1000) W_{100} =1,3850	–200 °С...+750 °С
ТСП (1000П) W_{100} =1,3910	–200 °С...+750 °С
ТСН (1000Н) W_{100} =1,6170	–60 °С...+180 °С

Наименование	Диапазон измерений
Термопары (по ГОСТ Р 8.585-2001)	
ТХК (L)	–200 °С...+800 °С
ТЖК (J)	–200 °С...+1200 °С
ТНН (N)	–200 °С...+1300 °С
ТХА (K)	–200 °С...+1300 °С
ТПП (S)	0 °С...+1600 °С
ТПП (R)	0 °С...+1600 °С
ТПР (B)	+200 °С...+1800 °С
ТВР (А-1)	0 °С...+2500 °С
ТВР (А-2)	0 °С...+1800 °С
ТВР (А-3)	0 °С...+1600 °С
ТМК (Т)	–200 °С...+400 °С
Унифицированные сигналы постоянного напряжения и тока (по ГОСТ 26.011-80)	
0...5,0 мА	0...100 %
0...20,0 мА	0...100 %
4,0...20,0 мА	0...100 %
–50,0...+50,0 мВ	0...100 %
0...1,0 В	0...100 %
0...10,0 В	0...100 %
Датчики сопротивления	
0... 5000 Ом	0...100 %

2.4 Условия эксплуатации

Контроллер ОВЕН ПЛК150 эксплуатируется при следующих условиях:

– закрытые взрывобезопасные помещения или шкафы электрооборудования без агрессивных паров и газов;

– температура окружающего воздуха от минус 20 °С до +70 °С;

– верхний предел относительной влажности воздуха – 80 % при 25 °С и более низких температурах без конденсации влаги;

– атмосферное давление от 84 до 106,7 кПа.

По устойчивости к климатическим воздействиям при эксплуатации ПЛК150 соответствует группе исполнения В4 по ГОСТ 12997&84.

По устойчивости к механическим воздействиям при эксплуатации ПЛК150 соответствует группе исполнения N2 по ГОСТ 12997.

2.5 Устройство контроллера ОВЕН ПЛК 150

Контроллер ОВЕН ПЛК150 выпускается в корпусе, предназначенном для крепления на DIN-рейке 35 мм. Подключение всех внешних связей осуществляется через разъемные соединения, расположенные по двум боковым и передней (лицевой) сторонам контроллера. Открытие корпуса для подключения внешних связей не требуется.

Схематический внешний вид контроллера показан на рис.1.

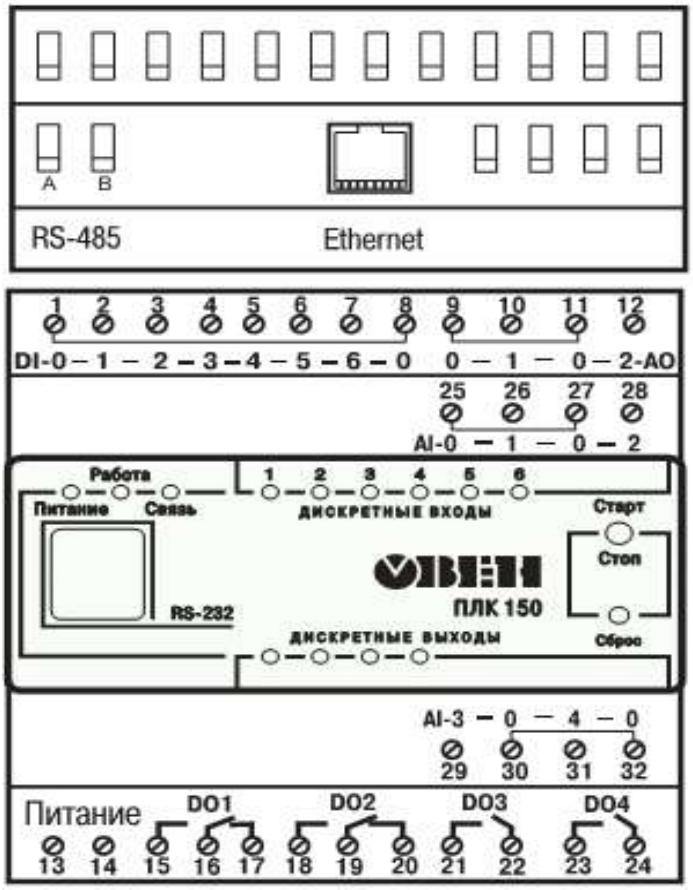


Рисунок 1 - Внешний вид ПЛК150

На боковой стороне расположены разъемы интерфейсов Ethernet и RS&485.

На лицевой панели расположен порт Debug RS&232, предназначенный для связи со средой программирования, загрузки

программы и отладки. Также порт Debug RS-232 может быть использован для подключения устройств, работающих по протоколам Modbus, ОВЕН и DCON.

По обеим боковым сторонам контроллера расположены клеммы для подключения дискретных датчиков и исполнительных механизмов. Схемы подключения приведены в Приложении А.

На переднюю панель контроллера выведена светодиодная индикация о состоянии дискретных входов и выходов, о наличии питания и о наличии связи со средой программирования CoDeSys. Также на передней панели имеются две кнопки: кнопка

, предназначенная для запуска и остановки программы в контроллере и скрытая кнопка

, предназначенная для перезагрузки контроллера. Нажать кнопку

возможно только тонким заостренным предметом.

В корпусе контроллера расположен маломощный звуковой излучатель, управляемый из пользовательской программы как дополнительный дискретный выход. Звуковой излучатель может быть использован для функций аварийной или иной сигнализации или для отладочных нужд. Частота звукового сигнала излучателя фиксированная и не поддается настройке.

Контроллер ПЛК150 оснащен встроенными часами реального времени, имеющими собственный аккумуляторный источник питания. Энергии полностью заряженного аккумулятора хватает на непрерывную работу часов реального времени в течение 6 месяцев (при температуре 15–35°С). В случае износа аккумулятора, не полной его зарядки, а также при работе при более низких температурах время работы часов реального времени может сократиться.

Аккумулятор, используемый для питания часов реального времени, дополнительно используется как источник аварийного питания микропроцессора контроллера. При случайном отключении основного

питания контроллер переходит на аварийное питание и сохраняет промежуточные результаты вычислений и работоспособность интерфейсов Ethernet в течение 10 минут. Светодиодная индикация и выходные элементы контроллера при этом не запитываются и не функционируют. При включении основного питания во время работы на аварийном питании контроллер сразу приступает к выполнению пользовательской программы, не тратя время на загрузку и сохраняя все промежуточные результаты вычислений. После 10 минут работы на аварийном питании контроллер записывает Retain-переменные в энергонезависимую память и отключается. Часы реального времени остаются в рабочем состоянии. После включения основного питания контроллер загружается и запускает программу пользователя (если установлена опция автозапуска). Время работы от аварийного источника питания может быть автоматически скорректировано самим контроллером в зависимости от степени зарядки аккумулятора и температуры окружающей среды.

Во время загрузки контроллера его выходы переводятся в заранее заданное «безопасное состояние», в которых находятся до полной загрузки контроллера и запуска пользовательской программы.

Примечание. «Безопасное состояние» – это состояние выходов контроллера, при котором подключенные к ним исполнительные механизмы находятся в состоянии, наиболее безопасном для объекта управления, не приводящим к его поломке. Значение «безопасного состояния» выходов задается при конфигурировании области ввода-вывода в PLC-Configuration.

Лекция 2

3 Программирование ПЛК в CoDeSys 2.3

3.1 Введение

CoDeSys - это современный инструмент для программирования контроллеров (CoDeSys образуется от слов Controllers Development System).

CoDeSys предоставляет программисту удобную среду для программирования контроллеров на языках стандарта МЭК 61131-3. Используемые редакторы и отладочные средства базируются на широко известных и хорошо себя зарекомендовавших принципах, знакомых по другим популярным средам профессионального программирования (такие, как Visual C++).

Таблица 5 - Состав базовой пользовательской документации по CoDeSys

Модуль	Документы	Файлы
Среда программирования CoDeSys	Печатное руководство и встроенная система помощи Первые шаги с CoDeSys (пример)	Manual_V23_RU.pdf First Steps with CoDeSys RU.pdf
Gateway Server	Концепции, установка, встроенная система помощи, интерфейс и настройка (открывается двойным щелчком мыши на иконке в панели задач)	Gateway Manual.pdf
OPC Server	OPC-Server V2.0, установка и применение	OPC_20_How_to_use.pdf
CoDeSys Visualization	Описание CoDeSys визуализации, включая CoDeSys HMI, Target- и Web-Visualization	CoDeSys_Visu_RU.pdf
SoftMotion	Описание применения и библиотек SoftMotion	SoftMotion_Manual_V23.pdf
Библиотеки	Standard.lib и Util.lib описаны в печатном руководстве по программированию. Для каждой системной библиотеки CoDeSys существует отдельный документ <library name>.pdf SoftMotion библиотеки: см. SoftMotion-документацию.	<SysLib-Name>.pdf UserManual_V23_RU.pdf
ENI Server	Установка и настройка ENI Сервера, управление версиями, работа с внешней базой данных. Настройка ENI в CoDeSys: описана в печатном руководстве по программированию. ENI Admin, ENI Control и ENI Explorer: см. встроенные системы помощи.	EniServerQuickstart.pdf UserManual_V23_RU.pdf

3.2 Компоненты проекта

Проект

Проект включает следующие объекты: POU, типы данных, визуализации, ресурсы, библиотеки. Каждый проект сохраняется в отдельном файле.

POU (Program Organization Unit)

К программным компонентам (POU) относятся функциональные блоки, функции и программы. Отдельные POU могут включать действия (подпрограммы).

Каждый программный компонент состоит из раздела объявлений и кода. Для написания всего кода POU используется только один из МЭК языков программирования (IL, ST, FBD, SFC, LD или CFC).

CoDeSys поддерживает все описанные стандартом МЭК компоненты. Для их использования достаточно включить в свой проект библиотеку standard.lib.

POU могут вызывать другие POU, но рекурсии недопустимы.

Функция

Функция – это POU, который возвращает только единственное значение (которое может состоять из нескольких элементов, если это битовое поле или структура). В текстовых языках функция вызывается как оператор и может входить в выражения.

При объявлении функции необходимо указать тип возвращаемого значения. Для этого после имени функции нужно написать двоеточие и тип.

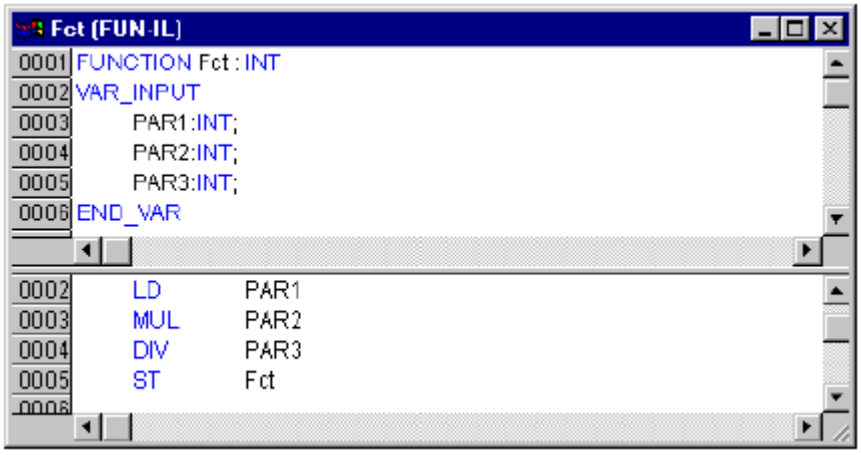
Правильно объявленная функция выглядит следующим образом:

FUNCTION Fct: INT;

Имя функции используется как выходная переменная, которой присваивается результат вычислений.

Объявление функции должно начинаться с ключевого слова FUNCTION и заканчиваться ключевым словом END_FUNCTION. Вот пример функции, написанной на IL, которая использует три входных переменных и возвращает результат деления произведения первых двух на третью.

Пример функции, написанной на языке IL:



В языке ST вызов функции может присутствовать в выражениях как операнд.

В SFC функция вызывается только из шага или перехода.

Функция не имеет внутренней памяти. Это означает, что функция с одними и теми же значениями входных переменных всегда возвращает одно и то же значение.

Например:

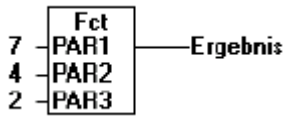
На IL:

```
LD    7
Fct   2,4
ST    Result
```

На ST:

```
Result : =  Fct  (7, 2, 4);
```

На FBD:



Функциональный блок

Функциональный блок - это POU, который принимает и возвращает произвольное число значений. В отличие от функции функциональный блок не формирует возвращаемое значение.

Объявление функционального блока начинается с ключевого слова FUNCTION_BLOCK и заканчивается ключевым словом END_FUNCTION_BLOCK.

Ниже приведен пример функционального блока, написанного на IL, который имеет две входных и две выходных переменных. Значение выходной переменной MULERG равно произведению значений двух входных переменных, а значение VERGL определяется в результате сравнения значений входных переменных.

