ПРОГРАММИРОВАНИЕ ПЛК В CoDeSys

А.Б. Лоскутов, А.А. Лоскутов, Д.В. Зырин

Программирование ПЛК в CoDeSys.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ……... 6
1. ОБЩИЕ СВЕДЕНИЯ О ПРОГРАММИРОВАНИИ ПЛК…… 7
1.1. Структура и состав ПЛК……... 7
1.2. Интеграция ПЛК в АСУТП…….. 11
1.3. Технология *OPC*…….... 13
1.4. Основные принципы стандарта МЭК 61131-3…….... 16
1.5. Комплекс *CoDeSys*…….... 17
1.6. Структура проекта в *CoDeSys*……... 18
1.6.1. Компоненты организации программ (*POU)*…….. 19
1.6.2. Типы данных……... 21
1.6.3. Переменные……. 23
1.6.4. Структура программного обеспечения ПЛК……... 23
1.7. Языки МЭК 61131-3…….. 24
1.7.1. Диаграммы *SFC*…….. 24
1.7.2. Список инструкций *IL*…….. 26
1.7.3. Структурированный текст *ST*…….... 27
1.7.4. Релейные диаграммы *LD*…….... 31
1.7.5. Функциональные диаграммы *FBD*……. 34
2. ПОДГОТОВИТЕЛЬНАЯ РАБОТА И ВЫПОЛНЕНИЕ ПРОГРАММИРОВАНИЯ ПЛК В *CODESYS*…….... 36
3. ***Практическая работа № 1.*** РЕАЛИЗАЦИЯ РАБОТЫ РЕВЕРСИВНОГО ПУСКАТЕЛЕ НА КОНТРОЛЛЕРЕ PLC 150.I-M *(CODESYS)*……. 37
3.1. Краткие теоретические сведения……. 37
3.2. Задание……... 39
3.3. Выполнение работы в программе *CodeSys*…….... 40
3.3.1. Создание программы на языке *LD*……. 43
3.3.2. Создание программы движения…… 47
3.3.3. Создание визуализации…….. 59
4. ***Практическая работа № 2.*** РЕАЛИЗАЦИЯ РАБОТЫ АВР НА КОНТРОЛЛЕРЕ *PLC 150.I-M (CODESYS)*…….... 65

4.1. Краткие теоретические сведения……. 65
4.2. Задание……... 67
4.3. Выполнение работы в программе *CodeSys*…….... 68
5. ***Практическая работа № 3.*** РЕАЛИЗАЦИЯ РАБОТЫ АЧР НА ПЛК…… 83
5.1. Краткие теоретические сведения……. 83
5.2. Задание……... 87
5.3. Выполнение работы в программе *CodeSys*…….... 87
6. ***Практическая работа № 4.*** РЕАЛИЗАЦИЯ АВТОМАТИЧЕСКОГО ВКЛЮЧЕНИЯ УСТРОЙСТВ КОМПЕНСАЦИИ РЕАКТИВНОЙ МОЩНОСТИ…… 93
6.1. Краткие теоретические сведения……. 93
6.2. Задание……... 94
6.3. Выполнение работы в программе *CodeSys*…….... 95
КОНТРОЛЬНЫЕ ВОПРОСЫ……. 99
БИБЛИОГРАФИЧЕСКИЙ СПИСОК……. 100

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

ПЛК – программируемый логический контроллер
ЦП – центральный процессор
АЛУ – арифметическо-логическое устройство
ПЗУ – постоянное запоминающее устройство
ОЗУ – оперативное запоминающее устройство
АВР – автоматический ввод резерва
АЧР – автоматическая частотная разгрузка
АПВ – автоматическое повторное включение
ЧАПВ – частотное автоматическое повторное включение
КБ – конденсаторная батарея
КРМ – компенсация реактивной мощности

Целевой файл (*Target file*) – набор файлов, поставляемых производителем ПЛК и описывающих аппаратные и программные особенности конкретного ПЛК, позволяет среде разработки корректно взаимодействовать с ПЛК

PLC – *programmable logic controller*
CodeSys – *Controller Development System*
SCADA – *Supervisory Control And Data Acquisition*
DDE – *Dynamic Data Exchange*
OPC – *OLE (Object Linking and Embedding) for Process Control*
POU – *Program Organization Unit*
IL – *Instruction List*
ST – *Structured Text*
SFC – *Sequential Function Chart*
FBD – *Function Block Diagram*
LD – *Ladder Diagram*
CFC – *Continuous Function Chart*

ВВЕДЕНИЕ

Крупнейшие лидеры ПЛК предлагают сегодня очень мощные комплексы по программированию с поддержкой МЭК-языков. Конъюктура рынка предъявляет новые требования к инженерам-электрикам. Необходимы знания микропроцессорных систем управления и умение программирования.

Процесс разработки и отладки программного обеспечения происходит при помощи специализированных комплексов программ, обеспечивающих комфортную среду для работы программиста. Одной из таких сред программирования является *CoDeSys 2.3*, позволяющая составлять программы в виде функциональных блоков, релейно-контактной схемы, структурированного текста и др.

В учебном пособии (практикуме) изложены краткие теоретические сведения, порядок выполнения практических работ, задания на выполнение работ, требования к содержанию отчета. Рассматриваются основные вопросы программирования контроллеров на языках стандарта МЭК 61131-3 в среде *CoDeSys* в системах электроснабжения при нормальных и аварийных режимах функционирования.

В ходе выполнения практических работ, описанных в учебном пособии, студенты приобретают практические навыки программирования контроллеров в среде *CoDeSys* для автоматического управления технологическими процессами в системах электроснабжения, закрепляют пройденный лекционный материал по работе и функционированию ПЛК и схем автоматизации систем электроснабжения.

1. ОБЩИЕ СВЕДЕНИЯ О ПРОГРАММИРОВАНИИ ПЛК

1.1. Структура и состав ПЛК

Приобретение практических навыков программирования контроллеров в среде *CoDeSys* для автоматического управления технологическими процессами в системах электроснабжения. Закрепление пройденного материала по работе и функционированию ПЛК и схем автоматизации систем электроснабжения.

Программируемый логический контроллер (ПЛК) - это программно управляемый дискретный автомат, имеющий некоторое множество входов, подключенных посредством датчиков к объекту управления, и множество выходов, подключенных к исполнительным устройствам. Он контролирует состояние входов и вырабатывает определенные последовательности программно заданных действий, отражающихся в изменении выходов [1].

В зависимости от решаемой задачи промышленные контроллеры могут применяться на транспорте, для управления климатом, электродвигателями, при автоматизации технологических процессов, позволяют измерять и регулировать температуру, влажность, давление, ток, напряжение, время и т.д.

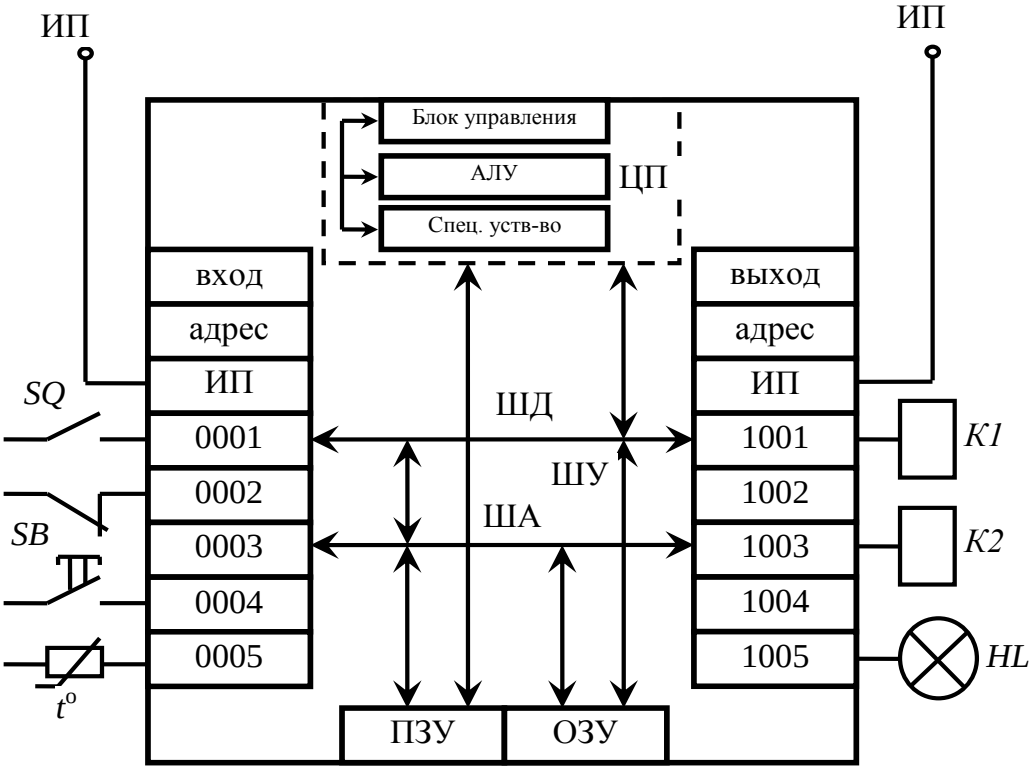


Рис. 1.1. Структурная схема ПЛК

а) *б*)

Рис. 1.2. Внешний вид контроллера:
а – ОВЕН ПЛК 150; *б* – ОВЕН ПЛК 160

Контроллер содержит центральный процессор (ЦП). Вычислительный процесс процессора организован между входными данными внешних устройств (или исполнительных механизмов) и выходными данными исполнительных сигнальных и управляющих устройств.

Внешние устройства: кнопки, датчики, устройства дискретного сигнала и др.; исполнительные и сигнальные устройства – сигнальные индикаторы, катушки управления.

Центральный процессор объединяет в себе арифметическое логическое устройство, блок управления и специальные устройства.

Арифметическое логическое устройство (АЛУ) - блок процессора, который под управлением устройства управления (УУ) служит для выполнения арифметических и логических преобразований (начиная от элементарных) над данными.

Выполняет АЛУ семь операций:

- сложение байт;
- логические операции (и, или, или логическое сравнение если);
- инкримент (увеличение на единицу);
- фекримент (уменьшение на единицу);
- инверсия (с 1 на 0 и наоборот);
- сдвиг влево и вправо;
- десятичная коррекция.

Блок управления отвечает за вызов команд из памяти и определение их типа.

Специальные устройства – устройства, поддерживающие работу процессора, – регистры, счетчики команд, кэш память и др.

Все компоненты контроллера соединены шинами. Физически шины представляют собой набор параллельно связанных проводов, по которым передаются адреса, данные и сигналы управления. Шина данных (ШД) производит считывание данных со входа. Шина данных поддерживает шина адресов (ША) и шина управления (ШУ). Шина адресов дает точный адрес откуда взять и куда положить информацию, ШУ передается управляющий сигнал.

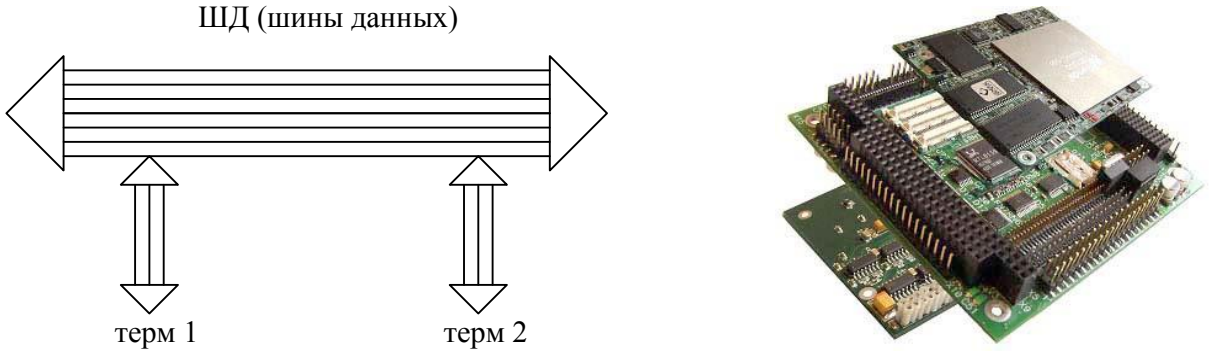


Рис. 1.3. Шины

Постоянное запоминающее устройство (ПЗУ) является программоподдерживающим устройством.