Пример функционального блока:

```
FUB (FB-AWL)
0001 FUNCTION_BLOCK FUB
0002 VAR_INPUT
0003     PAR1:INT;
0004     PAR2:INT;
0005 END_VAR
0006 VAR_OUTPUT
0007     MULERG:INT;
0008     VERGL:BOOL;
0009 END_VAR
0010

0001     LD    PAR1
0002     MUL   PAR2
0003     ST    MULERG
0004
0005     LD    PAR1
0006     EQ    PAR2
0007     ST    VERGL
```

Экземпляры функционального блока

Определение функционального блока подобно определению типа данных. Для работы с функциональным блоком необходимо объявить (создать) его экземпляр. Один функциональный блок может иметь произвольное число экземпляров, каждый из которых имеет собственные независимые данные (память).

Каждый экземпляр функционального блока получает свой собственный идентификатор (имя экземпляра) и свои данные, содержащие входные, выходные и внутренние переменные. Экземпляры

функционального блока объявляются глобально или локально как переменные, имеющие тип соответствующего функционального блока.

Пример объявления экземпляра с идентификатором INSTANCE функционального блока FUB:

INSTANCE: FUB;

Вызов экземпляра функциональный блока происходит с помощью его имени. Входные и выходные переменные доступны вне функционального блока, а внутренние переменные доступны только в самом блоке.

Пример использования входных переменных:

функциональный блок fb имеет входную переменную inl типа INT:

```
PROGRAM prog
VAR
inst1 : fb;
END_VAR

LD      17
ST      inst1.in1
CAL     inst1
END_PROGRAM
```

Экземпляры функционального блока могут быть объявлены в другом функциональном блоке или в программе. Объявлять экземпляр функционального блока в теле функции нельзя. Экземпляры функционального блока доступны в том POU, в котором они объявлены, если они не объявлены глобально.

Экземпляры функциональных блоков могут быть использованы в качестве входных переменных других функциональных блоков или функций.

Замечания.

После выполнения функционального блока все его переменные сохраняются до следующего выполнения. Следовательно, функциональный блок, вызываемый с одними и теми же входными параметрами, может производить различные выходные значения.

Если хотя бы одна переменная функционального блока объявлена как RETAIN, то все данные экземпляров целиком помещаются в энергонезависимый сегмент.

Вызов функционального блока

Для обращения к входным и выходным переменным функционального блока извне необходимо указать имя экземпляра функционального блока, следующей за ней точкой и именем переменной:

<Имя экземпляра>.<Имя переменной >

Присваивание параметров при вызове:

В текстовых языках (IL, ST) задать актуальные параметры и считать значения выходов можно непосредственно при вызове экземпляра функционального блока. Для входных переменных применяется присваивание ":=", выходы считываются при помощи "=>". Этот процесс упрощается, если использовать ассистент ввода (<F2>) с включенной опцией вставки с аргументами (With arguments).

Пример:

Допустим, FBINST - это локальная переменная типа функциональный блок, имеющий входную переменную xx и выходную переменную yy. При вставке FBINST в ST с помощью ассистента ввода получается следующая заготовка:

FBINST1(xx:= , yy=>);

Переменные вход-выход:

Обратите внимание, что переменные вход-выход (VAR_IN_OUT) передаются в экземпляр функционального блока через указатели. Поэтому таким переменным нельзя присваивать константы при вызове.

Пример:

```
VAR
  inst: fubo;
  var: int;
END_VAR
var1:= 2;
inst(instout1 := var1^);
```

не допустимые попытки присваивания констант:

inst(instout1:=2); или inst.inout1:=2;

Примеры вызова экземпляра вышеописанного функционального блока FUB:

Результат умножения сохраняется в переменной ERG, а результат сравнения в переменной QUAD. Экземпляр функционального блока FUB называется INSTANZ.

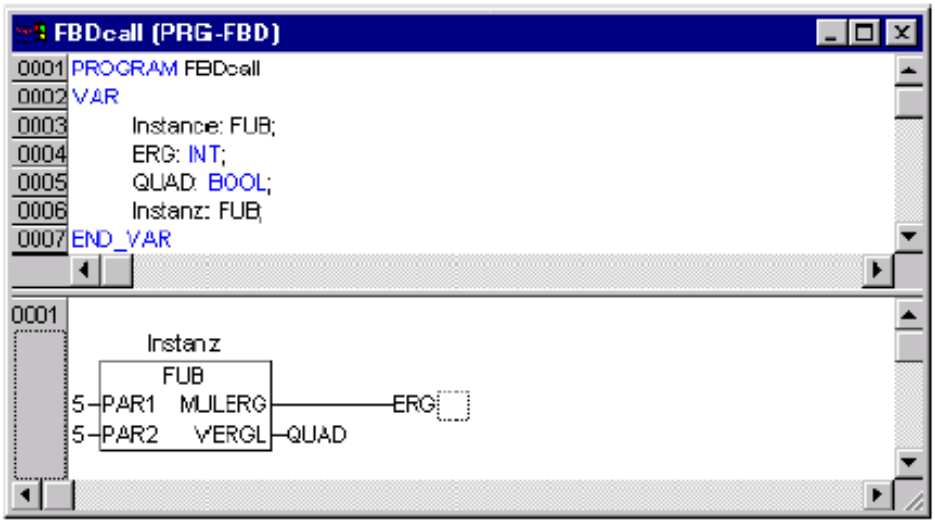
Вызов экземпляра функционального блока в IL:

```
AWLaufruf (PRG-AWL)
0001 PROGRAM AWLaufruf
0002 VAR
0003     QUAD:     BOOL;
0004     INSTANZ:  FUB;
0005     ERG:      INT:=0;
0006 END_VAR
0007

0001     CAL   INSTANZ(PAR1:=5,PAR2:=5)
0002
0003     LD    INSTANZ.VERGL
0004     ST    QUAD
0005
0006     LD    INSTANZ.MULERG
0007     ST    ERG
0008
```

Тот же пример в ST (раздел объявлений такой же, как и в предыдущем примере):

```
STcall (PRG-ST)
0001 PROGRAM STcall

0001 INSTANZ(PAR1:=5,PAR2:=5);
0002 QUAD:=INSTANZ.VERGL;
0003 ERG:=INSTANZ.MULERG;
```

Тот же пример в FBD:

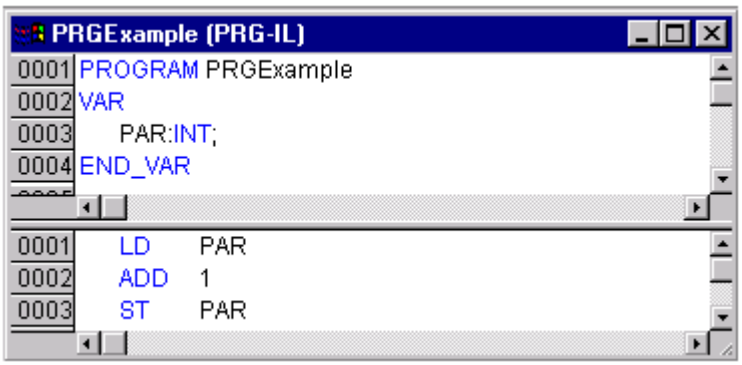


В SFC экземпляры функциональных блоков могут вызываться только из действий шага.

Программа

Программа – это POU, способный формировать произвольное значение во время вычислений. Значения всех переменных программы сохраняются между вызовами. В отличие от функционального блока экземпляров программы не существует. Программа является глобальной во всем проекте.

Пример программы:



Нельзя вызывать программу из функции.

Если вызвать программу, которая изменит значения своих переменных, то при следующем вызове ее переменные будут иметь те же значения, даже если она вызвана из другого POU.

В этом заключается главное различие между программой и функциональным блоком, в котором изменяются только значения переменных данного экземпляра функционального блока.

Объявление программы начинается ключевым словом PROGRAM и заканчивается ключевым словом END_PROGRAM.

Так же, как и для экземпляров функциональных блоков, в текстовых языках (IL, ST) задать актуальные параметры и считать значения выходов можно непосредственно при вызове программы. Для входных переменных применяется присваивание ":=", выходы считываются при помощи "=>".

Пример вызова программы:

IL:

```
CAL  PRGexample2
LD   PRGexample2.out_var
ST   ERG
```

Или с присваиванием параметров:

```
CAL  PRGexample2(in_var:=33, out_var=>erg )
```

ST:

```
PRGexample2;
Erg := PRGexample2.out_var;
```

Или с присваиванием параметров:

```
PRGexample2(in_var:=33, out_var=>erg );
```

FBD:

PLC_PRG

Программа PLC_PRG – это специальный POU, который должен быть в каждом проекте. Эта программа вызывается один раз за цикл управления.

При создании нового проекта автоматически открывается диалог "Project" "Object Add", предлагающий создать новый POU - программу с именем PLC_PRG.

Не следует менять предложенные установки.

Если определить последовательность выполнения задач в Task Configuration, то проект может не содержать PLC_PRG.

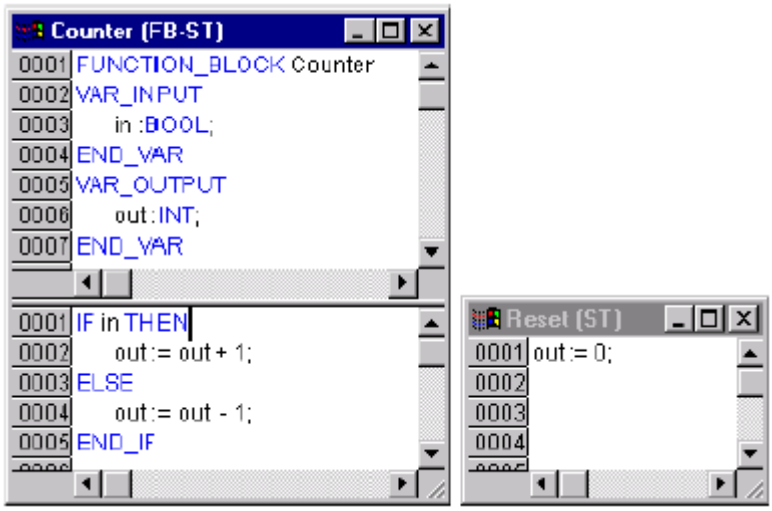
Внимание: Нельзя удалять или переименовывать POU PLC_PRG(если Task Configuration не используется). PLC_PRG является главной программой в однозадачном проекте.

Действие

Программы или функциональные блоки могут быть дополнены действиями. Фактически действия - это дополнительный набор встроенных в POU подпрограмм. Действия могут описываться на языке, отличном от того, на котором выполняется соответствующий функциональный блок или программа.

Действие оперирует с теми же данными, что и функциональный блок или программа, к которой оно принадлежит.

Пример действия функционального блока:



В данном примере функциональный блок Counter увеличивает или уменьшает выходную переменную "out" в зависимости от значения входа "in". При вызове действия Reset выходная переменная принимает значение 0. Одна и та же переменная "out" используется в обоих случаях.

<u>Вызов действия:</u>

Действие вызывается с помощью идентификатора:

<Имя_программы>.<Имя_действия> ии<Имя_экземпляра>.<Имя_действия >.

Если нужно вызвать действие из POU, к которому оно принадлежит, то в текстовых языках используется имя действия, а в графических – функциональный блок без указания имени экземпляра.

Пример вызова вышеописанного действия:

```
PROGRAM PLC_PRG
VAR
   Inst :  Counter;
END_VAR
```

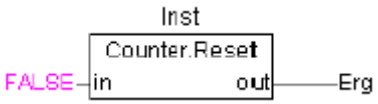
IL:

```
CAL  Inst.Reset(In : = FALSE)
LD    Inst.out
ST    ERG
```

ST:

```
Inst.Reset(In := FALSE);
Erg := Inst.out;
```

FBD:



Ресурсы

Ресурсы отвечают за конфигурацию проекта, включая:

- Глобальные переменные, используемые во всем проекте;
- Менеджер библиотек (Library manager) для подключения необходимых библиотек к проекту;
- Журнал записи действий во время исполнения;
- Конфигуратор тревог (Alarm Configuration) для конфигурирования обработки тревог в проекте;
- Конфигуратор ПЛК (PLC Configuration) для конфигурирования аппаратуры контроллера;
- Конфигуратор задач (Task Configuration) для управления задачами;
- Менеджер рецептов (Watch and Receipt Manager) для просмотра и заказа наборов значений переменных;
- Опции целевой системы (Target Settings).
- Рабочая область для отображения опций проекта;

В зависимости от системы исполнения и ее опций могут подключаться дополнительные объекты:

- Sampling Trace - для задания графической трассировки значений переменных;
- Parameter Manager - для взаимодействия с другими контроллерами в сети;
- PLC-Browser - монитор ПЛК;
- Tools – для вызова внешних, специфичных для каждой платформы инструментов;
- SoftMotion – компоненты системы управления движением (в соответствии с лицензией), редакторы CNC и CAM.

Библиотеки

Проект может использовать несколько библиотек, в которые входят POU, необходимые им типы данных и глобальные переменные. Библиотечные POU можно использовать точно так же, как и определенные пользователем.

Библиотеки "standard.lib" и "util.lib" обязательно входят в стандартный комплект поставки.

Типы данных

Кроме стандартных типов данных, вы можете использовать определяемые пользователем типы данных.

Ими могут быть структуры, перечисления и ссылки.

Визуализация

С помощью визуализации пользователь может создать графическое представление проекта. Форма и цвет графических элементов будут изменяться при работе программы в зависимости от значений переменных.

Визуализация может исполнятся в системе программирования, в отдельном приложении CoDeSys HMI или как Web или целевая (в ПЛК) визуализация.

Лекция 3

3.3 Языки программирования

CoDeSys поддерживает следующие текстовые:

- Instruction List (IL);
- Structured Text (ST);

и графические МЭК языки:

- Sequential Function Chart (SFC);
- Function Block Diagram (FBD);
- Ladder Diagram (LD).

Кроме того, CoDeSys включает поддержку, основанного на Функциональных Блоковых Диаграммах, редактора Continuous Function Chart (CFC).

3.3.1 Список инструкций (IL)

Язык IL (Instruction list) дословно – список инструкций. Каждая инструкция начинается с новой строки и содержит оператор и, в зависимости от типа операции, один и более операндов, разделенных запятыми.

Перед операндом может находиться метка, заканчивающаяся двоеточием (:). Комментарий должен быть последним элементом в строке. Между инструкциями могут находиться пустые строки.

Пример:

```
LD   17
ST   lint        (* комментарий*)
GE   5
JMPC   next
LD   idword
EQ   istruct.sdword
STN   test
next:
```

3.3.2 Модификаторы и операторы IL

В IL можно использовать следующие операторы и модификаторы:

Модификаторы:

C с JMP, CAL, RET: инструкция выполняется только тогда, когда результат аккумулятора ИСТИНА.

N с JMPC,CALC, RETC: инструкция выполняется тогда, когда результат аккумулятора ЛОЖЬ.

N в других случаях: отрицание операнда.

Ниже приведена таблица всех операторов IL с пояснениями и допустимыми модификаторами:

Оператор	**Модифи-катор**	**Значение**
LD	N	Присвоение аккумулятору значения оператора
ST	N	Присвоение значения аккумулятора операнду
S		Присвоить логическому операнду значение ИСТИНА, если значение аккумулятора ИСТИНА
R		Присвоить логическому операнду значение ЛОЖЬ
AND	N, (	Побитное И
OR	N, (	Побитное ИЛИ
XOR	N, (	Побитное исключающее ИЛИ
ADD	(	Сложение
SUB	(	Вычитание
MUL	(	Умножение
DTV	(	Деление
GT	(	>
GE	(	>=
QE	(	=
NE	(	< >
LE	(	<=
LT	(	<
JMP	CN	Переход к метке
CAL	CN	Вызов функционального блока
RET	CN	Выход из POU и возврат в вызывающую программу.

) Вычисление задержанной операции

Пример IL программы с использованием некоторых модификаторов:

```
LD TRUE        (*загрузить значение ИСТИНА в аккумулятор*)
AND    BOOL1   (*выполнить И с инверсным значением переменной  BOOL1*)
JMPC   mark    (*если значение аккумулятора ИСТИНА, то перейти к метке
                " mark"*)
LDN    BOOL2   (*сохранить инверсное значение BOOL2 в аккумуляторе*)
ST  ERG        (*сохранить значение аккумулятора в ERG*)
```

После оператора можно поставить скобки, тогда значение выражения внутри скобок рассматривается как операнд.

Например:

```
LD  2
MUL  2
ADD  3
ST   ERG
```

Здесь значение ERG равно 7. Если поставить скобки, то порядок вычислений изменится:

```
LD  2
MUL  (  2
ADD  3
)
ST  ERG
```

Теперь значение переменной ERG равно 10.

Операция MUL выполняется только тогда, когда программа доходит до ")". В качестве операнда MUL использует значение 5.

3.3.3 Структурированный текст (ST)

ST представляет собой набор инструкций высокого уровня, которые могут использоваться в условных операторах ("IF…THEN…ELSE") и в циклах (WHILE…DO).

Пример:

```
IF value < 7 THEN
WHILE   value < 8 DO
        value:=value+1;
```

```
        END_WHILE;
    END_IF;
```

Выражения

Выражение – это конструкция, возвращающая определенное значение после его вычисления.

Выражение состоит из операторов и операндов. Операндом может быть константа, переменная, функциональный блок или другое выражение.

Вычисление выражений

Вычисление выражений выполняется согласно правилам приоритета. Оператор с самым высоким приоритетом выполняется первым, оператор с более низким приоритетом – вторым и т.д., пока не будут выполнены все операторы.

Операторы с одинаковым приоритетом выполняются слева направо.

В следующей таблице приведен список ST операторов, расположенных в порядке приоритета.

Операция	**Обозначение**	**Приоритет**
Выражение в скобках	(Выражение)	Самый высокий.
Вызов функции	Имя функции (список параметров)	
Возведение в степень	EXPT	
Замена знаков	-	
Числовое дополнение	NOT	
Умножение	*	
Деление	/	
Абсолютная величина	MOD	
Сложение	+	
Вычитание	-	
Сравнение	< , > ,<=, >=	
Неравенство	< >	
Равенство	=	
Логическое И	AND	
Логическое исключающее	XOR	

ИЛИ.

Логическое ИЛИ OR Самый низкий

Ниже приведены примеры использования инструкций ST:

Тип инструкции	**Пример**
Присваивание	A: =B; CV: =CV+1; C: =SIN (X);
Вызов функционального блока и использование FB выхода	A: = CMD_TMR.Q
RETURN	RETURN;

IF	D: =B*B;
	IF D<0.0 THEN
	C: =A;
	ELSIF D=0.0 THEN
	C: =B;
	ELSE
	C: =D;
	END_IF;

CASE	CASE INT1 OF 1: BOOL1: = TRUE; 2: BOOL2: = TRUE; ELSE BOOL1: = FALSE; BOOL2 END_CASE;

FOR	J: =101;
	FOR I: =1 TO 100 BY 2 DO
	IF ARR [I] = 70 THEN
	J: =I;
	EXIT;
	END_IF;
	END_FOR;

WHILE	J: =1; WHILE J<= 100 AND ARR [J] <> 70 DO J: =J+2; END_WHILE;
REPEART	J: =-1; REPEAT J: = J+2; UNTIL J= 101 OR ARR [J] = 70 END_REPEAT;

EXIT EXIT;

Пустая инструкция ;

Оператор присваивания

Перед оператором присваивания находится операнд (переменная или адрес), которому присваивается значение выражения, стоящего после оператора присваивания.

Пример:

Var1: = Var2 * 10;

После выполнения этой операции Var1 принимает значение в десять раз большее, чем Var2.

Вызов функционального блока в ST

Функциональный блок вызывается с помощью имени экземпляра функционального блока и списка входных параметров с присваиванием данных в круглых скобках. В следующем примере вызывается таймер с параметрами IN и PT. Значение выходной переменной Q присваивается переменной A.

К выходной переменной, как и в IL, можно обратиться с помощью имени экземпляра функционального блока, точки, следующей за ним и имени выходной переменной:

```
CMD_TMR (IN: = %IX5, PT: = 300);
A: =CMD_TMR.Q
```

Инструкция RETURN

Инструкция RETURN позволяет выйти из POU, например, в зависимости от условия.

Инструкция IF

Используя инструкцию IF, можно проверить условие, и в зависимости от этого условия выполнить какие-либо действия.

Синтаксис:

```
IF <Boolean_expression1> THEN
  <IF_instructions>
{ELSIF <Boolean_expression2> THEN
```

```
  <ELSIF_instructions1>
.
.
.ELSIF <Boolean_expression n> THEN
  <ELSIF_instructions n-1>
ELSE
  <ELSE_instructions>}
END_IF;
```

Часть конструкции фигурных скобках не обязательна.

Если < <Boolean_expression1> возвращает истину, тогда <IF_Instructions> выполняется.

В противном случае будут выполняться остальные логические выражения одно за другим, пока одно из них не возвратит истину. Тогда выполняются инструкции, стоящие после этого логического выражения до следующего ELSIF или ELSE.

Если все логические выражения ложны, то выполняются инструкции, стоящие после ELSE.

Пример:

```
IF temp < 17
THEN   heating_on: = TRUE;
ELSE   heating_on: = FALSE;
END_IF
```

В этом примере нагревание (heating) включается, когда температура опустится ниже 17º градусов, иначе оно останется выключенным.

Инструкция CASE

С помощью инструкции CASE можно нескольким различным значениям целочисленной переменной сопоставить различные инструкции.

Синтаксис:

```
CASE <Var1> OF
<Value1>:  <Instruction 1>
<Value2>:   <Instruction 2>
<Value3, Value4, Value5>:   <Instruction 3>
<Value6 .. Value10>:    <Instruction 4>
...
<Value n>:   <Instruction n>
ELSE   <ELSE instruction>
END_CASE;
```

Инструкция CASE выполняется согласно следующим правилам:

- Если переменная <Var1> имеет значение <Value i>, то выполняется инструкция <Instruction i>;
- Если <Var1> не принимает ни одного из указанных значений, то выполняется <ELSE Instruction>;
- Чтобы одна и та же инструкция выполнялась при различных значениях переменной <Var1>, необходимо перечислить эти значения через запятую;
- Чтобы одна и та же инструкция выполнялась для целого диапазона значений, необходимо указать начальное и конечное значения, разделенные двумя точками.

Пример:

```
CASE INT1 OF
1, 5: BOOL1 : = TRUE;
    BOOL3 : = FALSE;
2:  BOOL2 : = FALSE;
    BOOL3 : = TRUE;
10. 20: BOOL1: = TRUE;
   BOOL3:= TRUE;
ELSE
BOOL1 := NOT BOOL1;
BOOL2 := BOOL1 OR BOOL2;
END_CASE;
```

Цикл FOR

С помощью FOR можно программировать повторяющиеся процессы.

Синтаксис:

```
INT_Var :INT;

FOR <INT_Var> := <INIT_VALUE> TO <END_VALUE> {BY <Step size>} DO
   <Instructions>
END_FOR
```

Часть конструкции, заключенная в фигурные скобки, не обязательна. <Instructions> выполняются, пока счетчик <INT_Var> не больше <END_VALUE>. Это условие проверяется перед выполнением <Instructions>, поэтому раздел <Instructions> не выполняется, если <INIT_VALUE> больше <END_VALUE>.

Всякий раз, когда выполняются <Instructions>, значение <INIT_VALUE>, увеличивается на <Step_ size>.

<Step_ size> может принимать любое целое значение. По умолчанию шаг устанавливается равным 1.

Пример:

```
FOR Counter: =1 TO 5 BY 1 DO
Var1: =Var1*2;
END_FOR;
Erg:=Var1;
```

В этом примере предполагается, что начальное значение Var1 равно 1. После выполнения цикла эта переменная будет равна 32.

Замечание: <END_VALUE>: не должно быть равно предельному значению счетчика <INT_VAR>. Например, если счетчик является переменной типа SINT и <END_VALUE> равно 127, то цикл становится бесконечным.

Цикл WHILE

Цикл WHILE может использоваться, как и цикл FOR, с тем лишь различием, что условие выхода определяется логическим выражением. Это означает, цикл выполняется, пока верно заданное условие.

Синтаксис:

```
WHILE <Boolean expression>
<Instructions>
END_WHILE
```

Раздел <Instructions> выполняется циклически до тех пор, пока <Boolean_expression> дает TRUE. Если <Boolean_expression> равно FALSE уже при первой итерации, то раздел <Instructions> не будет выполнен ни разу. Если <Boolean_expression> никогда не примет значение FALSE, то раздел <Instructions> будет выполняться бесконечно.

Замечание: Программист должен быть уверен, что цикл не станет бесконечным. Для этого в теле цикла значение входящей в условие переменной обязательно должно изменяться. Например, путем инкремента или декремента счетчика.

Пример:

```
WHILE counter<>0 DO
Var1: = Var1*2;
counter := counter-1;
```

END_WHILE

Цикл REPEAT

Цикл REPEAT отличается от цикла WHILE тем, что первая проверка условия выхода из цикла осуществляется, когда цикл уже выполнился 1 раз. Это означает, что независимо от условия выхода цикл выполняется хотя бы один раз.

Синтаксис:

REPEAT
<Instructions>
UNTIL <Boolean expression>
END_REPEAT

Раздел <Instructions> выполняется циклически до тех пор, пока <Boolean_expression> дает TRUE. Если <Boolean_expression> равно FALSE уже при первой итерации, то раздел <Instructions> не будет выполнен один раз. Если <Boolean_expression> никогда не примет значение FALSE, то раздел <Instructions> будет выполняться бесконечно.

Замечание: Программист должен быть уверен, что цикл не станет бесконечным. Для этого в теле цикла значение входящей в условие переменной обязательно должно изменяться. Например, путем инкремента или декремента счетчика.