В **оперативном запоминающем устройстве (ОЗУ)** хранятся данные о состоянии на входах.

Входные модули служат для приема сигнала от входных логических элементов. Это могут быть кнопки управления, тумблеры, конечные выключатели, датчики, контакты исполнительных реле, устройства дискретного сигнала.

Выходные модули преобразуют полученную информацию от ЦП и управляют исполнительными устройствами. Это могут быть промежуточные реле, усилители, пускатели, сигнальные индикаторы, катушки управления, ЭМ и т.п.

Существует несколько типов выходных модулей: на 8 или 24 выхода. У каждого модуля своя коммутирующая способность и свое направление в зависимости от исполнения на транзисторах, симисторах, тиристорах или электромагнитном реле. Индикация состояния выхода выполняется с помощью светодиодов по каждому выходному каналу. В некоторых используются совмещенные модули входов и выходов.

Центральный процессор определяет практически все основные параметры контроллера. Процессор выполняет операции считывания и обработки команд, следит за порядком выполнения программы, управляет процессами считывания и записи памяти, распределяет информацию в выходные модули. Элементом процессора являются однобитовые (работают с одним видом информации) и многобитовые буферы памяти.

Рабочий цикл любого процессора состоит из трех этапов:

1) загрузка в память состояния спрашиваемых входных модулей;

2) последовательная обработка состояний в соответствии с программой и запоминание промежуточных результатов;

3) передача результатов вычислений в выходные модули.

Работа программируемого контроля происходит циклически с повторением рабочих циклов без дополнительных условий запуска.

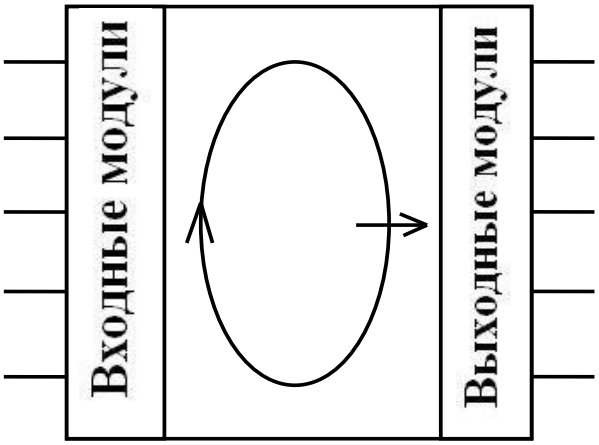


Рис. 1.4. Рабочий цикл ЦП

Элементы памяти определяют возможности и характеристики контроллера.

Два вида памяти:

1) служебная - для хранения программы и управления работой контроллера (недоступна для пользователя);

2) рабочая - для хранения программ и информации пользователя.

По месту нахождения памяти:

1) внутренняя ПЗУ- *ROM* - для обеспечения автономной работы ПЛК;

2) внешняя ОЗУ- *RAM* - для обеспечения отладки программ, а также их хранения.

Существуют контроллеры с тремя видами ПЗУ:

1) программируемые при изготовлении (обозначают ПЗУ или *RO*M);

2) с однократным программированием, позволяющим пользователю однократно изменить состояние матрицы памяти электрическим путем по заданной программе (обозначают ППЗУ или *PROM*);

3) перепрограммируемые (репрограммируемые) с возможностью многократного электрического перепрограммирования, с ультрафиолетовым (обозначают РПЗУУФ или *EPROM*) или электромагнитным (обозначают РПЗУЭС или *EEPROM*, или *E2PROM*) стиранием информации. Для обеспечения возможности объединения по выходу при наращивании памяти все ПЗУ имеют выходы с тремя состояниями или открытые коллекторные выходы.

1.2. Интеграция ПЛК в АСУТП

Контроллеры традиционно работают в нижнем звене автоматизированных систем управления предприятием (АСУ) – систем, непосредственно связанных с технологией производства. Программируемые логические контроллеры обычно являются первым шагом при построении систем АСУ. Это объясняется тем, что необходимость автоматизации отдельного механизма или установки всегда наиболее очевидна. Она дает быстрый экономический эффект, улучшает качество производства, позволяет избежать физически тяжелой и рутинной работы. Контроллеры по определению созданы именно для такой работы.

Далеко не всегда удается создать полностью автоматическую систему. Часто «общее руководство» со стороны квалифицированного человека-диспетчера – необходимо. В отличие от автоматических систем управления такие системы называют автоматизированными. Еще 10 - 15 лет назад диспетчерский пульт управления представлял собой табло с множеством кнопок и световых индикаторов. В настоящее время подобные пульты применяются только в очень простых случаях, когда можно обойтись несколькими кнопками и индикаторами. В более «серьезных» системах применяются персональные компьютеры.

Появился целый класс программного обеспечения, реализующего интерфейс человек–машина (*HMI*). Это системы сбора данных и оперативного диспетчерского управления (*Supervisory Control And Data Acquision System – SCADA*). Современные *SCADA*-системы выполняются с обязательным применением средств мультимедиа. Помимо живого отображения процесса производства, хорошие диспетчерские системы позволяют накапливать полученные данные, проводят их хранение и анализ, определяют критические ситуации и производят оповещение персонала по каналам телефонной и радиосети, позволяют создавать сценарии управления (как правило, *Visual Basic*), формируют данные для анализа экономических характеристик производства.

Разделение производства ПЛК, средств программирования и диспетчерских систем привело к появлению стандартных протоколов обмена данными. Наибольшую известность получила технология *OPC (OLE for Process Control)*, базирующаяся на механизме *DCOM Microsoft Windows*. Механизм динамического обмена данными (*DDE*) применяется пока еще достаточно широко, несмотря на то что требованиям систем реального времени не удовлетворяет.

Все это «многоэтажное» объяснение призвано подчеркнуть еще одно немаловажное преимущество ПЛК – средства системной интеграции

являются составной частью базового программного обеспечения современного ПЛК (рис. 1.5). Например, был разработан и отлажен автономный проект на контроллере при помощи системы подготовки программ *CoDeSys*. Далее дорабатывать программу, чтобы связать ПЛК с системой диспетчерского управления, базой данных или Интернет-сервером не требуется. В комплекс программирования ПЛК входит *OPC*-сервер. Он получает доступ к данным ПЛК также прозрачно, как и отладчик. Достаточно обеспечить канал передачи данных ПЛК – *OPC*-сервер. Обычно такой канал уже существует. Он был использован при отладке. Вся дальнейшая работа сводится к определению списка доступных переменных, правильной настройке сети, конфигурированию *OPC*-сервера и *SCADA*-системы. Операция очень напоминает настройку общедоступных устройств локальной сети персонального компьютера.

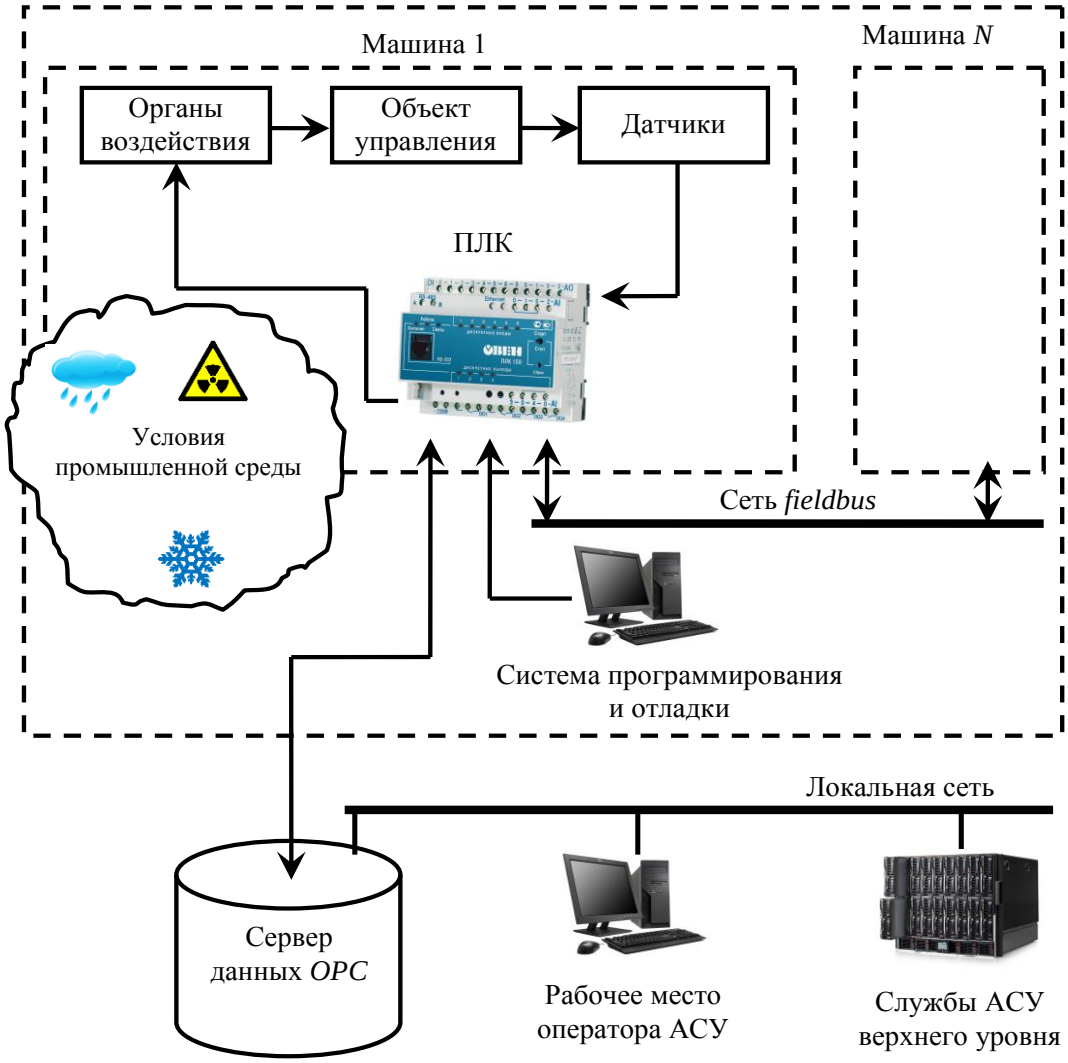


Рис. 1.5. Связь среды программирования с ПЛК

Часто возникающей задачей является интеграция нескольких ПЛК с целью синхронизации их работы. Здесь появляются сети, обладающие

рядом специфических требований. Это требования, аналогичные требованиям к ПЛК: режим реального времени; надежность в условиях промышленной среды; ремонтопригодность; простота программирования. Такой класс сетей получил название промышленных сетей (*fieldbus*). Существует масса фирменных реализаций и достаточно много стандартов таких сетей (Bitbus, Modbus, Profibus, CANopen, DeviceNet), позволяющих интегрировать аппаратуру различных фирм, но ни один из них нельзя признать доминирующим.

Благодаря продуктивному развитию средств сетевой интеграции появилась возможность создания распределенных систем управления. В 80-е годы XX в. доминировали ПЛК с числом входов-выходов в несколько сотен. В настоящее время большим спросом пользуются микроПЛК с количеством входов-выходов до 64. В распределенных системах каждый ПЛК решает локальную задачу. Задача синхронизации управления выполняется компьютерами среднего звена АСУ. Распределенные системы выигрывают по надежности, гибкости монтажа и простоте обслуживания [1].

1.3. Технология *OPC*

OPC (OLE for Process Control) – промышленный стандарт, созданный консорциумом всемирно известных производителей оборудования и программного обеспечения при участии *Microsoft*. Этот стандарт описывает интерфейс обмена данными между устройствами управления технологическими процессами. Главной целью было предоставление разработчикам систем диспетчеризации некоторой независимости от конкретного типа контроллеров. OPC основывается на технологии OLE/COM/DCOM компании *Microsoft*.

Стандарт OPC разрабатывался с целью сокращения затрат на создание и сопровождение приложений промышленной автоматизации. В начале 1990 г. у разработчиков промышленного ПО возникла потребность в универсальном инструменте обмена данными с устройствами разных производителей или по разным протоколам обмена данными.

Суть OPC – предоставить разработчикам промышленных программ универсальный фиксированный интерфейс (то есть набор функций) обмена данными с любыми устройствами. В то же время разработчики устройств предоставляют программу, реализующую этот интерфейс.