Инструкция EXIT

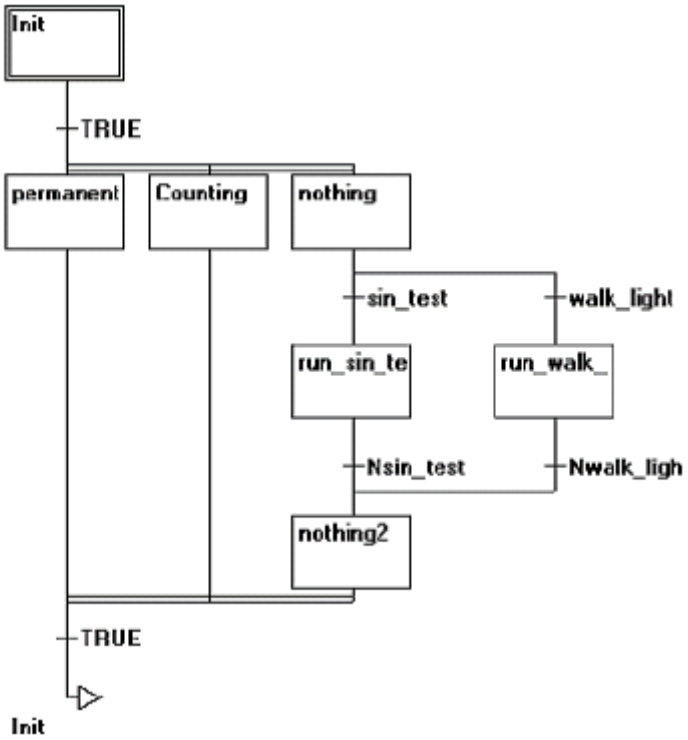
Если EXIT встречается в циклах FOR, WHILE, REPEAT, то цикл заканчивает свою работу независимо от значения условия выхода.

Лекция 4

3.3.4 Язык последовательных функциональных схем (SFC)

SFC – это графический язык, который позволяет описать хронологическую последовательность различных действий в программе. Для этого действия связываются с шагами (этапами), а последовательность работы определяется условиями переходов между шагами.

Пример SFC диаграммы:



Шаг

SFC POU состоит из набора шагов, связанных переходами.

Существуют 2 вида шагов:

- Шаг простого типа (упрощенный SFC) может включать единственное действие. Графический флажок (небольшой треугольник в верхнем углу шага) показывает, пустой шаг или нет.
- МЭК шаг (стандартный SFC) связан с произвольным числом действий или логических переменных.

Связанные действия располагаются с правой стороны от шага.

Действие

Действие может содержать список инструкций на IL или ST, схемы на FBD или LD, или снова схемы на SFC.

При использовании простых шагов действие всегда связывается с этим шагом. Для того, чтобы редактировать действие, необходимо дважды щелкнуть левой клавишей мышки на шаге. Или выделить шаг и выбрать команду меню "Extras" "ZoomAction/Transition". Помимо основного действия, шаг может включать одно входное и одно выходное действие.

Действия МЭК шагов показаны в Организаторе Объектов, непосредственно под вызывающей их POU.

Редактирование действия запускается двойным щелчком мыши или клавишей <Enter>. Новые действия добавляются командой главного меню "Project" "Add Action". Вы можете сопоставить одному шагу до 9 действий.

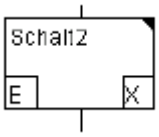
Входное или выходное действие

В шаг можно добавить входное и выходное действие.

Входное действие выполняется один раз при активизации шага, выходное – при деактивизации. Шаг, который имеет входное действие, обозначается буквой "E" в левом нижнем углу, шаг с выходными действиями – буквой "X" в правом нижнем углу.

Входные и выходные действия могут описываться на любом языке. Для того чтобы отредактировать входное или выходное действие, надо дважды щелкнуть мышкой в соответствующем углу шага.

Пример простого шага с входным и выходным действиями:



Переход/условие перехода

Между шагами находятся так называемые переходы. Условием перехода может быть логическая переменная или константа, логический адрес или логическое выражение, описанное на любом языке. Условие может включать серию инструкций, образующих логический результат, в виде ST выражения (т.е. (i<= 100) AND b) либо на любом другом языке. Но условие не должно содержать присваивания, вызов программ и экземпляров функциональных блоков!

В редакторе SFC условие перехода можно записать непосредственно около символа перехода либо в отдельном окне редактора для ввода условия (См. раздел 5.4.4,'Extras' 'Zoom Action/Transition'). Условие заданное окне редактора предпочтительнее!

Замечание: помимо условий переходов, можно использовать тактируемый режим переходов.

Активный шаг

После вызова SFC POU начальный шаг (шаг, выделенный двойной рамкой) выполняется первым. Шаг, выполняемый в данный момент, называется активным. Действия, связанные с активным шагом, выполняются один раз в каждом управляющем цикле. В режиме online активные шаги выделяются синим цветом. Следующий за активным шагом шаг станет активным, только когда условие перехода к этому шагу будет истинно.

В каждом управляющем цикле будут выполнены действия, содержащиеся в активных шагах. Далее проверяются условия перехода, и, возможно, уже другие шаги становятся активными, но выполняться они будут уже в следующем цикле.

Замечание: выходное действие выполняется однократно в следующем цикле, после того, как условие перехода станет истинным.

Шаг МЭК

В отличие от упрощенного SFC МЭК шаги могут включать несколько действий. Действия МЭК шагов описываются отдельно от них и могут неоднократно использоваться в пределах данного POU, для чего их надо связать с шагом с помощью команды главного меню "Extras" "Associate action".

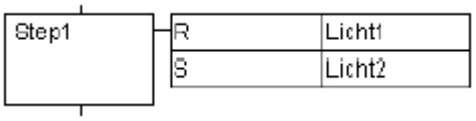
Кроме действий с шагом, можно связывать логические переменные.

С помощью так называемых классификаторов действия и логические переменные могут активироваться и деактивироваться, возможно, с задержкой времени. Например: действие может продолжать работу, даже если запустивший его шаг утратил активность; с помощью классификатора S (установка) можно программировать параллельные процессы и т.д.

Логическая переменная, связанная с шагом, получает значение ИСТИНА при каждой активации шага.

Действие, связанное с МЭК шагом, описывается справа от него в блоке, состоящем из двух частот. Левая часть этого блока содержит классификатор, возможно, с константой времени, а правая часть содержит имя действия или логической переменной.

Пример МЭК шага с двумя действиями:



В режиме online все активные действия выделяются синим цветом, подобно активным шагам. Благодаря чему достаточно легко проследить ход выполнения процесса после каждого управляющего цикла.

Замечание: Если действие деактивируется, то оно выполняется еще один раз. Это означает, что каждое действие выполняется хотя бы два раза.

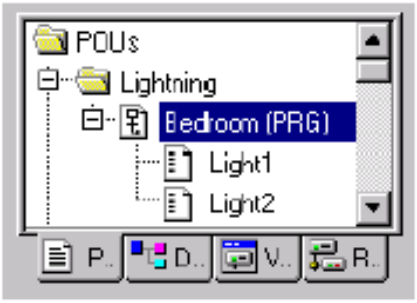
При выполнении шага сначала производится деактивация действий, затем выполняются активные действия в алфавитном порядке.

Для того чтобы использовать шаги с МЭК действиями, необходимо установить опцию "Extras" "Use IEC-Steps" и подключить к проекту специальную библиотеку Iecsfc.lib.

В Организаторе объектов действия показаны непосредственно под SFC POUs, которые их вызывают.

Новые действия можно создавать с помощью команды "Project" "Add Action".

SFC POU со списком действий в Организаторе Объектов:



Классификаторы действий

Для связи шагов и действий применяются классификаторы:

N	Не сохраняемое	Действие активно в течение активности шага
R	Внеочередной Сброс	Деактивация действия
S	Установка	Действие активно вплоть до сброса
L	Ограниченное по времени	Действие активно в течение указанного времени, но не дольше времени активности шага
D	Отложенное	Действие активируется по прошествии указанного времени, если шаг еще активен и продолжает быть активным
P	Импульс	Действие выполняется один раз, если шаг активен
SD	Сохраняемое и отложенное	Действие активно после указанного времени до сброса
DS	Отложенное и сохраняемое	Действие активно после указанного времени, если шаг еще активен, вплоть до сброса

SL	Сохраняемое и ограниченное по времени	Активно после указанного времени.

Классификаторы L, D, SD, DS, SL требуют указания временной константы в формате TIME.

Замечание: В процессе деактивации действие выполняется еще один раз. Это относится и к действиям с классификатором P!

Неявные переменные в SFC

В SFC существуют неявно объявленные переменные, которые могут оказаться полезны.

Каждому шагу соответствует флаг, который хранит информацию о его состоянии. Этот флаг обозначается <StepName>.x для МЭК шагов и<StepName> для простых шагов. Этот флаг имеет значение ИСТИНА, когда соответствующий шаг активен, и ЛОЖЬ, когда неактивен. Он может использоваться в любом действии или переходе SFC.

Аналогично, с помощью переменной <ActionName>.x, можно узнать, активно действие или нет. С помощью неявной переменной <StepName>.t можно узнать время активности МЭК шага.

К неявным переменным разрешен доступ даже из другой программы.

Например: boolvar1:=sfc1.step1.x; где step1.x - неявная логическая переменная, представляющая состояние МЭК шага step1 в POU sfc1.

Флаги SFC

Для управления работой SFC компонента предусмотрены специальные флаги. Для использования этих флагов необходимо объявлять их глобально или локально как входные или выходные переменные.

Пример: Если в SFC POU некоторый шаг активен дольше, чем время, заданное в его атрибутах, устанавливается специальный флаг, доступный через переменную "SFCError" (SFCError принимает значение TRUE в этом случае).

Вы можете использовать следующие переменные-флаги:

SFCEnableLimit: Переменная типа BOOL. Когда значение этой переменной ИСТИНА, задержка времени шага регистрируется в SFCError. Другие задержки времени игнорируются.

SFCInit: Переменная типа BOOL. Когда переменная получает значение ИСТИНА, программа переходит обратно на шаг Init и все SFC флаги сбрасываются. Шаг Init становится активным, но не выполняется, пока переменная имеет значение ИСТИНА. Как только SFCInit примет значение ЛОЖЬ, выполнение программы продолжится.

SFCReset: Переменная типа BOOL. Работает подобно SFCInit. Но последующее выполнение будет продолжено после шага инициализации Init. Поэтому флаг SFCReset можно сбросить в FALSE в самом шаге Init.

SFCQuitError: Переменная типа BOOL. Выполнение программы SFC приостанавливается, пока переменная имеет значение ИСТИНА. Посредством чего сбрасывается возможный таймаут. Время работы любого активного шага сбрасывается, когда эта переменная снова получает значение ЛОЖЬ.

SFCPause: Переменная типа BOOL. Выполнение программы SFC приостанавливается, пока эта переменная имеет значение ИСТИНА.

SFCError: Эта логическая переменная принимает значение ИСТИНА, когда происходит задержка времени. Если следом возникает вторая ошибка, она не фиксируется, пока флаг SFCError не будет сброшен. Для уточнения причины ошибки необходимо использовать флаги: SFCErrorStep, SFCErrorPOU, SFCQuitError, SFCErrorAnalyzation.

SFCTrans: Переменная типа BOOL. Принимает значение ИСТИНА, когда переход активируется.

SFCErrorStep: Переменная типа STRING. В этой переменной хранится имя шага, в котором обнаружена ошибка (задержка времени).

SFCErrorPOU: Переменная типа STRING. В этой переменной хранится имя компонента, в котором обнаружена ошибка (задержка времени).

SFCCurrentStep: Переменная типа STRING. В этой переменной хранится имя активного шага. В случае одновременного выполнения шагов в переменной сохраняется имя того шага который находится в правой ветви SFC диаграммы.

SFCErrorAnalyzationTable: Переменная типа ARRAY [0..n] OF ExpressionResult сообщает результаты анализа условного выражения перехода. Для каждого элемента выражения, формирующего значение FALSE и соответственно задерживающего переход, заполняется структура, содержащая наименование, адрес, комментарий и текущее значение.

Допускается максимум до 16 элементов (переменных), поэтому индексы массива имеют значения от 0 до 15.

Структура ExpressionResult и неявно используемые функции анализа включены в библиотеку AnalyzationNew.lib. Модули библиотеки можно использовать явно в других POU, не запрограммированных в SFC.

Предварительным условием анализа является обнаружение задержки в шаге. Поэтому контроль времени выполнения обязателен при анализе. Также обязательно должна быть объявлена переменная SFCError.

SFCTip, SFCTipMode: Переменные типа BOOL позволяют задать тактируемый режим выполнения SFC. Если этот режим включен – SFCTipMode=TRUE, то переход к следующему шагу возможен только при SFCTip, равном TRUE. Пока SFCTip имеет значение FALSE, переход не разрешен, даже если условие выполнено.

Альтернативная ветвь

Две и более ветви SFC могут быть альтернативными. Каждая альтернативная ветвь должна начинаться и заканчиваться переходом. Альтернативные ветви могут содержать параллельные ветви и другие альтернативные ветви. Альтернативная ветвь начинается горизонтальной линией (начало альтернативы), а заканчивается горизонтальной линией (конец альтернативы) или переходом на произвольный шаг (jump). Если шаг, который находится перед линией альтернативного начала, активен, то

первые переходы альтернативных ветвей начинают оцениваться слева направо.

Таким образом, первым активируется тот шаг, который следует за первым слева истинным переходом.

Параллельные ветви

Две и более ветви SFC могут быть параллельными.

Каждая параллельная ветвь должна начинаться и заканчиваться шагом. Параллельные ветви могут содержать альтернативные ветви и другие параллельные ветви. Параллельная ветвь наносится двойной горизонтальной линией (параллельное начало) и заканчивается двойной горизонтальной линией (конец параллели) или переходом на произвольный шаг (jump).

Если шаг активен, условие перехода после этого шага истинно и за этим переходом следуют параллельные ветви, то активируются первые шаги этих ветвей. Эти ветви выполняются параллельно друг другу. Шаг, находящийся после параллельных ветвей, становится активным только тогда, когда все предыдущие шаги активны и условие перехода истинно.