В состав *SCADA* – систем входит определенное множество программ-клиентов, которым необходимо получать данные от различных источников – контроллеров, плат расширения, интеллектуальных датчиков – посредством драйверов, написанных разработчиками

оборудования (рис. 1.6). Но при этом возникают следующие проблемы, которые послужили причинами создания *OPC*:

- каждая программа диспетчеризации должна иметь драйвер для конкретного устройства ввода-вывода;
- возникают конфликты между драйверами различных разработчиков, что приводит к тому, что какие-то режимы или параметры работы оборудования не поддерживаются всеми разработчиками ПО;
- модификации оборудования могут привести к потере функциональности драйвера;
- конфликты при обращении к устройству – различные программы диспетчеризации не могут получить доступ к одному устройству одновременно из-за использования различных драйверов.

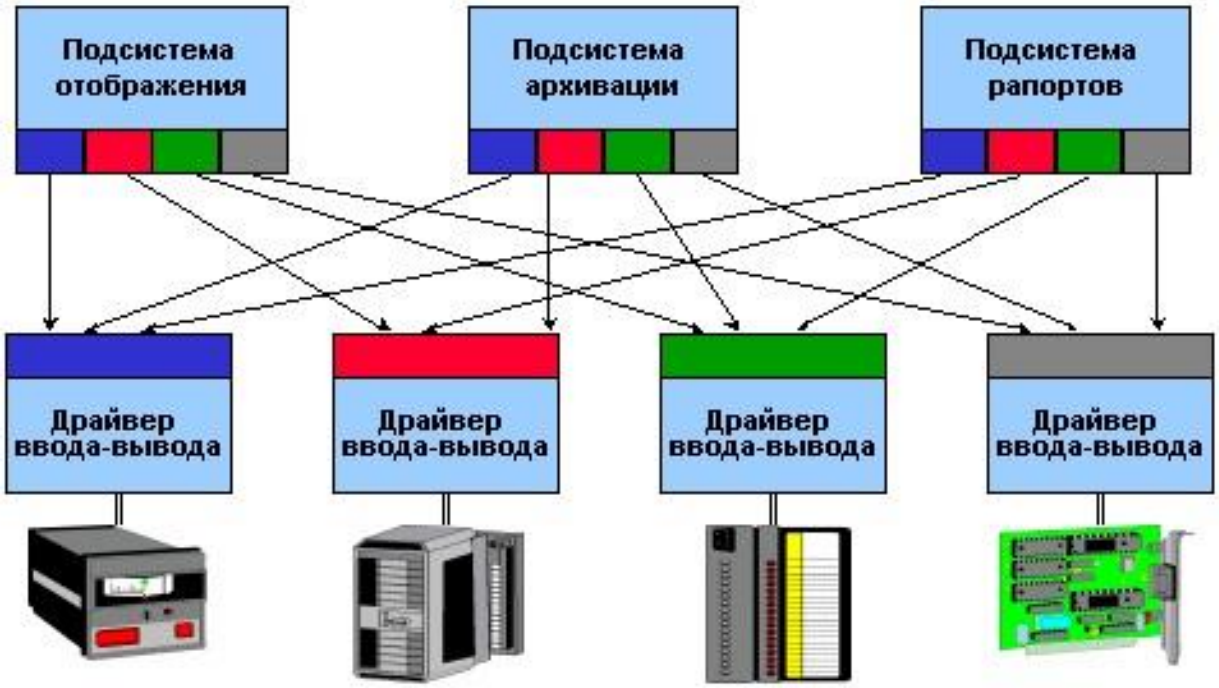


Рис. 1.6. Пример схемы работы множества различных драйверов

Производители оборудования стараются решить эту проблему с помощью разработки дополнительных драйверов. Однако эти попытки встречают сильное сопротивление разработчиков систем диспетчеризации, которые должны в этом случае усложнять свои клиентские протоколы.

OPC проводит четкую разграничительную линию между производителями оборудования и разработчиками драйверов. Данная технология предоставляет механизм сбора данных из различных источников и передачу этих данных любой клиентской программе, независимо от типа используемого оборудования.

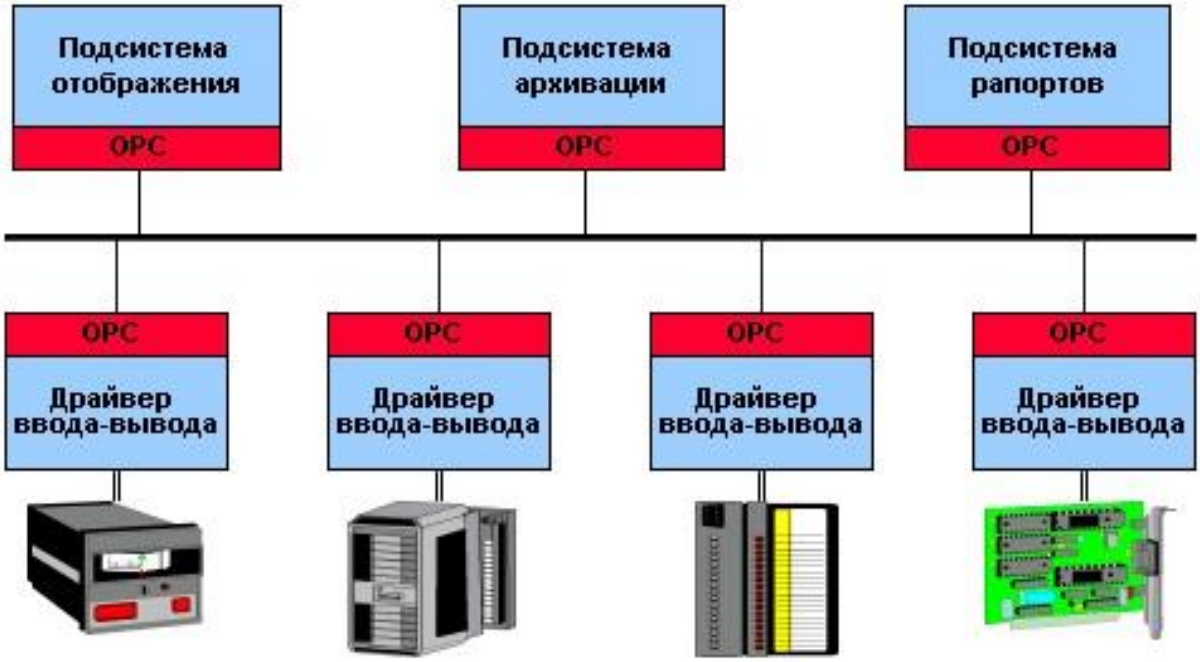


Рис. 1.7. Пример схемы решения на базе *OPC*

Стандарт *OPC* был разработан для обеспечения доступа клиентской программы к нижнему уровню технологического процесса в наиболее удобной форме. Широкое распространение технологии *OPC* в промышленности имеет следующие преимущества:

- независимость в применении систем диспетчеризации от используемого в конкретном проекте оборудования;
- разработчики ПО не должны постоянно модифицировать свои продукты из-за модификации оборудования или выпуска новых изделий;
- заказчик получает свободу выбора между поставщиками оборудования, а также имеет возможность интегрировать это оборудование в информационную систему предприятия, которая может охватывать всю систему производства и управления.

Стандарт обмена данными *OPC* базируется на распространенной общепринятой схеме Клиент-Сервер. Эта архитектура позволяет подключить множество клиентов к одному серверу. И наоборот, данный стандарт позволяет использовать одному клиенту различные *OPC*-серверы.

SCADA-программы диспетчеризации являются *OPC*-клиентами, получающими данные технологического процесса из *OPC*-серверов. *OPC*-серверы опираются на коммуникационный протокол представленного оборудования. Соединение с *OPC*-серверами происходит либо локально в пределах одного компьютера, либо через сеть, что расширяет возможности в построении топологии сбора данных при помощи *OPC*-серверов (рис. 1.7).

Некоторые *SCADA* являются вертикально-интегрированными, в их состав входят системы программирования для свободно-программируемых контроллеров. В них также используются внутренние драйверы для связи с контроллером. Такие *SCADA* позволяют создать программно-технические комплексы с использованием оборудования разных производителей.

1.4. Основные принципы стандарта МЭК 61131-3

МЭК 61131-3 – международный стандарт, описывающий языки программирования для программируемых логических контроллеров.

Стандарт позволяет разработчику не зависеть от производителя системы программирования и определяет принципы программирования ПЛК:

- пять различных языков программирования;
- типы программных компонентов (*POU*): функции, программы и функциональные блоки;
- правила объявления и типы переменных.

Стандартные языки программирования:

- ***IL*** *(Instruction List)* – текстовый язык – аппаратно-независимый низкоуровневый ассемблероподобный язык;
- ***FBD*** *(Function Block Diagram)* – графический язык. Функциональный блок (ФБ) выражает некую подпрограмму. Каждый ФБ имеет входы (слева) и выходы (справа). Программа создаётся ссоединением множества ФБ;
- ***LD*** *(Ladder Diagram)* – графический язык – программная реализация электрических схем на базе электромагнитных реле;
- ***ST*** *(Structured Text)* – текстовый – паскалеподобный язык программирования;
- ***SFC*** *(Sequential Function Chart)* – графический высокоуровневый язык, созданный на базе математического аппарата сетей Петри. Описывает последовательность состояний и условий переходов.

Инженер, спроектировавший машину, должен иметь возможность самостоятельно написать программу управления. Никто лучше его не знает, как должна работать данная машина. Инженер, привыкший работать с электронными схемами, гораздо легче сможет выражать свои мысли в *LD* или *FBD*. Если он знаком с языками *PASCAL* или *C*, то использование языка *ST* не составит для него сложности.

1.5. Комплекс *CoDeSys*

CoDeSys - это современный инструмент для программирования контроллеров (*CoDeSys* образуется от слов *Controllers Development System*) на языках стандарта МЭК 61131-3 [1,2].

Используемые редакторы и отладочные средства базируются на широко известных и хорошо себя зарекомендовавших принципах, знакомых по другим популярным средам профессионального программирования (например, *Visual C++*).

Для привязки к конкретному ПЛК требуется адаптация, касающаяся низкоуровневых ресурсов распределение памяти, интерфейс связи и драйверы ввода-вывода.

Среди особенностей комплекса можно отметить следующее:

- прямая генерация машинного кода. Генератор кода *CoDeSys* - это классический компилятор, что обеспечивает очень высокое быстродействие программ пользователя;
- полноценная реализация МЭК-языков, в некоторых случаях даже расширенная;
- «разумные» редакторы языков построены таким образом, что не позволяют делать типичные для начинающих МЭК программистов ошибки;
- встроенный эмулятор контроллера позволяет проводить отладку проекта без аппаратных средств. Эмулируется не некий абстрактный контроллер, а конкретный ПЛК с учетом аппаратной платформы. При подключении реального контроллера (режим *online*) отладчик работает аналогичным образом;
- встроенные элементы визуализации дают возможность создать модель объекта управления и проводить отладку проекта без изготовления средств имитации. Существует «операционная» версия *CoDeSys*. Это компактное приложение, выполняющее только визуализацию, без средств разработки. Во многих простых случаях нет необходимости приобретать отдельно *SCADA*-систему. Серверы данных (*DDE* и *OPC*) также входят в стандартный пакет поставки;
- очень широкий набор сервисных функций, ускоряющих работу программиста [1];

Базовый состав комплекса программирования ПЛК состоит из двух обязательных частей: системы исполнения и рабочего места программиста. Система исполнения функционирует на контроллере и, кроме непосредственно исполнения управляющей программой, обеспечивает загрузку кода прикладной программы и отладочные функции. Связь между системой исполнения и рабочим местом организована через стандартные интерфейсы (*RS 232, RS 422, RS485*).

Посредником между средой разработки и ПЛК служит шлюз связи (*gateway*). Шлюз связи взаимодействует с интегрированной средой через *Windows* сокет-соединение, построенное на основе протокола *TCP/IP*. Такое соединение обеспечивает единообразное взаимодействие приложений, работающих на одном компьютере или в сети (рис. 1.8).