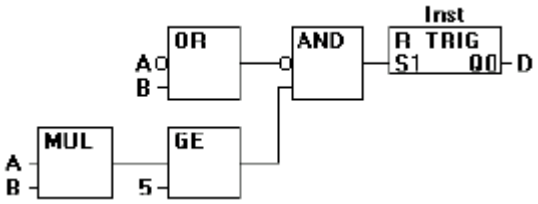
Переход на произвольный шаг (Jump)

Переход на произвольный шаг - это соединение на шаг, имя которого указано под знаком «jump». Такие переходы нужны для того, чтобы избежать пересекающихся и идущих вверх соединений.

3.3.5 Язык функциональных блоковых диаграмм (FBD)

FBD – это графический язык программирования. Он работает с последовательностью цепей, каждая из которых содержит логическое или арифметическое выражение, вызов функционального блока, переход или инструкцию возврата.

Вот пример типичной FBD схемы в CoDeSys:



Лекция 5

3.3.6 Непрерывные функциональные схемы (CFC)

В отличие от FBD редактор непрерывных функциональных схем не использует цепи, но дает возможность свободно размещать компоненты и соединения, что позволяет создавать обратные связи, как показано в примере.

Пример CFC:

3.3.7 Язык релейных диаграмм (LD)

Язык релейных или релейно-контактных схем (РКС) – графический язык, реализующий структуры электрических цепей.

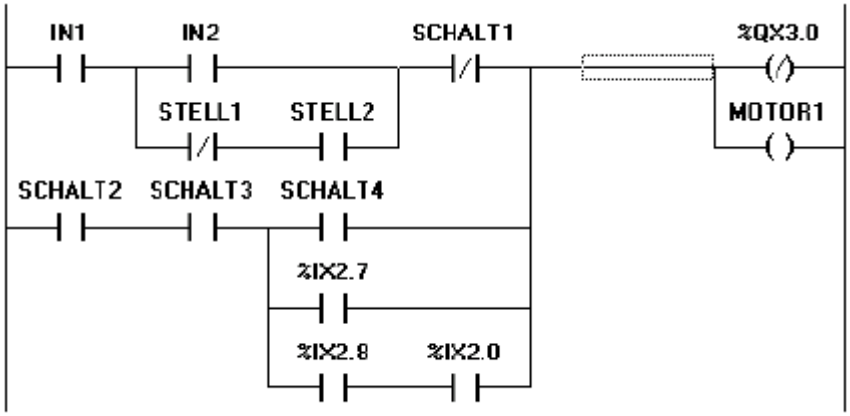
Лучше всего LD подходит для построения логических переключателей, но достаточно легко можно создавать и сложные цепи - как в FBD. Кроме того, LD достаточно удобен для управления другими компонентами POU.

Диаграмма LD состоит из ряда цепей.

Слева и справа схема ограничена вертикальными линиями - шинами питания. Между ними расположены цепи, образованные контактами и обмотками реле, по аналогии с обычными электронными цепями.

Слева любая цепь начинается набором контактов, которые посылают слева направо состояние "ON" или "OFF", соответствующие логическим значениям ИСТИНА или ЛОЖЬ. Каждому контакту соответствует логическая переменная. Если переменная имеет значение ИСТИНА, то состояние передается через контакт. Иначе правое соединение получает значение выключено ("OFF").

Пример типичной LD цепи:



Контакт

Контакты обозначаются двумя параллельными линиями и могут иметь состояния "ON" или "OFF". Эти состояния соответствуют значениям ИСТИНА или ЛОЖЬ. Каждому контакту соответствует логическая переменная. Если значение переменной ИСТИНА, то контакт замкнут.

Контакты могут быть соединены параллельно, тогда соединение передает состояние "ON", когда хотя бы одна из ветвей передает "ON".

Если контакты соединены последовательно, то для того, чтобы соединение передало "ON", необходимо, чтобы оба контакта передавали "ON". Это соответствует электрической параллельной и последовательной схеме.

Контакт может быть инвертируемым. Такой контакт обозначается с помощью символа |/| и передает состояние "ON", если значение переменной ЛОЖЬ.

Обмотка

В правой части схемы может находиться любое количество обмоток (реле), которые обозначаются круглыми скобками (). Они могут соединяться только параллельно. Обмотка передает значение соединения слева направо и копирует его в соответствующую логическую переменную. В целом цепь может быть либо замкнутой (ON), либо разомкнутой (OFF). Это как раз и отражается на обмотке и соответственно на логической переменной обмотки (ИСТИНА/ЛОЖЬ).

Обмотки также могут быть инверсными (в примере - %QX3.0). Если обмотка инверсная (обозначается символом (/)), тогда в соответствующую логическую переменную копируется инверсное значение.

Функциональные блоки в LD

Кроме контактов и обмоток, в LD можно использовать функциональные блоки и программы. Они должны иметь логические вход и выход и могут использоваться так же, как контакты.

SET и RESET обмотка

Обмотки могут быть с «самофиксацией» типов SET и RESET. Обмотки типа SET обозначаются буквой "S" внутри круглых скобок (S). Если соответствующая этой обмотке переменная принимает значение ИСТИНА, то она навсегда (до сброса R) сохраняет его.

Обмотки типа RESET обозначаются буквой R. Если соответствующая переменная принимает значение ЛОЖЬ, то она навсегда (до установки S) сохраняет его.

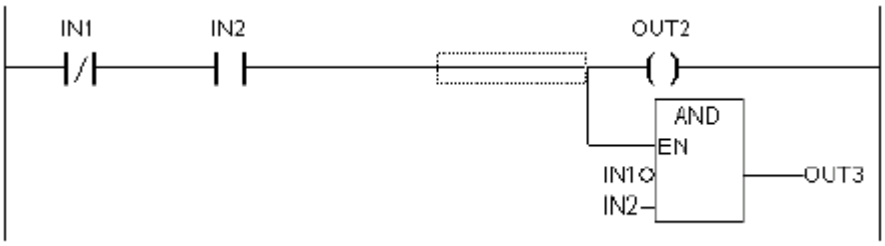
LD в качестве FBD

Весьма вероятно, что при работе с LD вы захотите с помощью контакта управлять другими POU.

Во-первых, можно использовать обмотку для передачи значения глобальной переменной, которая будет использоваться в другом месте. Кроме того, можно вставить вызов прямо в схему LD.

Такой POU может быть оператором, функцией, программой или функциональным блоком, который имеет добавочный вход, обозначаемый EN. Вход EN всегда логического типа, и POU выполняется, только когда значение EN=ИСТИНА. POU встраивается в схему параллельно обмоткам, и вход EN соединяется ответвлением. Использование таких POU делает LD схему похожей на FBD схему.

Пример LD цепи с EN POU:



CoDeSys поддерживает все операторы стандарта МЭК 61131-3:

- Оператор присваивания;
- Битовые операторы;
- Сдвиговые операторы;
- Операторы сравнения;
- Арифметические операторы;
- Работа с действительными числами;
- Логарифмические операторы;
- Тригонометрические операторы;
- Операторы выбора;

Операторы для различных языков программирования в CoDeSys приведены в приложении Б.

В МЭК 61131-3 определенны следующие стандартные типы данных:

Таблица 6 - Стандартные типы данных

Ключевое слово	**Диапазон**	**Пример**
BOOL	0 , 1	FALSE, TRUE, 0, 1
SINT, INT, DINT	-128 .. 127, -32768 .. 32767, -2147483648 .. 2147483647	0, 24453 -38099887
USINT, UINT, UDINT	0 .. 255, 0 .. 65535,	200, 47453

	0 .. 4294967295	138099887
BYTE, WORD, DWORD	0 .. 255, 0 .. 65535, 0 .. 4294967295	8450 16#2102
REAL, LREAL	-1.2x 10-38 .. 3.4x 1038 -2.3x 10-308 .. 1.7x 10308	1.34996 2.8377E-15
TIME, TOD, DATE, DT	0 ms .. 1193h2m47s295ms 00:00:00 .. 23:59:59 01.01.1970 до. 06.02.2106	T#1d8h12m8s125ms TOD#12:34:17 D#2001-03-15 DT#2001-03-15- 12:17:03
STRING	1 .. 255 символов	`Emergency Stop`

3.4 Отладка и online функции

Трассировка.

Sampling Trace позволяет отображать последовательные значения переменных, записанные периодически, с возможной привязкой к событию, заданному так называемым триггером трассировки. Sampling Trace обладает способностью трассировать до 20 переменных одновременно. Для каждой переменной сохраняются 500 последних значений.

Отладка

Специальная опция отладки CoDeSys заставляет компилятор формировать дополнительный код, упрощающий поиск ошибок. Опция включается через меню "Project" "Options" в диалоговом окне "Debugging".

Точки останова

Точки останова – это места, в которых выполнение программы будет приостанавливаться, что позволяет просмотреть значения переменных на определенном этапе работы программы. Точки останова можно задавать во всех редакторах. В текстовом редакторе точка останова устанавливается на номер строки, в FBD и LD - на графический элемент и в SFC - на шаг.

Пошаговое выполнение

Под «шагом» подразумевается:

- В IL: Выполнить программу до следующего оператора CAL, LD or JMP.
- В ST: Выполнить следующую инструкцию.
- В FBD, LD: Выполнить следующую цепь.
- В SFC: Продолжить действие до следующего шага.

Пошаговое выполнение позволяет проверить логическую правильность программы.

Выполнение по циклам

Команда одинарный цикл (Single Cycle) выполняет один рабочий цикл и останавливает контроллер после выполнения.

Изменения значений переменных в режиме Online.

Изменив значение переменной в режиме online, вы можете однократно записать его в контроллер (запись значений) либо включить режим записи заданных значений в каждом рабочем цикле (фиксация значений).

Для изменения текущего значения переменной щелкните по нему дважды мышкой. Если это логическая переменная, то она изменит свое значение на противоположное, если же эта переменная любого другого типа, то новое значение переменной нужно будет ввести в диалоговом окне (Write Variable).

Мониторинг

В режиме online значения всех видимых на экране переменных отображаются CoDeSys в реальном времени. Вы можете наблюдать актуальные значения переменных в разделах объявлений и кода редакторов, в менеджере «рецептов» (watch and receipt manager) и на экранах визуализации. Значения переменных экземпляров функциональных блоков отображаются в виде иерархического дерева.

Для указателей, в разделе кода, отображаются только их собственные значения. В разделе объявлений вы можете «раскрыть» указатель и увидеть значение адресуемой им переменной.

Пример мониторинга указателей:

```
PLC_PRG (PRG-ST)
0001 PROGRAM PLC_PRG
0002 VAR
0003     ppa:POINTER TO POINTER TO atype;
0004     pa:POINTER TO atype;
0005     var1:DWORD;
0006     var2:DWORD;
0007     avar:atype;
0008     b:BOOL;
0009     str1:STRING:='ja';
0010     str: STRING:='nein';
0011     str2: STRING;
0012 END_VAR
```

```
atype
0001 TYPE atype :
0002 STRUCT
0003     a:POINTER TO INT;
0004     b:INT;
0005 END_STRUCT
0006 END_TYPE
```

```
PLC_PRG (PRG-ST)
0001 ⊟···ppa = <014d8ee4>
0002     ⊟···ppa^ = <014d8ef0>
0003         ⊟···ppa^^
0004             ⊟···.a = <014d8ef4>
0005                 ·····.a^ = 0
0006             ·····.b = 0
0007 ⊟···pa = <014d8ef0>
0008     ⊟···pa^
0009         ⊟···.a = <014d8ef4>
0010             ·····.a^ = 0
0011         ·····.b = 0
0012     var1 = 0
0013     var2 = 0
0014 ⊟···avar
0015     ⊟···.a = <014d8ef4>
0016         ·····.a^ = 0
0017     ·····.b = 0
0018     b = FALSE
0019     str1 = 'ja'
0020     str = ''
0021     str2 = ''
```

```
0001 avar.a:=ADR(avar.b);        avar.a = <014d8ef4>
0002 pa:=ADR(avar);              pa = <014d8ef0>
0003 ppa :=ADR(pa);              ppa = <014d8ee4>
0004 IF b THEN                   b = FALSE
0005 str:=str1;                  str = ''
0006 ELSE
0007 str:=str2;                  str = ''
0008 END_IF
```

При мониторинге массивов отображаются не только элементы с константой в индексе, но и элементы с индексной переменной:

anarray[1] = 5

anarray[i] = 1

Значение элемента массива не отображается только в случае, если в качестве индекса задано выражение (например [i + j] или [I*2]).

Внимание: Если превышено число одновременно отображаемых переменных, то в строке значений переменных вы увидите сообщение: "Too many monitoring variables".

Эмуляция

Во время эмуляции созданная программа выполняется не в ПЛК, а в компьютере, на котором запущен CoDeSys. В этом режиме допустимы все функции online, что позволяет проверить логическую правильность программ, не используя контроллер.

Внимание: функции внешних библиотек не выполняются в режиме эмуляции.

Бортжурнал

Бортжурнал (log) хронологически записывает действия пользователя, внутренние процессы, изменения состояния и исключения в режиме online. Это необходимо при анализе условий возникновения ошибки при отладке.

4 Контрольные вопросы к лабораторной работе

1 Что такое ПЛК. Какие модификации ПЛК 150 вы знаете?