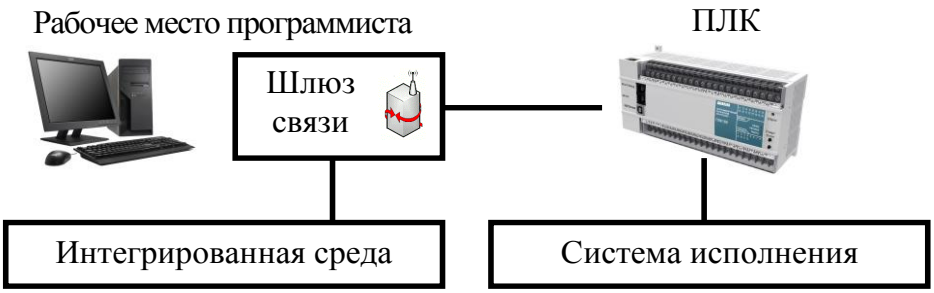


Рис. 1.8. Связь среды программирования с ПЛК

CoDeSys поддерживает следующие текстовые языки программирования:

- *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).*

1.6. Структура проекта в *CoDeSys*

Проект содержит ряд разнородных объектов *POU*, данных разных типов, элементов визуализации и ресурсов (рис. 1.9) [3].

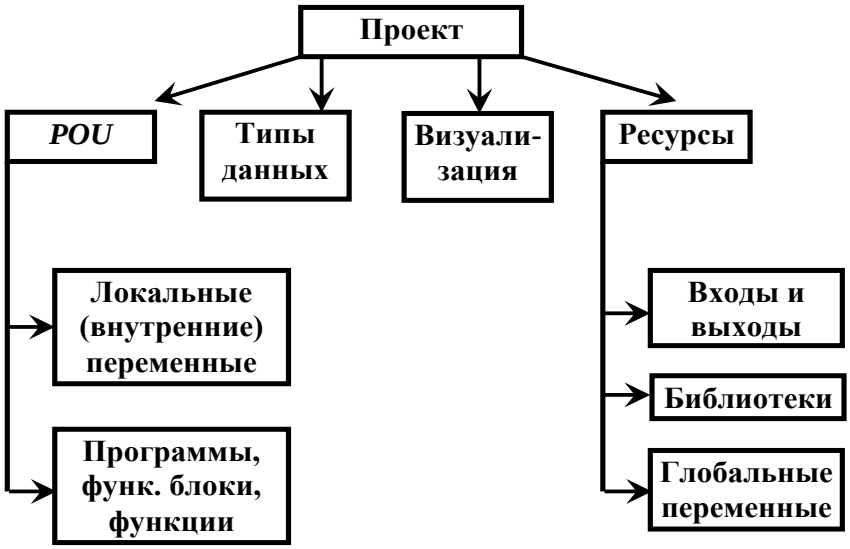


Рис. 1.9. Структура проекта в *CoDeSys*

1.6.1. *Компоненты организации программ (POU)*

POU (Program Organization Unit) – компоненты организации программ, образующие код прикладного программного обеспечения ПЛК. Именно на уровне компонентов доступно совмещение различных языков МЭК. Каждый компонент программы имеет собственное наименование, определенный интерфейс и описание на одном из МЭК-языков.

Компоненты организации программ являются базовыми элементами, из которых строится код проекта. Аналогичным образом электронные устройства составляются обычно из модулей. Каждый компонент программы имеет собственное наименование, определенный интерфейс и описание на одном из МЭК-языков. Один компонент может вызвать другие. Вызов самого себя (рекурсия) в стандарте МЭК не разрешена. Комбинировать различные языки в одном проекте можно при описании различных компонентов, но отдельный компонент целиком реализуется на одном языке МЭК. При вызове компонента язык его реализации значения не имеет.

К компонентам организации программ в МЭК-стандарте относятся: **функции, функциональные блоки и программы**. Все они во многом похожи, но имеют определенные особенности и различное назначение.

Одной из задач, решаемых компонентами, является локализация имен переменных. Это означает, что в различных компонентах можно использовать повторяющиеся имена. Так, любимую переменную с оригинальным идентификатором *X* можно использовать в каждом компоненте, и всякий раз это будет новая переменная. Область видимости локальных переменных определяется рамками одного компонента.

Реализации любого *POU* всегда должен предшествовать **раздел объявлений.** Объявления функции, функционального блока и программы начинаются соответственно с ключевых слов **FUNCTION, FUNCTION_BLOCK** и **PROGRAM**. За ним следует идентификатор (имя компонента). Далее определяется интерфейс *POU*.

К **интерфейсу компонента** относятся входы **VAR__INPUT,** выходы **VAR__OUTPUT** и переменные типа вход-выход **VAR__IN_OUT**. Завершают раздел объявлений локальные переменные **VAR**.

В функциях разделы **VAR_OUTPUT** и **VAR__IN__OUT** отсутствуют. Выходом функции служит единственная переменная, совпадающая с именем функции. Тип возвращаемого значения указывается при определении идентификатора через двоеточие.

Например: **FUNCTION** iNearby **INT**

Структура раздела объявлений POU показана в табл. 1.1.

Таблица 1.1

Структура раздела объявлений POU

Тип POU	Функция	Функциональный блок	Программа
	FUNCTION имя: **ТИП**	**FUNCTION__BLOCK** имя	**PROGRAM** имя
Интерфейс	**VAR_INPUT**	**VAR_INPUT**	**VAR_INPUT**
	—	**VAR_OUTPUT**	**VAR_OUTPUT**
	—	**VAR_IN_OUT**	**VAR_IN_OUT**
Локальные переменные	**VAR**	**VAR**	**VAR**

Функция — это программный компонент, отображающий множество значений входных параметров на выход. Функция всегда возвращает только одно значение. При объявлении функции указывается тип возвращаемого значения, имя функции и список входных параметров. Вызов функции производится по имени с указанием значений входных параметров. Функция может использоваться в математических выражениях наряду с операторами и переменными.

Тип функции (тип возвращаемого значения) может быть любым из числа стандартных типов данных или типов созданных пользователем. Тело функции может быть описано на языках *IL, ST, LD* или *FBD*. Использовать *SFC* нельзя. Из функции можно вызывать библиотечные функции и другие функции текущего проекта.

Функциональный блок — программный компонент, отображающий множество значений входных параметров на множество выходных. После выполнения экземпляра 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. Сохраняются все переменные, включая входные и выходные. Так, если мы вызовем экземпляр функционального блока, не определяя значения некоторых входных параметров, он будет использовать ранее установленные значения. Возможность задания переменного числа входных значений заложена по определению и не требует каких-либо дополнительных усилий. Извне доступны только входы и выходы функционального блока, получить доступ к внутренним переменным блока нельзя.

С позиций объектно-ориентированного программирования (ООП) функциональные блоки — это объекты, великолепно реализующие инкапсуляцию, т. е. сокрытие деталей реализации. Объединение кода и данных роднит функциональные блоки с классами ООП.

Программа — глобальный программный элемент, отображающий множество значений входных параметров на множество выходных.

Программа очень похожа на функциональный блок. Из всех программных компонентов МЭК-программа самый крупный. При помощи программ определяется верхний уровень проекта и реализуется управление многозадачностью. Программы являются глобальными компонентами и объявляются на уровне ресурсов.

При создании программы первый *POU* помещается в новый проект автоматически и получает название *PLC_PRG*. Именно с него и начинается выполнение процесса (по аналогии с функцией *main* в языке *C*), из него будут вызываться другие программные блоки (программы, функциональные блоки и функции).

1.6.2. *Типы данных*

Программируемый логический контроллер способен работать с различными типами данных, которые определяют род информации, диапазон представления и множество допустимых операций. Типы данных МЭК разделяются на элементарные и пользовательские.

Элементарные типы данных:

1. **Целочисленные переменные** отличаются различным диапазоном сохраняемых данных и, естественно, различными требованиями к памяти. Подробно данные характеристики представлены в табл. 1.2.

Таблица 1.2

Целочисленные типы данных

Тип	Нижний предел	Верхний предел	Размер, байты
BYTE	8 бит		1
WORD	16 бит		2
DWORD	32 бита		4
LWORD	64 бита		8
SINT	-128	127	1
INT	-32768	32767	2
DINT	-2^{31}	$2^{31}-1$	4
LINT	-2^{63}	$2^{63}-1$	8
USINT	0	255	1
UINT	0	65535	2
UDINT	0	$2^{32}-1$	4
ULINT	0	$2^{64}-1$	8

2. **Логические переменные** объявляются ключевым словом **BOOL**. Это означает их принадлежность к алгебре Буля. Они могут принимать только значение логического нуля («0») *FALSE* (ЛОЖЬ) или логической

единицы («1») *TRUE* (ИСТИНА). При начальной инициализации логическое значение по умолчанию — ЛОЖЬ.

3. **Переменные действительного типа (REAL)** представляют действительные числа в диапазоне $\pm 10^{\pm 38}$. Из 32 бит, занимаемых числом, мантисса занимает 23 бита. Точность представления приблизительно составляет 6 - 7 десятичных цифр.

4. **Время суток и дата** типы переменных, выражающие время дня или дату, представляются в соответствии с *ISO 8601*.

Таблица 1.3

Переменные времени суток и даты

Тип	Короткое обозначение	Начальное значение
DATE	D	1 января 1970 г.
TIME__OF_DAY	TOD	00:00
DATE_AND__TIME	DT	00:00 1 января 1970 г.

5. **Интервал времени -** переменные типа **TIME**. В отличие от времени суток (TIME_OF_DAY) временной интервал не ограничен максимальным значением в 24 часа. Числа, выражающие временной интервал, должны начинаться с ключевого слова **TIME# (**в сокращенной форме **T#).**

6. Тип **строковых переменных (STRING)** определяет переменные, содержащие текстовую информацию. Размер строки задается при объявлении.

Пользовательские типы данных:

1. **Массивы** представляют собой множество однотипных элементов с произвольным доступом. Массивы могут быть многомерными. Размерность массива и диапазоны индексов задаются при объявлении.

<Имя массива>***:ARRAY***

[<li1>..<hi1>,<li2>..<hi2>,<li3>..<hi3> **OF <***тип элемента*>;

где li1, li2, li3 указывают нижние пределы индексов; hi1, hi2 и hi3 - верхние пределы. Индексы должны быть целого типа и только положительные. Отрицательные индексы использовать нельзя.

2. **Структуры** предназначены для создания новых типов данных на основе элементов разных базовых типов. С переменной типа структура можно обращаться как с единым элементом, передавать в качестве параметра, создавать указатели, копировать и т. д.

Объявление структуры должно начинаться с ключевого слова **STRUCT** и заканчиваться **END_STRUCT**.

3. **Перечисление** позволяет определить несколько последовательных значений переменной и присвоить им наименования.

Также существуют типы переменных с ограниченным диапазоном значений, псевдонимы типов и др.

1.6.3. *Переменные*

Среди элементов МЭК-языков есть **переменные**.

Каждая переменная обязательно имеет наименование и тип. Сущность переменной может быть различной: представлять вход или выход ПЛК, данные в оперативной или энергонезависимой памяти.

Переменные принято разделять на глобальные и локальные по области видимости.

Глобальные переменные определяются на уровне ресурсов проекта (**VAR_GLOBAL**) и доступны для всех программных компонентов проекта.

Локальные переменные описываются при объявлении компонента и доступны только внутри него.

Описание любого программного компонента содержит, как минимум, один раздел объявления локальных переменных **VAR**, переменных интерфейса **VAR_INPUT, VAR_OUTPUT, VAR_IN_OUT** и внешних глобальных переменных **VAR_EXTERNAL.**

1.6.4. *Структура программного обеспечения ПЛК*

Среди средств, реализующих выполнение программ, можно выделить задачи, ресурсы, конфигурацию.

Назначение **задач** состоит в управлении работой программ проекта, исполняемых одним процессором. Как и программа, каждая задача должна иметь собственный уникальный идентификатор. Задачи подразделяются на циклические и разовые (*single*). Выполнение **разовой задачи** запускается по фронту логической триггерной переменной. **Циклические задачи** выполняются через заданные интервалы времени. Каждая задача может включать вызов одной или нескольких программ. Если программа имеет входные параметры (**VAR__INPUT**), то они задаются в описании задачи. Все программы одной задачи выполняются в одном рабочем цикле ПЛК.

Определение задач в системах программирования МЭК выполняется по-разному. Это может быть текстовое описание или графическое представление. *CoDeSys* содержит специальный инструмент — менеджер задач (*Task configuration*), представляющий задачи и их программы в виде иерархического дерева.

С точки зрения стандарта МЭК **ресурс** — это один процессор, снабженный собственной системой исполнения. То есть одна или несколько задач загружается в ресурс и исполняется им. В *CoDeSys* применяется понятие проект - все прикладное программное обеспечение, предоставляющее работу конкретного приложения. Слово же «ресурсы» употребляется во множественном числе и определяет набор аппаратно зависимых деталей проекта. То есть проект включает аппаратно независимые реализации программ (функции, функциональные блоки и их локальные данные) и требующие настройки ресурсы.

Ресурсы содержат:

- определение глобальных и прямо адресуемых переменных;
- конфигурацию ПЛК;
- установки целевой системы исполнения (тип микропроцессора, распределение памяти, порядок байт в слове, параметры сети и т. д.);
- менеджер задач.

Сюда же включаются и дополнительные фирменные инструменты, зависящие от особенностей реализации конкретной системы исполнения.

Конфигурация - это множество ресурсов, взаимодействующих определенным образом (сконфигурированных). В одной системе может быть несколько интеллектуальных ресурсов, каждый из которых обладает собственным процессором, памятью и системой исполнения. Каждый из них можно программировать. Это могут быть реальные модули (возможно, удаленные) или виртуальные машины, эмулируемые одним процессором. Все они имеют доступ к определенным наборам входов-выходов и координируют свою работу посредством глобальных переменных, расположенных в общедоступной памяти.

1.7. Языки МЭК 61131-3

1.7.1. *Диаграммы SFC*

В стандарте МЭК 61131-3 диаграммы *SFC (Sequential Function Chart)* стоят выше по отношению к остальным четырем языкам. Прототипом для разработки *SFC* послужила сеть Петри [10].

Сеть Карла Адама Петри была предложена в 1962 г. как оригинальный метод формального описания дискретных систем. Он опирается на разделение системы или отдельных ее частей на множество простых позиций. **Позиция** описывает состояние части системы, причем состояние понимается здесь достаточно гибко — это может быть состояние оборудования, процесса или программы. **Переходы** между позициями происходят при выполнении определенных условий. Графически позиция отображается в виде окружности (рис. 1.10).

Переходам соответствуют отрезки, соединенные с позициями направленными **дугами**. Каждая позиция способна обладать **маркером** и передавать его другим позициям по исходящим дугам. Маркеры отображается в виде жирной точки. Допускается одновременное присутствие нескольких маркеров. К переходу приходит одна или несколько дуг, идущих от разных позиций. От перехода также могут отходить несколько исходящих дуг, ведущих к разным позициям. Проверка условия перехода (разрешение) производится, только если хотя бы одна из его исходных позиций владеет маркером.

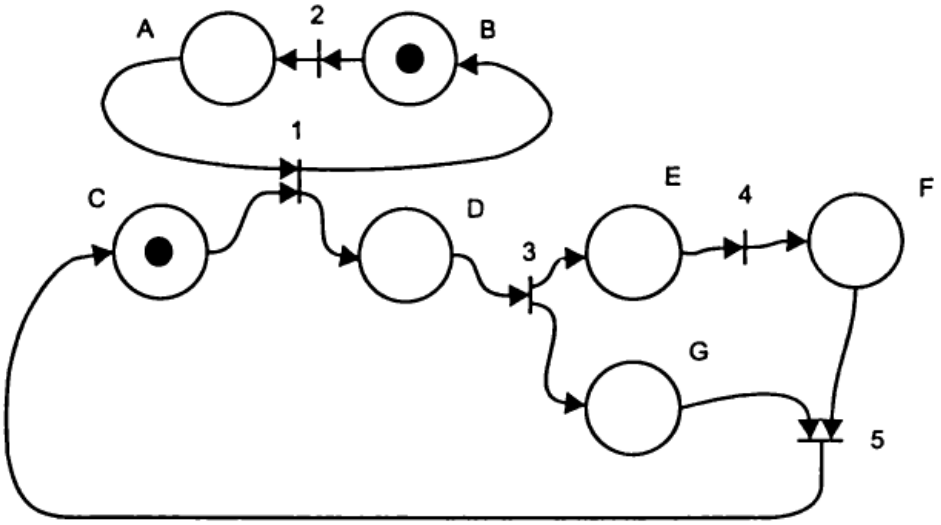


Рис. 1.10. Сеть Петри

В отличие от сетей Петри дуги в *SFC* имеют выраженную направленность сверху вниз и отражаются прямыми линиями. Позиции в *SFC* называют **шагами,** или **этапами.** На диаграмме они отражаются в виде прямоугольников. Благодаря такому «кубизму» существует возможность реализации диаграмм в символах псевдографики (рис. 1.11). Задать несколько стартовых шагов в *SFC* нельзя, только один шаг диаграммы является начальным.

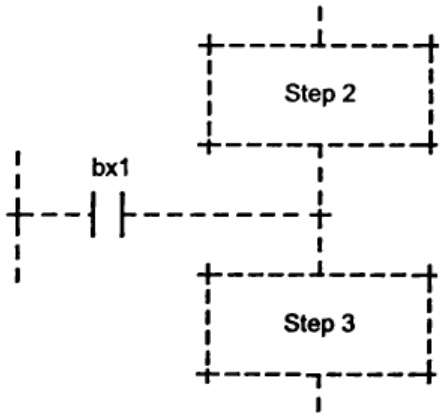


Рис. 1.11. Диаграмма *SFC* (условие перехода – язык *IL*)

Графическая диаграмма SFC состоит из шагов и переходов между ними. Разрешение перехода определяется **условием**. С шагом связаны определенные действия. Описания действий выполняются на любом языке МЭК. Сам *SFC* не содержит каких-либо управляющих команд ПЛК. Действия могут быть описаны и в виде вложенной *SFC*-схемы. Можно создать несколько уровней подобных вложений, но действия нижнего уровня все равно необходимо будет описать на *IL, ST, LD* или *FBD.*

Целью применения *SFC* является разделение задачи на простые этапы с формально определенной логикой работы системы. *SFC* дает возможность быстрого построения прототипа системы без программирования. Причем для отработки верхнего уровня не требуется детальное описание действий, так же, как и привязка к конкретным аппаратным средствам.

1.7.2. *Список инструкций IL*

*Я*зык *IL (Instruction List)* — список инструкций. Это типичный ассемблер с аккумулятором и переходами по меткам. Набор инструкций стандартизован и не зависит от конкретной целевой платформы. Поскольку *IL* самый простой в реализации язык, он получил очень широкое распространение до принятия стандарта МЭК. *Я*зык *IL* позволяет работать с любыми типами данных, вызывать функции и функциональные блоки, реализованные на любом языке. Таким образом, на *IL* можно реализовать алгоритм любой сложности, хотя текст будет достаточно громоздким.

В составе МЭК-языков *IL* применяется при создании компактных компонентов, требующих тщательной проработки, на которую не жалко времени. При работе с *IL* гораздо адекватнее, чем с другими языками, можно представить, как будет выглядеть оттранслированный код. Благодаря чему *IL* выигрывает там, где нужно достичь наивысшей эффективности. К компиляторам это относится в полной мере.

Текст на *IL* — это текстовый список последовательных инструкций. Каждая инструкция записывается на отдельной строке. Инструкция может включать четыре поля, разделенные пробелами или знаками табуляции:

Метка: Оператор Операнд Комментарий

Метка инструкции не является обязательной, она ставится только там, где нужно. Оператор присутствует обязательно. Операнд необходим почти всегда. Комментарий — необязательное поле, записывается в конце строки. Ставить комментарии между полями инструкции нельзя. Пример *IL*-программы:

```
МЕТКА1:  LD      Sync      (*пример IL*)
         AND     Start
         S       Q
```

Аккумулятор *IL* является универсальным контейнером, способным сохранять значения переменных любого типа. Команды сравнения сравнивают значение операнда и аккумулятора, результат сравнения ИСТИНА или ЛОЖЬ вновь помещается в аккумулятор.

Программа на *IL* выполняется подряд, сверху вниз. Для изменения порядка выполнения и организации циклов применяется **переход на метку.** Переход на метку может быть безусловным **JMP** — выполняется всегда, независимо от чего-либо. Условный переход **JMPC** выполняется только при значении аккумулятора ИСТИНА. Переход можно выполнять как вверх, так и вниз.

Стандартные операторы, модификаторы см. в [1].

1.7.3. *Структурированный текст ST*

Язык *ST (Structured Text)* — это язык высокого уровня. Синтаксически *ST* представляет собой несколько адаптированный язык Паскаль. Вместо процедур Паскаля в *ST* используются компоненты программ стандарта МЭК.

Для специалистов, знакомых с языком *C*, освоение *ST* также не вызовет никаких сложностей.

Сравнение эквивалентных программ на языках *ST* и *C*:

ST:

```
WHILE Counter <> 0 DO
    Counter := Counter-1;
    Varl:= Varl*2;

IF Varl > 100 THEN
     Varl := 1;
     Var2 := Var2 + 1;
   END_IF
END_WHILE
```

C:

```
while (Counter-!=0)
{
Varl *= 2;

if (Var1>100)
        {
        Varl = 1;
        ++ Varl2;
        }
}/*while*/
```

В большинстве комплексов программирования ПЛК язык *ST* по умолчанию предлагается для описания действий и условий переходов *SFC*. Это наилучшее сочетание языков, позволяющее эффективно решать любые задачи.

Основой *ST*-программы служат выражения. Результат вычисления выражения присваивается переменной при помощи оператора **«:=»**, как и в Паскале. Каждое выражение обязательно заканчивается точкой с запятой «;». Выражение состоит из переменных констант и функций, разделенных операторами:

iVarl := 1 + iVar2 / ABS(iVar2);

Стандартные операторы в выражениях *ST* имеют символьное представление, например математические действия: **+**, *****, **/**, операции сравнения и т. д.

Помимо операторов, элементы выражения можно отделять пробелами и табуляциями для лучшего восприятия. В текст могут быть введены комментарии. Везде, где допустимы пассивные разделители, можно вставлять и комментарии:

iVarl := 1 + (*получить знак*) iVar2 / ABS(iVar2); (*проверка на 0 была выше*)

Несколько выражений можно записать подряд в одну строку. Но хорошим стилем считается запись одного выражения в строке. Длинные выражения можно перенести на следующую строку. Перенос строки равноценен пассивному разделителю.

Выражение может включать другое выражение, заключенное в скобки. Выражение, заключенное в скобки, вычисляется в первую очередь.

Тип выражения определяется типом результата вычислений:

bAlarm := bylnp1 > bylnp2 AND bylnp1 + bylnp2 <> 0 OR bAlarm2;

Вычисление выражения происходит в соответствии с правилами **приоритета операций.** Первыми выполняются операции с наивысшим приоритетом.

В порядке уменьшения приоритета операции располагаются так: выражение в скобках; вызов функции; степень **EXPT**; замена знака (-); отрицание **NOT** умножение, деление и деление по модулю **MOD**; сложение и вычитание **(+, -)**; операции сравнения **(<, >, <=, >=)**; равенство **(=)**; неравенство **(<>)**; логические операции **AND, XOR** и **OR**.

Пустое выражение состоит из точки с запятой «;» Для точки с запятой транслятор не генерирует никакого кода. Если случайно поставить лишнюю **«;»,** это не вызовет ошибки.

Оператор выбора (IF-Если) позволяет выполнить различные группы выражений в зависимости от условий, выраженных логическими выражениями. Полный синтаксис оператора IF (если) выглядит так:

IF <логическое выражение IF>
THEN
<выражения IF> ;
[

ELSIF <логическое выражение ELSEIF 1>
THEN
<выражения ELSEIF 1> ;

ELSIF <логическое выражение ELSEIF n>
THEN
<выражения ELSEIF n> ;
ELSE
<выражения ELSE> ;
]
END_IF

Если <логическое выражение IF> ИСТИНА, то выполняются выражения первой группы — <выражения IF>. Прочие выражения пропускаются, альтернативные условия не проверяются.

Часть конструкции в квадратных скобках является необязательной и может отсутствовать.

Если <логическое выражение IF> ЛОЖЬ, то одно за другим проверяются условия ELSIF. Первое истинное условие приведет к выполнению соответствующей группы выражений. Прочие условия ELSIF анализироваться не будут. Групп ELSIF может быть несколько или не быть совсем.

Если все логические выражения дали ложный результат, то выполняются выражения группы ELSE, если она есть. Если группы ELSE нет, то не выполняется ничего.

Циклы WHILE и **REPEAT** обеспечивают повторение группы выражений, пока верно условное логическое выражение. Если условное выражение всегда истинно, то цикл становится бесконечным.