2 Изобразите схему подключения к ОВЕН ПЛК 150.

3 Что называют «безопасным состоянием» ПЛК?

4 С помощью какой системы осуществляют программирование ПЛК 150?

5 Из каких частей состоит базовая пользовательская документация по CoDeSys?

6 Перечислите компоненты проекта CoDeSys.

7 Какие текстовые языки программирования поддерживает CoDeSys?

8 Какие графические языки программирования поддерживает CoDeSys?

9 Для чего необходима «точка останова»?

10 Расскажите подробнее об «online функции».

5 Пример

Рассмотрим первые шаги для программирования в CoDeSys:

1. Для создания нового проекта необходимо в среде CoDeSys вызвать команду меню File | New или воспользоваться одноименной кнопкой на панели инструментов.

2. После создания проекта нужно выбрать Target-файл, соответствующий названию контроллера. Окно выбора Target-файла представлено на рис. 4

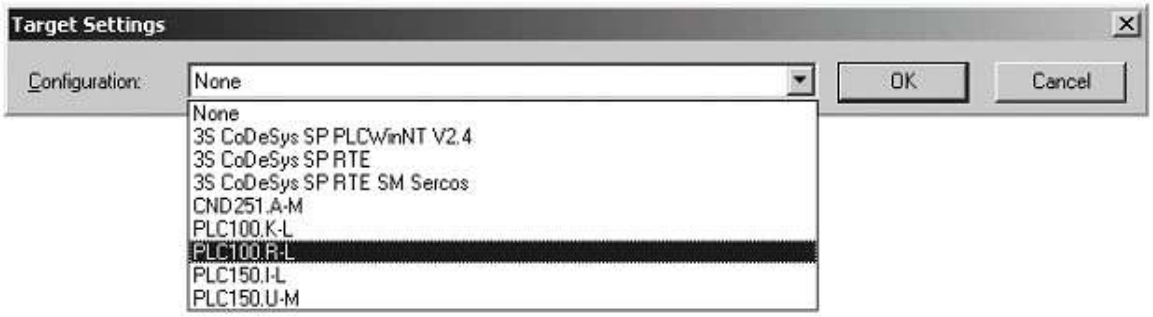


Рисунок 4 - Окно выбора Target-файла

3. Затем откроется окно настроек Target-файлов. Как правило, настройки установлены производителем и не требуют изменения.
4. После задания настроек Target-файла необходимо создать основной POU (главную программу проекта). Окно этого диалога представлено на рис. 5. Главная программа всегда должна иметь тип Program и имя PLC_PRG. Поэтому в данном диалоге разрешается менять только язык программирования (Language of the POU).

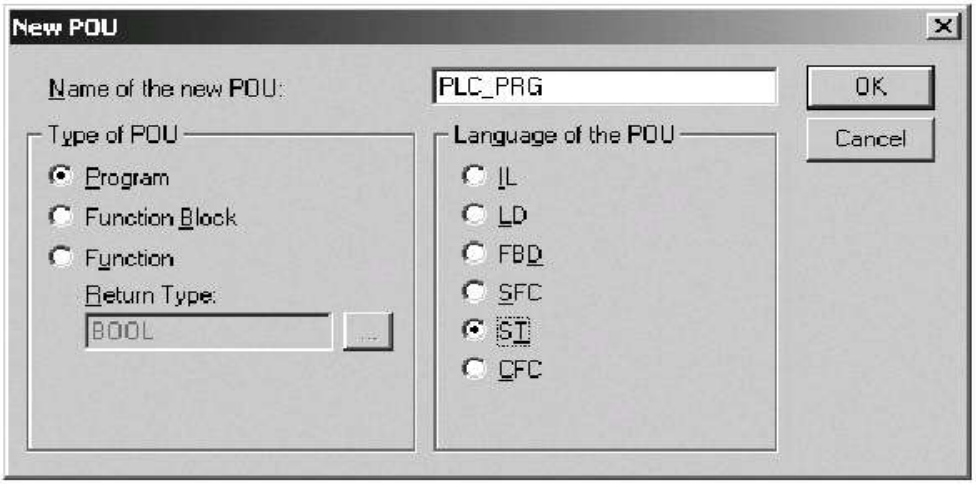


Рисунок 5 - Окно создания основного POU

5. В зависимости от выбранного языка программирования откроется окно, в котором необходимо создать программу, исполняемую на контроллере. Примеры программ на языках FBD, LD и ST приведены на рис. 6

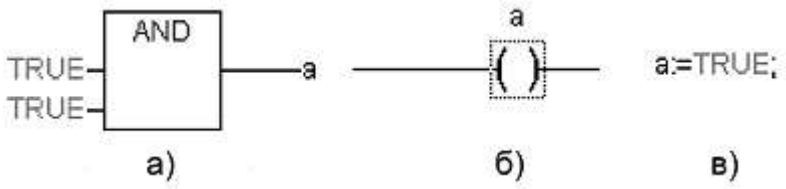


Рисунок 6 - Примеры программ на языках FBD (а), LD (б) и ST (в)

При написании любого из примеров программ, представленных на рис. 3, будет вызван ассистент ввода (рис. 7) для описания переменной **a**.

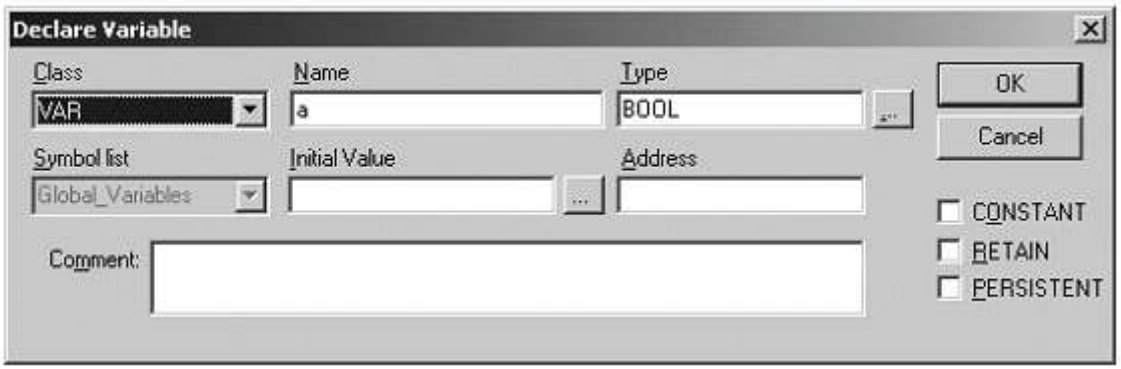


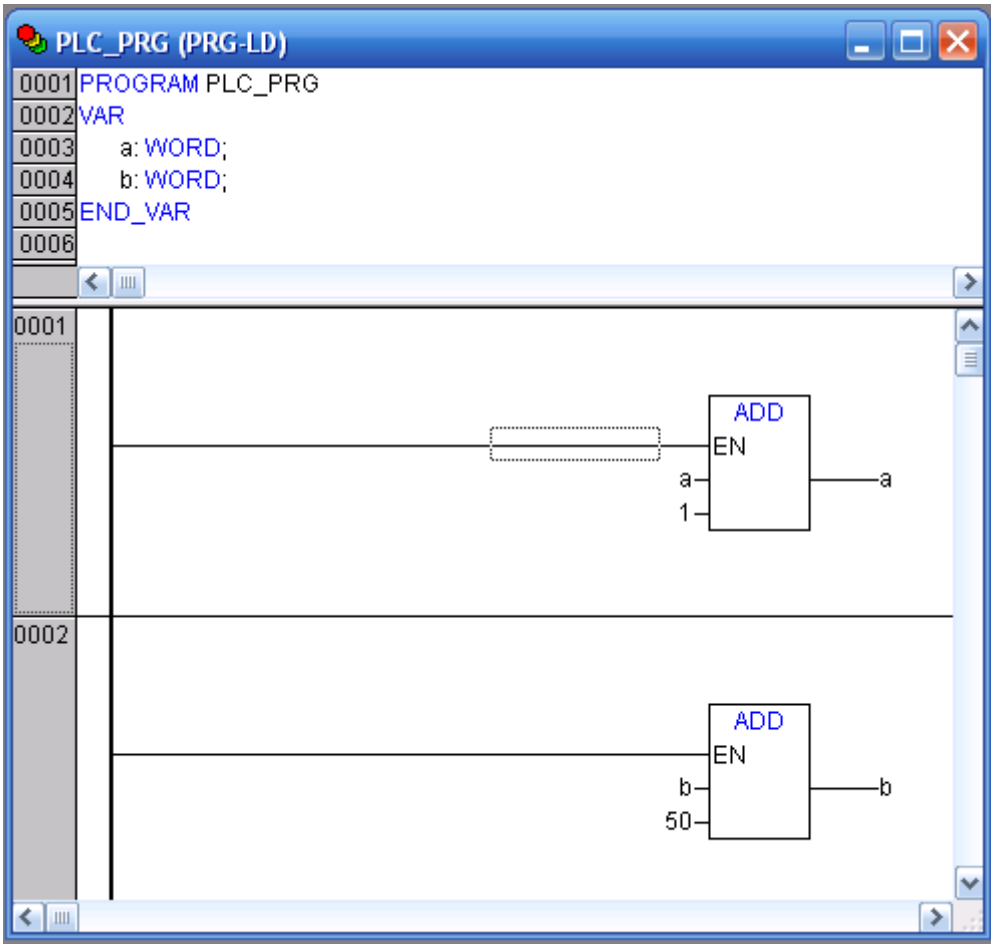
Рисунок 7 - Окно объявления переменной

6. Для загрузки программы в контроллер установите связь с контроллером, вызвав команду меню Online | Login. Запустите выполнение загруженной программы, вызвав команду меню Online | Run или нажатием кнопки на передней панели контроллера.

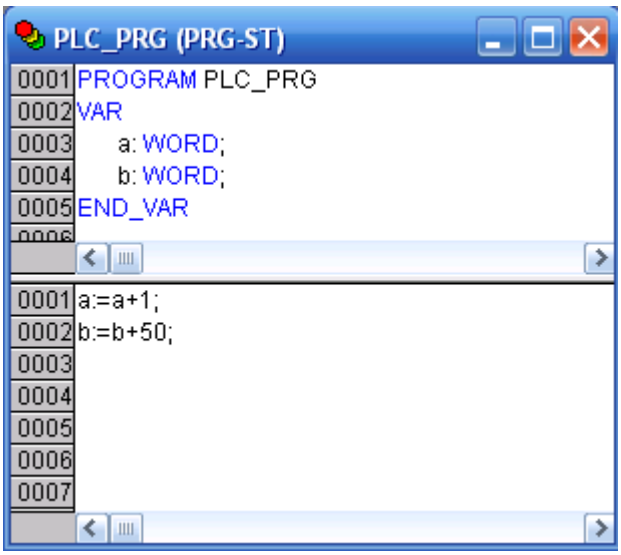
Для примера приведены следующие упражнения:

Упражнение 1. Оператор сложения:

Пример функции сложения, написанной на языке LD:

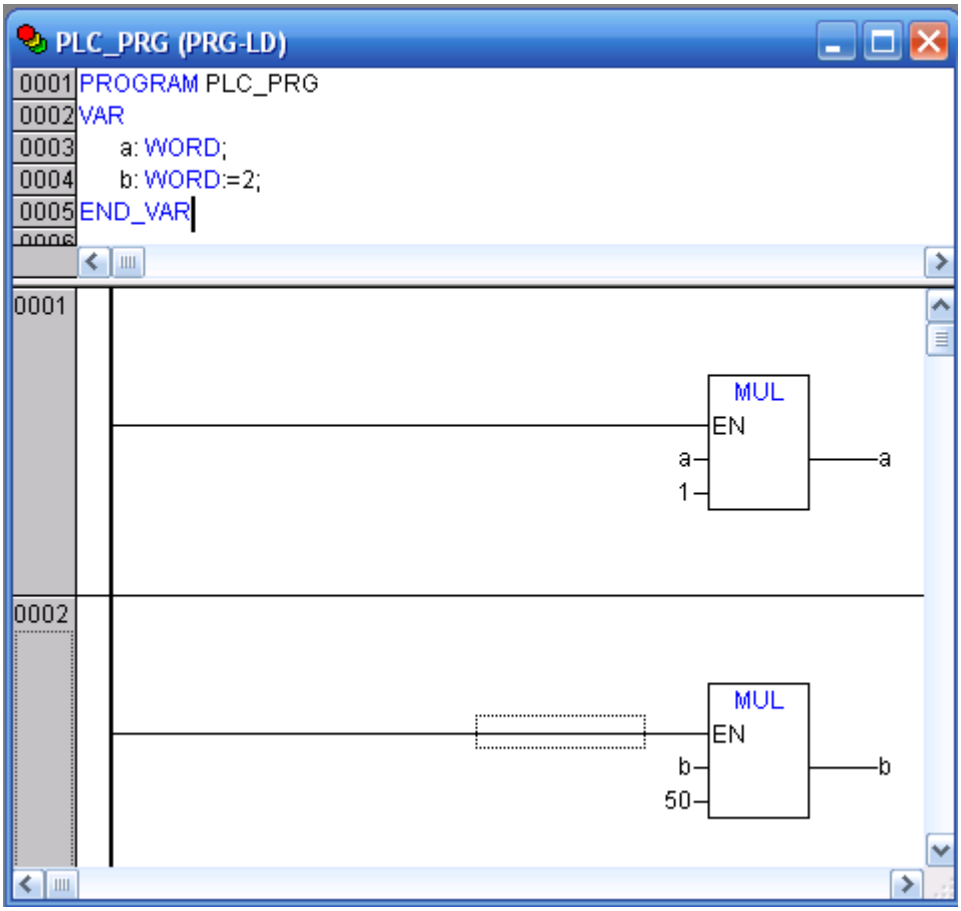


Пример функции сложения на языке ST

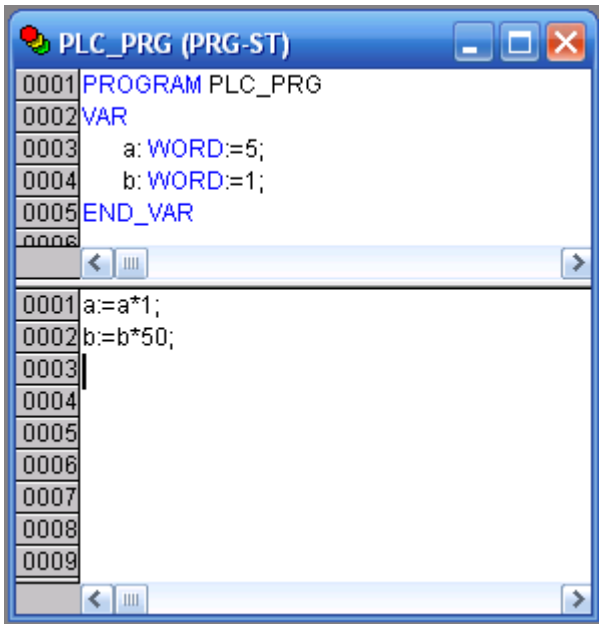


Упражнение 2. Оператор умножения:

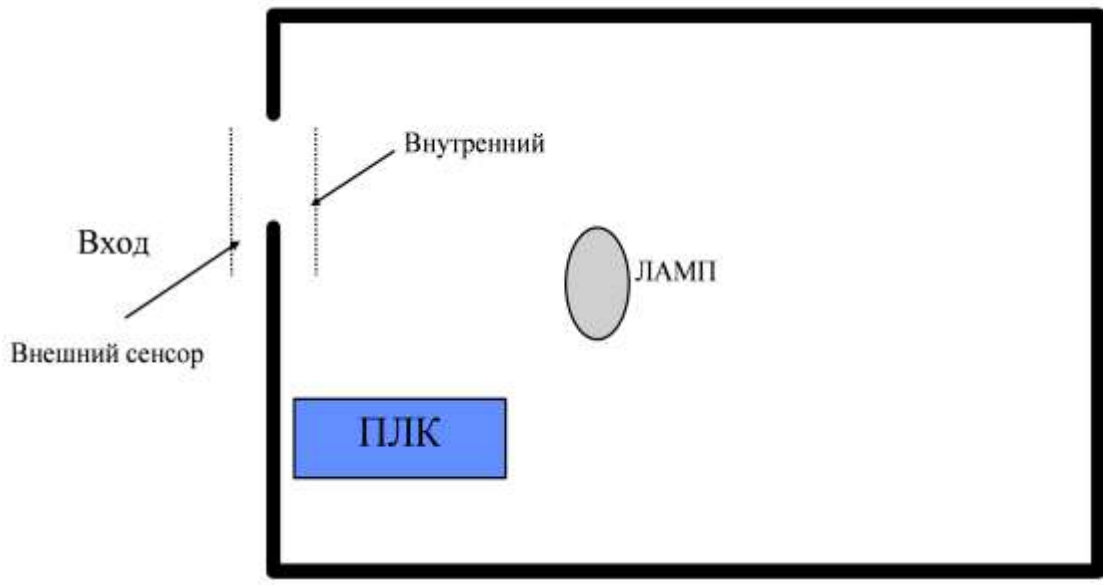
Пример функции умножения, написанной на языке LD:



Пример функции умножения на языке ST:



Упражнение 3. Управление освещением в комнате:



Цель - свет должен быть выключен, когда в комнате никого нет.

На входе установлены два дискретных датчика: один снаружи комнаты, другой внутри.

Когда срабатывает сначала внешний датчик, затем внутренний, это означает, что человек зашел в комнату. Когда срабатывает сначала внутренний датчик, затем внешний, это означает, что человек вышел из комнаты.

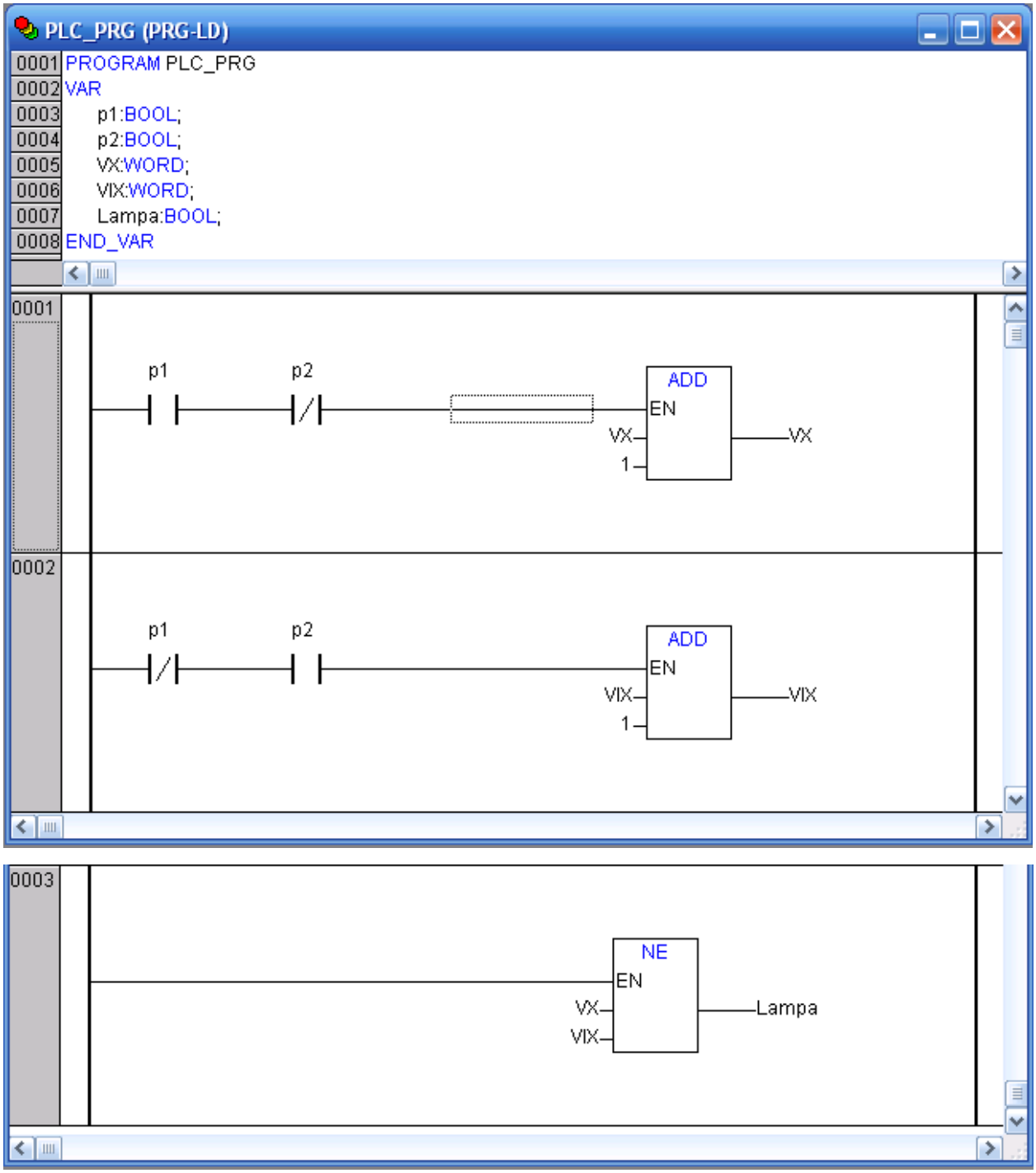
Если человек вошел – включить свет, Если человек вышел – выключить свет.

Необходимо считать количество людей, заходящих и выходящих из комнаты. Пока в комнате остается хотя бы один человек, свет должен быть включен.

Решение упражнения 3:

1. Создадим новый проект, для чего проделаем первые шаги для программирования в CoDeSys (1-4), которые описаны выше.
2. Выбрав языка программирования LD, откроется окно, в котором необходимо создать программу

Программа на языке LD:



Комментарии:

- Для решения данной задачи необходимо было считать количество входящих в комнату людей (VX) и выходящих (VIX).

- Когда срабатывает сначала внешний датчик (p1), затем внутренний (p2), это означает, что человек зашел в комнату, и счетчик считает входящих людей (VX).

- Когда срабатывает сначала внутренний датчик (p2), затем внешний (p1), это означает, что человек вышел из комнаты, и пошел счет выходящих людей (VIX).

- Затем сравнив результаты подсчетов, можно сделать вывод: если количество входящих в комнату людей равно количество выходящих, то лампочка гаснет, если не равно (NE), то горит.

3. При написании программы будет вызван ассистент ввода (рис. 7) для описания переменной, которое производиться с использованием таблицы 6.

4. Для загрузки программы в контроллер установите связь с контроллером, вызвав команду меню Online | Login. Запустите выполнение загруженной программы, вызвав команду меню Online | Run или нажатием кнопки

на передней панели контроллера.

6 Порядок выполнения работы

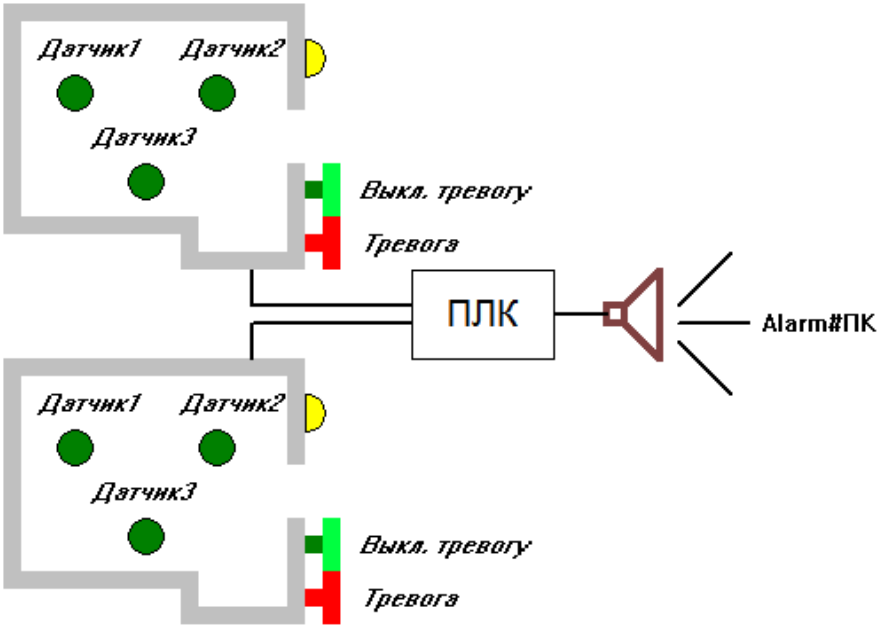
1 Получить у преподавателя исходные данные для выполнения лабораторной работы.

2 Создать программу в системе CoDeSys соответствующую заданию.

3 Ввести в память ПЛК 150 программу. Произвести отладку программы. Результат выполнения отлаженной программы записать в рабочую тетрадь и показать преподавателю.

7 Задание

Система пожарной сигнализации здания



В здании две одинаковые комнаты.

В каждой комнате установлено три пожарных датчика, кнопка ручного включения сигнализации и кнопка ручного отключения сигнализации. Для каждой комнаты предусмотрена сигнальная лампа. Сигнализация пожара является общей для обеих комнат.

Если в комнате срабатывает хотя бы один из датчиков, то загорается сигнальная лампа для соответствующей комнаты. Лампа гаснет, если все датчики в комнате отключены.

Если в комнате срабатывает любые два из трех датчиков, то включается пожарная сигнализация. Сигнализация работает до тех пор, пока ее не отключат соответствующей кнопкой.

Сигнализация может быть включена кнопкой включения вне зависимости от состояния датчиков.

Литература

1 Петров, И.В. Программируемые контроллеры. Стандартные языки и приемы прикладного проектирования/И.В. Петров, В.П. Дьяконова. - М.: СОЛОН_Пресс, 2004.

2 Петров, И.В. Отладка прикладных ПЛК программ в CoDeSys (часть 3)/ И.В. Петров, Р. Вагнер. - Промышленные АСУ и контролеры. №4. 2006.

3 Хесс, Д. Объектно-ориентированные расширения МЭК 61131_3. - СТА. № 2. 2006.