Синтаксис WHILE:

WHILE <Условное логическое выражение> **DO** <Выражения — тело цикла>
END_WHILE

Условие в цикле WHILE проверяется до начала цикла. Если логическое выражение изначально имеет значение ЛОЖЬ, тело цикла не будет выполнено ни разу.

Синтаксис REPEAT:

REPEAT
<Выражения — тело цикла >
UNTIL <Условное логическое выражение>
END_REPEAT

Условие в цикле REPEAT проверяется после выполнения тела цикла. Если логическое выражение изначально имеет значение ЛОЖЬ, тело цикла будет выполнено один раз.

Пример:

```
ci := 64;
WHILE ci > 1 DO
        Varl := Varl + 1;
        ci := ci/2;
END_WHILE
```

Цикл **FOR** обеспечивает заданное количество повторений группы выражений. Синтаксис:

FOR <Целый счетчик> := <Начальное значение>

TO <Конечное значение>

[**BY** <Шаг>] **DO**

<Выражения — тело цикла>

END__FOR

Перед выполнением цикла счетчик получает начальное значение. Далее тело цикла повторяется, пока значение счетчика не превысит конечного значения. Счетчик увеличивается в каждом цикле. Начальное и конечное значения и шаг могут быть как константами, так и выражениями.

Счетчик изменяется после выполнения тела цикла. Поэтому, если задать конечное значение меньшее начального, то при положительном приращении цикл не будет выполнен ни разу. При одинаковых начальном и конечном значениях тело цикла будет выполнено один раз.

Часть конструкции **BY** в скобках необязательна, она определяет шаг приращения счетчика. По умолчанию счетчик увеличивается на единицу в каждой итерации. В качестве счетчика можно использовать переменную любого целого типа.

Пример:

```
Varl := 0;
FOR cw := 1 TO 10 DO Varl
    := Varl + 1;
END_FOR
```

Данный цикл будет выполнен 10 раз и соответственно Varl будет иметь значение 10.

Цикл FOR исключительно удобен для итераций с заранее известным числом повторов.

Для построения правильного цикла достаточно соблюдать два простых формальных требования:

- не изменяйте счетчик цикла и условие окончания в теле цикла. Счетчик и переменные, образующие конечное условие в цикле, можно использовать только для чтения;
- не задавайте в качестве конечного условия максимальное для типа переменной счетчика значение. Так, если для однобайтного целого без знака задать константу 255, то условие окончания не будет выполнено никогда. Цикл станет бесконечным.

Оператор **EXIT**, помещенный в теле циклов WHILE, REPEAT и FOR, приводит к немедленному окончанию цикла. Хороший стиль программирования призывает избегать такого приема, но иногда он весьма удобен.

Оператор **RETURN** осуществляет немедленный возврат из POU. Это единственный способ прервать вложенные итерации без введения дополнительных проверок условий. Оператор RETURN выполняется очень быстро, фактически это одна машинная команда процессора.

Иногда бывает удобно создать безусловный цикл, а условия выхода формировать в теле цикла с использованием EXIT. Например, могут потребоваться несколько равновероятных, но невзаимосвязанных условий выхода из цикла. Создать безусловный (бесконечный) цикл в *ST* проще всего так:

WHILE TRUE **DO...**

1.7.4. *Релейные диаграммы LD*

LD (Ladder Diagram) язык релейных диаграмм или релейно-контактных схем (РКС) — графический язык, реализующий структуры электрических цепей. РКС — это американское изобретение. В начале 70-х годов XX в. релейные автоматы сборочных конвейеров начали постепенно вытесняться программируемыми контроллерами. Некоторое время те и другие работали одновременно и обслуживались одними и теми же людьми.

Так появилась задача прозрачного переноса релейных схем в ПЛК. Различные варианты программной реализации релейных схем создавались практически всеми ведущими производителями ПЛК. Благодаря простоте представления РКС обрел заслуженную популярность, что и стало основной причиной включения его в стандарт МЭК.

Графически *LD*-диаграмма представлена в виде двух вертикальных шин питания. Между ними расположены цепи, образованные соединением контактов (рис. 1.12). Нагрузкой каждой цепи служит реле. Каждое реле имеет контакты, которые можно использовать в других цепях. Количество контактов в цепи произвольно, реле одно. Если последовательно соединенные контакты замкнуты, ток идет по цепи и реле включается. При необходимости можно включить параллельно несколько реле, последовательное включение не допускается.

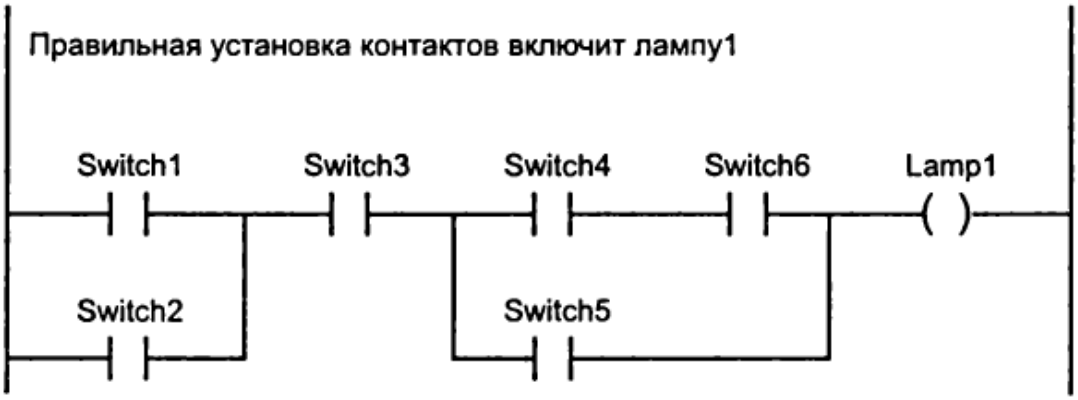


Рис. 1.12. Диаграмма *LD*

В *LD* каждому контакту ставится в соответствие логическая переменная, определяющая его состояние. Если контакт замкнут, то переменная имеет значение ИСТИНА. Если разомкнут ЛОЖЬ. Имя переменной пишется над контактом и фактически служит его названием.

Последовательное соединение контактов или цепей равноценно логической операции И. Параллельное соединение образует монтажное ИЛИ.

Цепь может быть либо замкнутой (ON), либо разомкнутой (OFF). Это отражается на обмотке реле и соответственно на значении логической переменной обмотки (ИСТИНА/ЛОЖЬ).

Приведенная на рис. 1.12 схема эквивалентна выражению

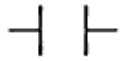
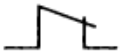
Lamp1 := (Switch1 OR Switch2) AND Switch3 AND ((Switch4 AND Switch6) OR Switch5);

Зрительное восприятие *LD*-диаграмм должно быть интуитивно понятным. Преимущество состоит в возможности применения символов псевдографики для построения *LD*-диаграмм.

Сопоставление обозначений базовых элементов *LD* и обозначений ЕСКД приведено в табл. 1.4.

Таблица 1.4

Обозначения базовых элементов

LD	ЕСКД	Обозначение
		Нормально разомкнутый контакт
		Нормально замкнутый контакт
		Обмотка реле

Контакт может быть инверсным — нормально замкнутым. Такой контакт обозначается с помощью символа |/| и замыкается, если значение переменной ЛОЖЬ. Инверсный контакт равнозначен логической операции НЕ.

Переключающий контакт образуется комбинацией прямого и инверсного контактов (рис. 1.13).

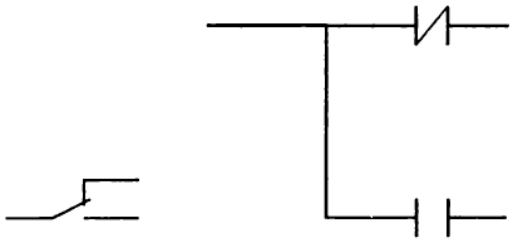


Рис. 1.13. Переключающий контакт

Обмотки реле также могут быть инверсными, что обозначается символом **(/).** Если обмотка инверсная, то в соответствующую логическую переменную копируется инверсное значение состояния цепи.

Помимо обычных реле, в релейных схемах часто применяются поляризованные реле. Такое реле имеет две обмотки, переключающие его из одного положения в другое. Переключение производится импульсами тока. При отключении тока питания поляризованное реле остается в заданном положении, что реализует элементарную ячейку памяти.

В *LD* такое реле реализуется при помощи двух специальных обмоток **SET** и **RESET**. Обмотки типа SET обозначаются буквой *S* внутри круглых скобок *(S).* Обмотки типа RESET обозначаются буквой *R.* Если соответствующая обмотке *(S)* переменная принимает значение ИСТИНА, то сохраняет его бесконечно. Вернуть данную переменную в ЛОЖЬ можно только обмоткой *(R).*

Логически последовательное (И), параллельное (ИЛИ) соединение контактов и инверсия (НЕ) образуют базис Буля. В результате *LD* идеально подходит не только для построения релейных автоматов, но и

для программной реализации комбинационных логических схем. Благодаря возможности включения в *LD* функций и функциональных блоков, выполненных на других языках, сфера применения языка практически не ограничена.

1.7.5. *Функциональные диаграммы FBD*

FBD (Function Block Diagram) – это графический язык программирования. Диаграмма *FBD* очень напоминает принципиальную схему электронного устройства на микросхемах (рис. 1.14). В отличие от *LD* «проводники» в *FBD* могут проводить сигналы (передавать переменные) любого типа (логический, аналоговый, время и т. д.). Иногда говорят, что в релейных схемах соединительные проводники передают энергию. Проводники *FBD* тоже передают энергию, но в более широком смысле. Очевидно, что шины питания и контакты здесь уже неэффективны. Шины питания на *FBD* диаграмме не показываются. Выходы блоков могут быть поданы на входы других блоков либо непосредственно на выходы ПЛК. Сами блоки, представленные на схеме как «черные ящики», могут выполнять любые функции.

FBD-схемы очень четко отражают взаимосвязь входов и выходов диаграммы. Если алгоритм изначально хорошо описывается с позиции сигналов, то его *FBD*-представление всегда получается нагляднее, чем в текстовых языках.

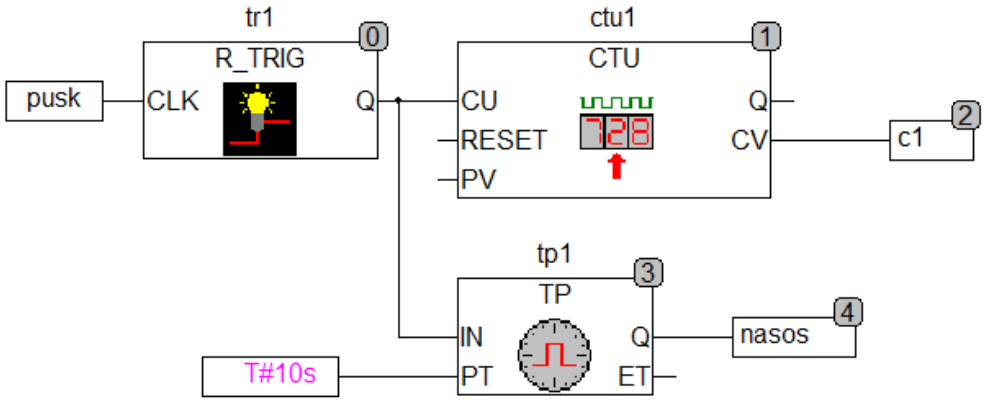


Рис. 1.14. Диаграмма *FBD*

Диаграмма *FBD* строится из компонентов, отображаемых на схеме прямоугольниками. Входы *POU* изображаются слева от прямоугольника, выходы справа. Внутри прямоугольника указывается тип *POU* и наименования входов и выходов. Для экземпляра функционального блока его наименование указывается сверху, над прямоугольником. В графических системах программирования прямоугольник компонента может содержать картинку, отражающую его тип. Размер прямоугольника

зависит от числа входов и выходов и устанавливается графическим редактором автоматически.

Программа в *FBD* не обязательно должна представлять большую единую схему. Как и в *LD*, диаграмма образуется из множества цепей, выполняемых одна за другой.

Прямоугольники *POU* в *FBD* соединены линиями связи. Соединения имеют направленность слева направо. Вход блока может быть соединен с выходом блока, расположенного слева от него. Помимо этого, вход может быть соединен с переменной или константой. Соединение должно связывать переменные или входы и выходы одного типа. В отличие от компонента переменная изображается на диаграмме без прямоугольной рамки. Ширина соединительной линии в *FBD* роли не играет.

2. ПОДГОТОВИТЕЛЬНАЯ РАБОТА И ВЫПОЛНЕНИЕ ПРОГРАММИРОВАНИЯ ПЛК В *CODESYS*

До выполнения практических работ необходимо следующее:

1. Установить ПО *CoDeSys* версия 2.3.9.41 (Русифицированная версия) по ссылке: http://www.owen.ru/catalog/codesys_v2/opisanie [3].
2. Установить Автоматический установщик библиотек (ПЛК100/150/154/110/160/63/73).
3. Установить установщик *Target* файлы для следующих моделей ПЛК: ПЛК 100, ПЛК 150, ПЛК 154.
4. Ознакомится с «Руководством пользователя по программированию ПЛК в *CodeSys* 2.3» [4-6].
5. Ознакомиться с видео-уроками.

Для создания собственного проекта необходимо:

1. Определить конфигурацию ПЛК в соответствии с аппаратными средствами своего контроллера.
2. Создать программные компоненты, необходимые для решения проблемы.
3. Написать программный код для созданных компонентов на выбранных языках.
4. После завершения программирования, скомпилировать проект и исправить ошибки, если они есть.
5. Приступить к отладке. Для этого:

- включите флажок эмуляция (***Simulation***) и «подключитесь» к контроллеру. Теперь вы в режиме ***Online***;
- откройте окно с конфигурацией ПЛК (***PLC Configuration***) и проверьте правильность выполнения проекта. Для этого измените вручную входные данные и убедитесь, что выходы контроллера отреагировали нужным образом.

В случае ошибок в работе кода вы можете задать точки останова. Когда процесс остановлен в определенной точке, вы можете просмотреть значения переменных проекта в данный момент времени. Выполняя проект в пошаговом режиме (***single step***), вы можете проверить логическую корректность своих программ.

Для программирования разрешается использование любых языков программирования, однако рекомендуется использовать язык релейных диаграмм (*LD*).

3. *Практическая работа № 1*
РЕАЛИЗАЦИЯ РАБОТЫ РЕВЕРСИВНОГО ПУСКАТЕЛЕ НА КОНТРОЛЛЕРЕ *PLC 150.I-M (CODESYS)*

3.1. Краткие теоретические сведения

Для запуска электродвигателя в прямом и обратном направлении применяется реверсивная схема управления на магнитном пускателе.

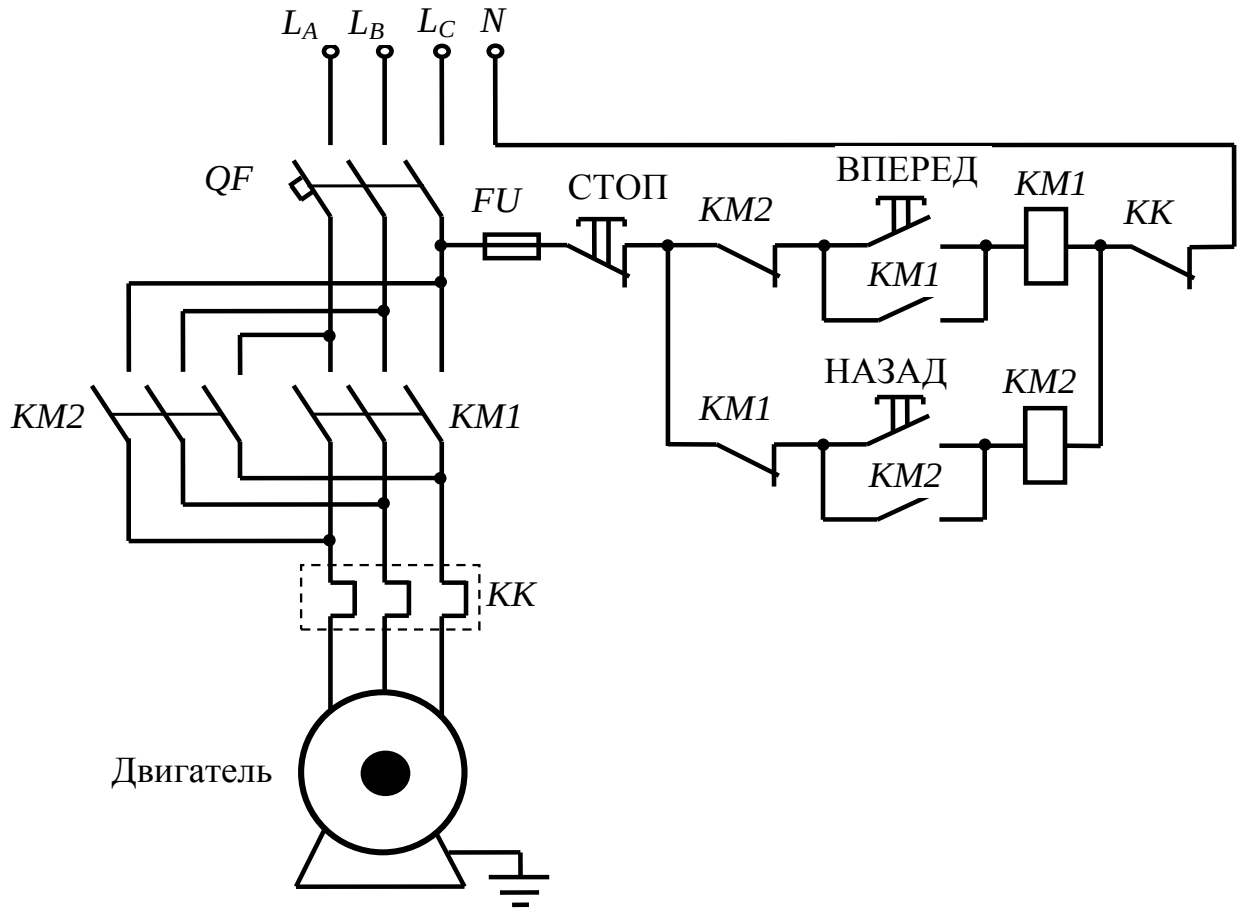


Рис. 3.1. Принципиальная схема работы реверсивного пускателя

Фазы *A*, *B* и *C* питающего напряжения подводятся к клеммам асинхронного двигателя через:

- трехполюсный автоматический выключатель, который защищает всю схему и позволяет отключать питающее напряжение;
- поочередно через три пары силовых контактов магнитных пускателей *KM1* и *KM2*;
- тепловое реле Р, которое служит для защиты от перегрузок.

Для изменения направления вращения трехфазного электродвигателя необходимо поменять местами подключение любых двух фаз. Для этого в цепь обмотки двигателя включены силовые контакты от двух пускателей, которые подключаются поочередно, меняя чередование фаз. В схеме (рис. 3.1) при вращении вперед

последовательность фаз такая - *A, B, C,* при вращении назад - *C, B, A*, т.е. чередование фаз *A* и *C* меняется местами.

Катушки магнитных пускателей с одной стороны подключены к нулевому рабочему проводнику *N* через нормально замкнутый контакт теплового реле *KK*, с другой - через кнопочный пост к фазе *C*.

Кнопочный пост состоит из трех кнопок:

- нормально разомкнутой кнопки ВПЕРЕД;
- нормально разомкнутой кнопки НАЗАД;
- нормально замкнутой кнопки СТОП.

К кнопке ВПЕРЕД параллельно подключен нормально разомкнутый вспомогательный контакт пускателя *KM1* и соответственно к кнопке НАЗАД - нормально-разомкнутый вспомогательный контакт пускателя *KM2*.

Также в цепь питания обмотки пускателя *KM1* включен нормально замкнутый контакт пускателя *KM2*, а в цепь обмотки пускателя *KM2* включен нормально замкнутый контакт пускателя *KM1*. Это сделано для блокировки, чтобы предотвратить запуск двигателя назад, когда он вращается вперед, и наоборот. То есть запустить двигатель в любую из сторон можно только из положения останова.

Работа схемы

Переводим рычаг трехполюсного автоматического выключателя во включенное положение, его контакты замыкаются, схема готова к работе.

1. Запуск вперед

Нажимаем кнопку ВПЕРЕД. Цепь питания обмотки магнитного пускателя *KM1* замыкается, якорь катушки втягивается, замыкает силовые контакты *KM1* и вспомогательный нормально открытый контакт *KM1*, который шунтирует кнопку ВПЕРЕД.

Одновременно вспомогательный нормально замкнутый контакт *KM1* размыкает цепь управления магнитным пускателем *KM2*, блокируя тем самым возможность запуска реверса двигателя.

Три питающих фазы в последовательности *A, B, C* подаются на обмотки двигателя, и он начинает вращаться вперед.

Отпускаем кнопку ВПЕРЕД, она возвращается в исходное нормально разомкнутое состояние. Теперь питание на обмотку пускателя *KM1* подается через замкнутый вспомогательный контакт *KM1*. Двигатель запущен и вращается вперед.

2. Останов двигателя из положения вперед

Для остановки двигателя или запуска в другую сторону необходимо сначала нажать кнопку СТОП. Питание цепи управления размыкается. Якорь магнитного пускателя *KM1* под действием пружины возвращается в исходное состояние. Силовые контакты размыкаются, отключая питающее напряжение от электродвигателя. Двигатель останавливается.

Одновременно с этим размыкается вспомогательный контакт *КМ1* в цепи питания обмотки пускателя *КМ1* и замыкается вспомогательный контакт *КМ1* в цепи питания пускателя *КМ2*.

Отпускаем кнопку СТОП. Она возвращается в исходное нормально замкнутое положение. Поскольку вспомогательный контакт *КМ1* разомкнут, питание на обмотку пускателя *КМ1* не подается, двигатель остается выключенным, и схема готова к следующему запуску.

3. Реверс двигателя

Чтобы запустить двигатель в обратном направлении, нажимаем кнопку НАЗАД.

Питание подается на обмотку пускателя *КМ2*. Он срабатывает, замыкая силовые контакты *КМ2* в цепи питания двигателя и вспомогательный контакт *КМ2*, который шунтирует кнопку НАЗАД. Одновременно с этим другой вспомогательный контакт *КМ2* разрывает цепь питания пускателя *КМ2*.

На обмотки двигателя подаются три фазы в порядке *C, B, A,* он начинает вращаться в другую сторону.

Отпускаем кнопку НАЗАД. Она возвращается в исходное положение, но питание на обмотку пускателя *КМ2* продолжает поступать через замкнутый вспомогательный контакт *КМ2*. Двигатель продолжает вращаться в обратном направлении.

4. Останов двигателя из положения назад

Для останова повторно нажимаем кнопку СТОП. Цепь питания обмотки пускателя *КМ2* размыкается. Якорь возвращается в исходное положение, размыкая силовые контакты *КМ2*. Двигатель останавливается. Одновременно с этим вспомогательные контакты *КМ2* возвращаются в исходное состояние.

Отпускаем кнопку СТОП, схема готова к следующему пуску.

5. Защита от перегрузок

Защита от перегрузок выполняется тепловым реле Р.

3.2. Задание

1. Создать коммутационную программу для принципиальной электрической схемы работы электрического двигателя в прямом и обратном направлении.

2. Создать визуализацию (графический интерфейс) работы электродвигателя. Сымитировать вращение двигателя, кнопки ВПЕРЕД, НАЗАД, тепловое реле (*КК*).

Пример оформления визуализации приведен на рис. 3.2.