Приложение А. Схемы подключения к ОВЕН ПЛК150

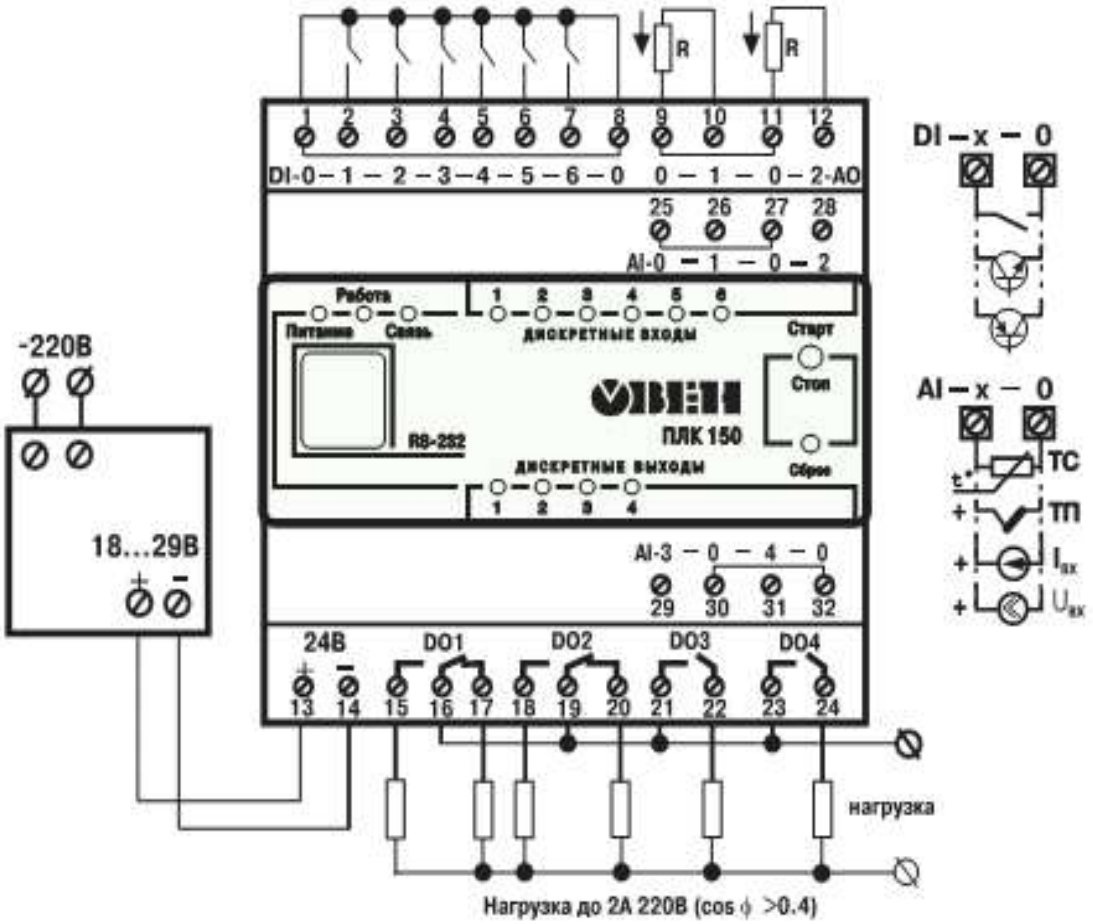


Рисунок 2 - Схема подключения питания, входов и выходов к ПЛК150-24.Х-Х

Примечания.

1. Клеммы 1, 8 и 13 электрически объединены внутри контроллера, подключение датчиков к дискретным входам может осуществляться относительно любой из этих клемм.

2. Нагрузочное сопротивление аналогового выхода (R) составляет до 900 Ом при выходном сигнале "ток 4...20 мА" и более 2 кОм при выходном сигнале "напряжение 0...10 В". Подключение внешнего блока питания для аналоговых выходов не требуется, блок питания встроен в контроллер.

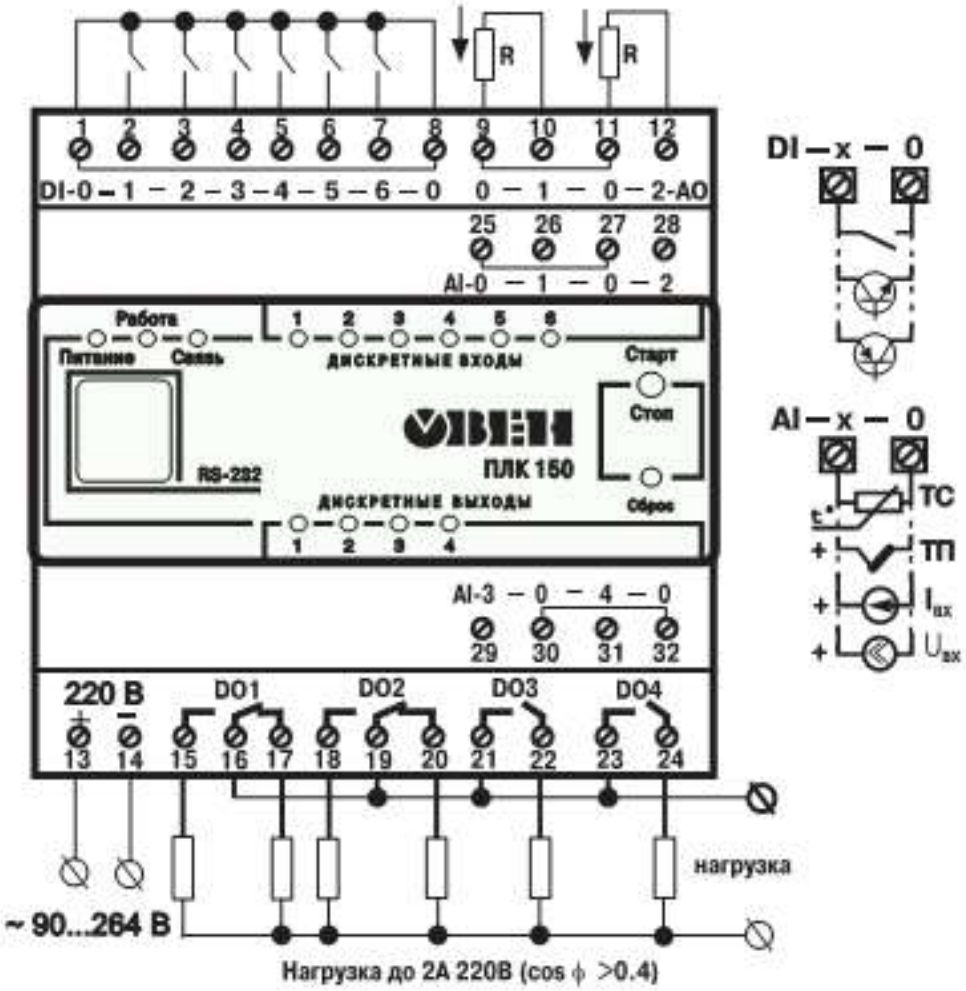


Рисунок 3 - Схема подключения питания, входов и выходов к ПЛК150-220.Х-Х

Примечания.

1. Клеммы 1 и 8 электрически объединены внутри контроллера, подключение датчиков к дискретным входам может осуществляться относительно любой из этих клемм.

2. Нагрузочное сопротивление аналогового выхода (R) составляет до 900 Ом при выходном сигнале "ток 4...20 мА" и более 2 кОм при выходном сигнале "напряжение 0...10 В". Подключение внешнего блока питания для аналоговых выходов не требуется, блок питания встроен в контроллер.

Приложение Б. Операторы в CoDeSys

Операторы присваивания

- Используются для работы со всеми типами данных

Оператор	IL	FBD	LD	ST
LD / ST	LD A ST X	A ——— X	A X	X := A;
LDN / ST	LDN A ST X	A-o ——— X	A X	X := NOT(A);
LD / S	LD A S X	A ———[S]-X	A X	IF A THEN X := TRUE; END_IF
LD / R	LD A R X	A ———[R]-X	A X R	IF A THEN X := FALSE; END_IF

Битовые операторы

- Используются для работы с двоичными типами данных

(BOOL, BYTE, WORD, DWORD)

Оператор	IL	FBD	LD	ST
NOT	LD A STN X	NOT A- -X	A X	X := NOT(A);
AND	LD A AND B ST X	AND A- B- -X	A B X	X := A AND B;
OR	LD A OR B ST X	OR A- B- -X	A X B	X := A OR B;
XOR	LD A XOR B ST X	XOR A- B- -X	A B X A B	X := A XOR B;

Сдвиговые операторы

- Используются для работы с двоичными типами данных (BOOL, BYTE, WORD, DWORD)

Оператор	IL	FBD	LD	ST
SHL	LD A SHL 1 ST X	SHL A- 1- -X	SHL EN A- 1- -X	X := SHL(A, 1);
SHR	LD A SHR 4 ST X	SHR A- 4- -X	SHR EN A- 4- -X	X := SHR(A, 4);
ROL	LD A ROL 4 ST X	ROL A- 3- -X	ROL EN A- 3- -X	X := ROL(A, 3);
ROR	LD A ROR 1 ST X	ROR A- 1- -X	ROR EN A- 1- -X	X := ROR(A, 1);

Операторы сравнения

- Используются для работы со всеми типами данных

Оператор	IL	FBD	LD	ST
EQ	LD A EQ B ST X	EQ A- B- -X	EQ EN A- B- -X	X := (A = B);
NE	LD A NE B ST X	NE A- B- -X	(аналогично)	X := (A <> B);
GE	LD A GE B ST X	GE A- B- -X	(аналогично)	X := (A >= B);
GT	LD A GT B ST X	GT A- B- -X	(аналогично)	X := (A > B);
LE	LD A LE B ST X	LE A- B- -X	(аналогично)	X := (A <= B);
LT	LD A LT B ST X	LT A- B- -X	(аналогично)	X := (A < B);

Арифметические операторы

- Выполняют алгебраические операции над целыми числами и числами с плавающей запятой

Оператор	IL	FBP	LD	ST
ADD	LD A ADD 1 ST X	A- ADD -X 1-	EN ADD A- 1- -X	X := A + 1;
SUB	LD A SUB 4 ST X	A- SUB -X 4-	EN SUB A- 3- -X	X := A - 4;
MUL	LD A MUL B ST X	A- MUL -X B-	EN MUL A- B- -X	X := A * B;
DIV	LD A DIV 8 ST X	A- DIV -X 8-	*(аналогично)*	X := A / 8;
MOD	LD 12 MOD 8 ST X	12- MOD -X 8-	*(аналогично)*	X := 12 MOD 8; *(Result = 4)* *(не исп. для REAL)*

Работа с числами с плавающей запятой

Оператор	IL	FBD	LD	ST
ABS	LD A ABS ST X	A- ABS -X	EN ABS A- -X	X := ABS(A); *(Result = 12)* *(if A = -12.0)*
TRUNC	LD A TRUNC ST X	A- TRUNC -X	EN TRUNC A- -X	X := TRUNC(A); *(Result = 4)* *(if A = 4.32)*
EXPT	LD A EXPT 3 ST X	A- EXPT -X 3-	EN EXPT A- 3- -X	X := EXPT(A,3); *(Result = 8)* *(if A = 2)*
SQRT	LD A SQRT ST X	A- SQRT -X	EN SQRT A- -X	X := SQRT(A); *(Result = 5)* *(if A = 25)*

Логарифмические операторы

- Вычисление логарифмов и экспоненты

Оператор	IL	FBD	LD	ST
EXP	LD A EXP ST X	A- EXP -X	EN EXP A- -X	X := EXP(A); *(Result = 7.389)* *(if A = 2)*
LN	LD A LN ST X	A- LN -X	EN LN A- -X	X := LN(A); *(Result = 2)* *(if A = 7.389)*
LOG	LD A LOG ST X	A- LOG -X	EN LOG A- -X	X := LOG(A); *(Result = 3)* *(if A = 1000)*

Тригонометрические операторы

Оператор	IL	FBD	LD	ST
SIN	LD A SIN ST X	SIN A- -X	SIN EN A- -X	X := SIN(A);
ASIN	LD A ASIN ST X	ASIN A- -X	ASIN EN A- -X	X := ASIN(A);
COS	LD A COS ST X	COS A- -X	COS EN A- -X	X := COS(A);
ACOS	LD A ACOS ST X	ACOS A- -X	ACOS EN A- -X	X := ACOS(A);
TAN	LD A TAN ST X	TAN A- -X	TAN EN A- -X	X := TAN(A);
ATAN	LD A ATAN ST X	ATAN A- -X	ATAN EN A- -X	X := ATAN(A);

Операторы выбора

- Предназначены для ограничения и выбора операндов;
- Используются с любыми типами данных.

Оператор	IL	FBD	LD	ST
MIN	LD A MIN B ST X	MIN A- B- -X	MIN EN A- B- -X	X := MIN(A,B);
MAX	LD A MAX 1 ST X	MAX A- 1- -X	***(как выше)***	X := MAX(A,1);
LIMIT	LD -8 LIMIT A,5 ST X	LIMIT -8- A- 5- -X	***(как выше)***	X := LIMIT(-8,A,5); ***X = -8 if A < -8*** ***X = 5 if A > 5***
SEL	LD A SEL 10,B ST X	SEL A- 10- B- -X	***(как выше)***	X := SEL(A,10,B); ***X = 10 if A is FALSE*** ***X = B if A is TRUE***
MUX	LD A MUX 0,10,B ST X	MUX A- 0- 10- B- -X	***(как выше)***	X := MUX(A,0,10,B); ***X = 0 if A is 0*** ***X = 10 if A is 1*** ***X = B if A is 2***