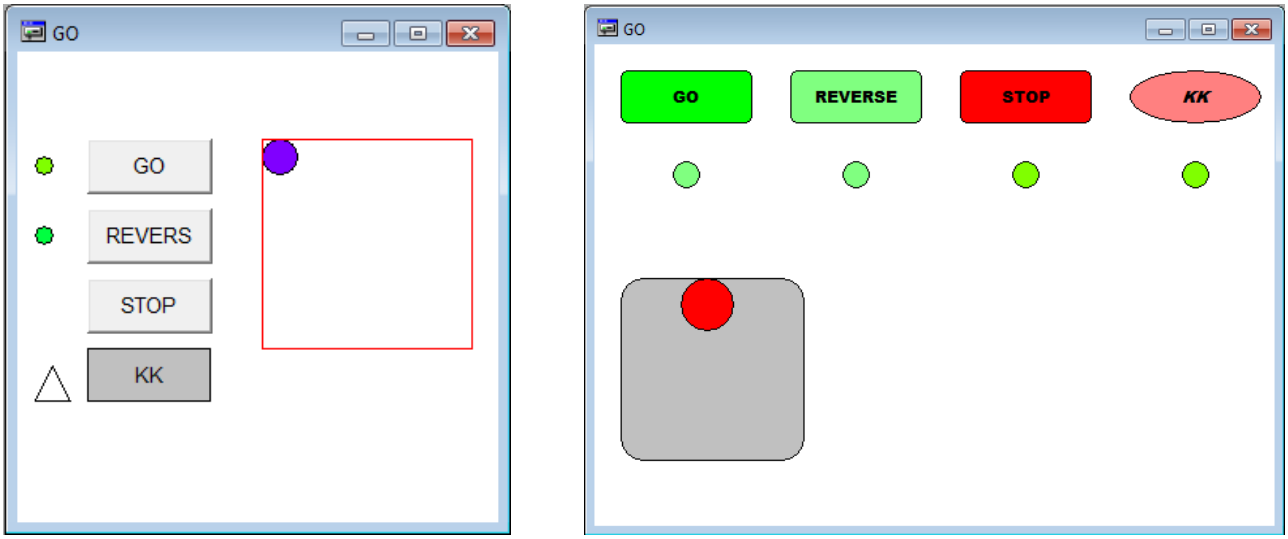


Рис. 3.2. Примеры оформления визуализации

3. Подготовить отчет по плану:

- цель работы;
- описание принципиальной электрической схемы работы электродвигателя. Последовательность действий пуска двигателя в прямом и обратном направлении и при перегрузке ЭД;
- подробное пошаговое словесное описание коммутационной программы (всех элементов, контактов катушек реле);
- описание визуализации и используемых для этого *POU*. Блок схема. Алгоритм построения визуализации;
- вывод (не менее трех).

3.3. Выполнение работы в программе *CodeSys*

Для создания первого *POU* необходимо:

1. Открыть программу с помощью ярлыка *CoDeSys* v 2.3.

2. В открывшейся программе в левом верхнем углу нажать «Файл» - «Создать» (рис. 3.3).

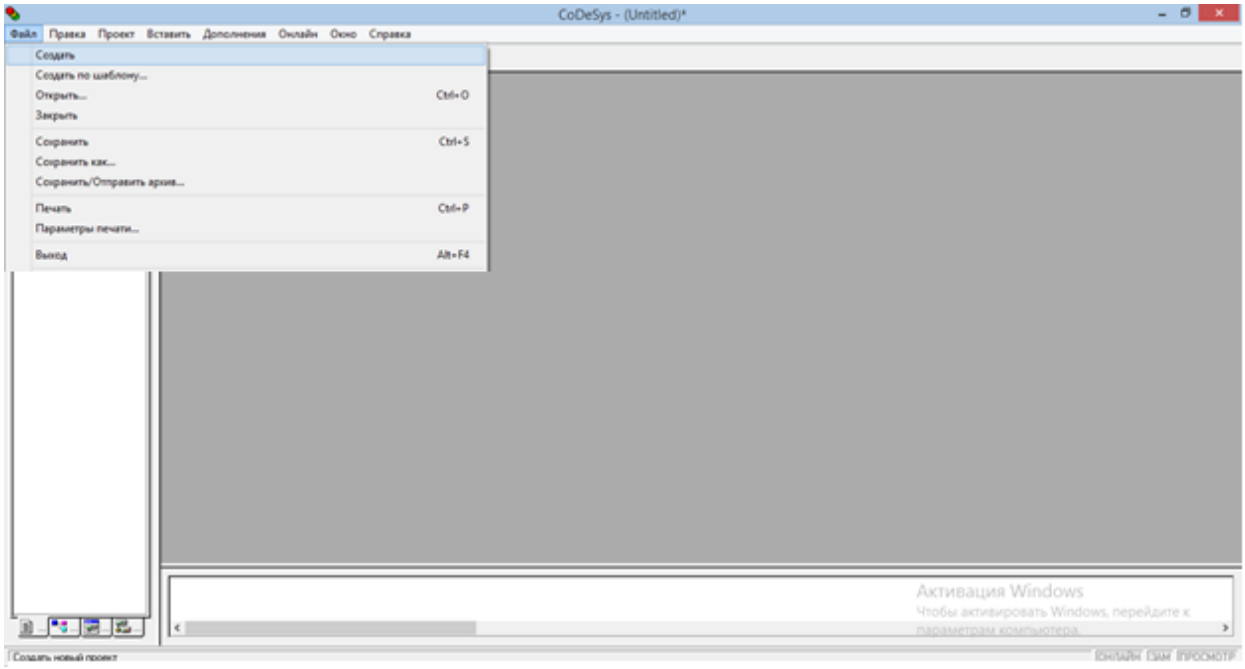


Рис. 3.3. Создание *POU*

3. Предлагается определить конфигурацию контроллера. Открыв выпадающий список, выбрать *PLC150.I-M* и нажать кнопку «*ОК*» (рис. 3.4).

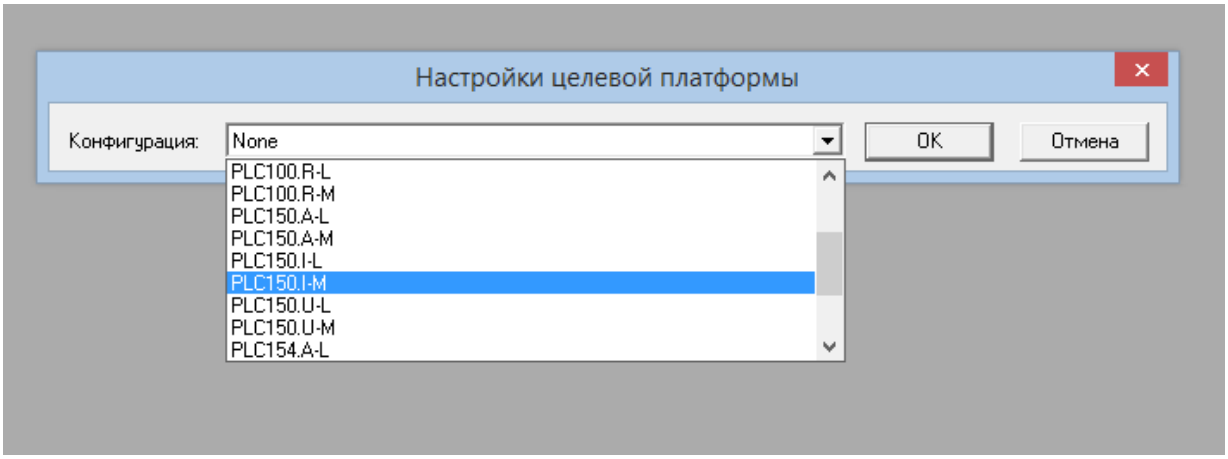


Рис. 3.4. Настройка конфигурации ПЛК

4. Далее, не внося никаких изменений в настройки целевой платформы, нажать кнопку «*ОК*» (рис. 3.5).

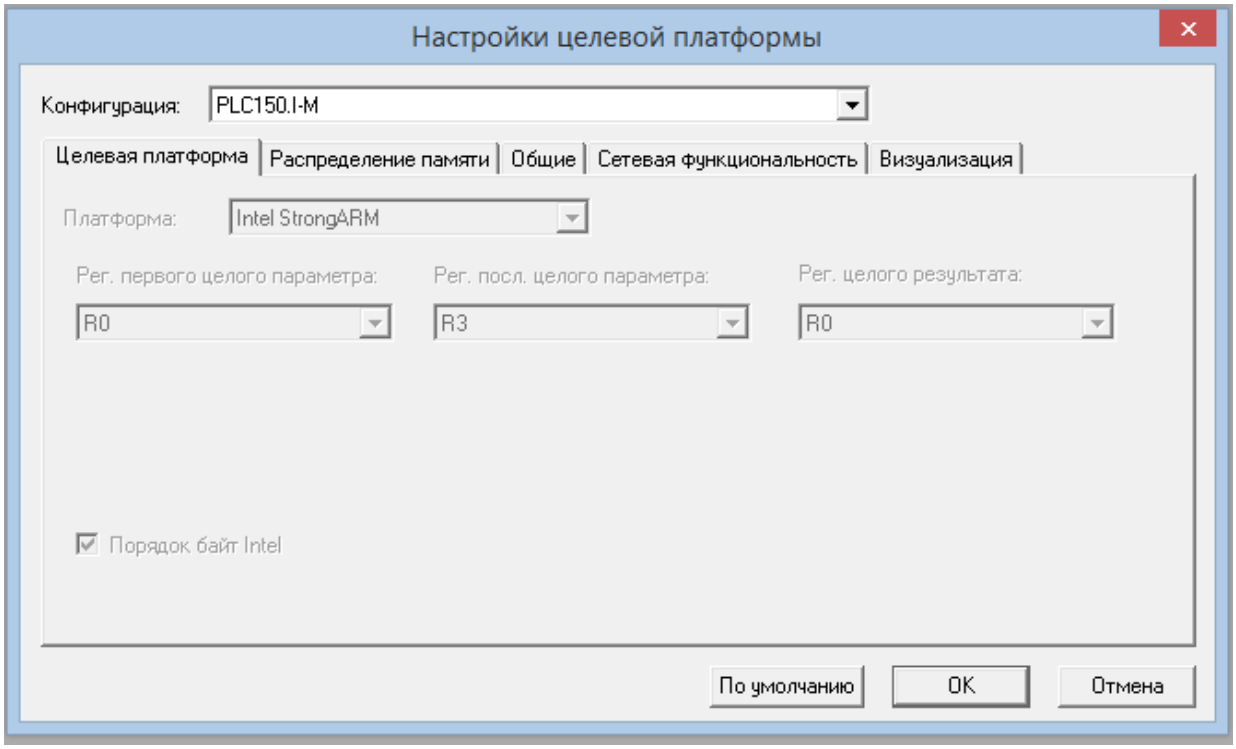


Рис. 3.5. Настройка целевой платформы

5. Новый программный компонент (*POU*):

- «Имя нового *POU*» - оставить без изменений;
- «Тип *POU*» - выбрать «Программа»;
- «Язык реализации» - выбрать «*LD*» (язык лестничных диаграмм);
- Нажать кнопку «*OK*» (рис. 3.6).

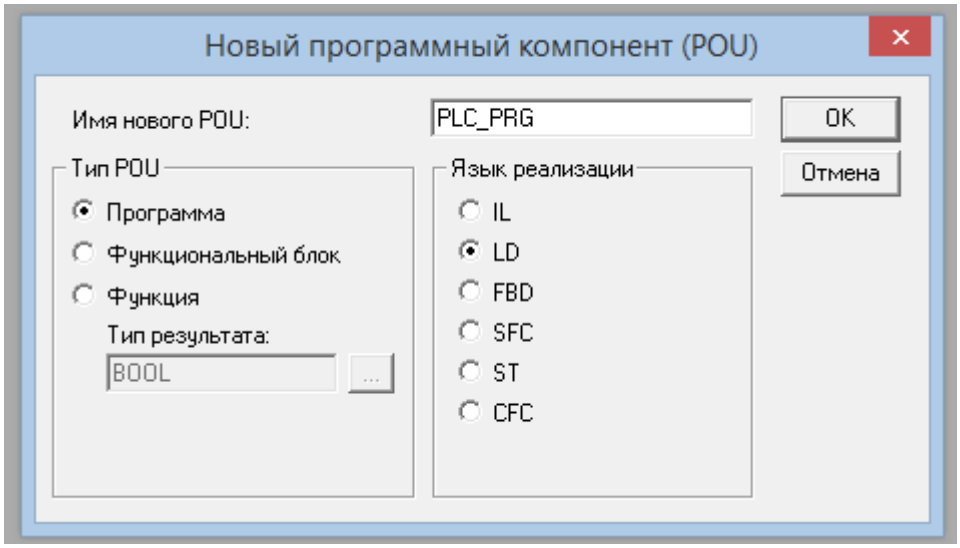


Рис. 3.6. Новый программный компонент *POU*

3.3.1. *Создание программы на языке LD*

1. Во-первых, необходимо задать переменные и их тип. В текстовой области произвести описание «входных» переменных (рис. 3.7, рис. 3.8).

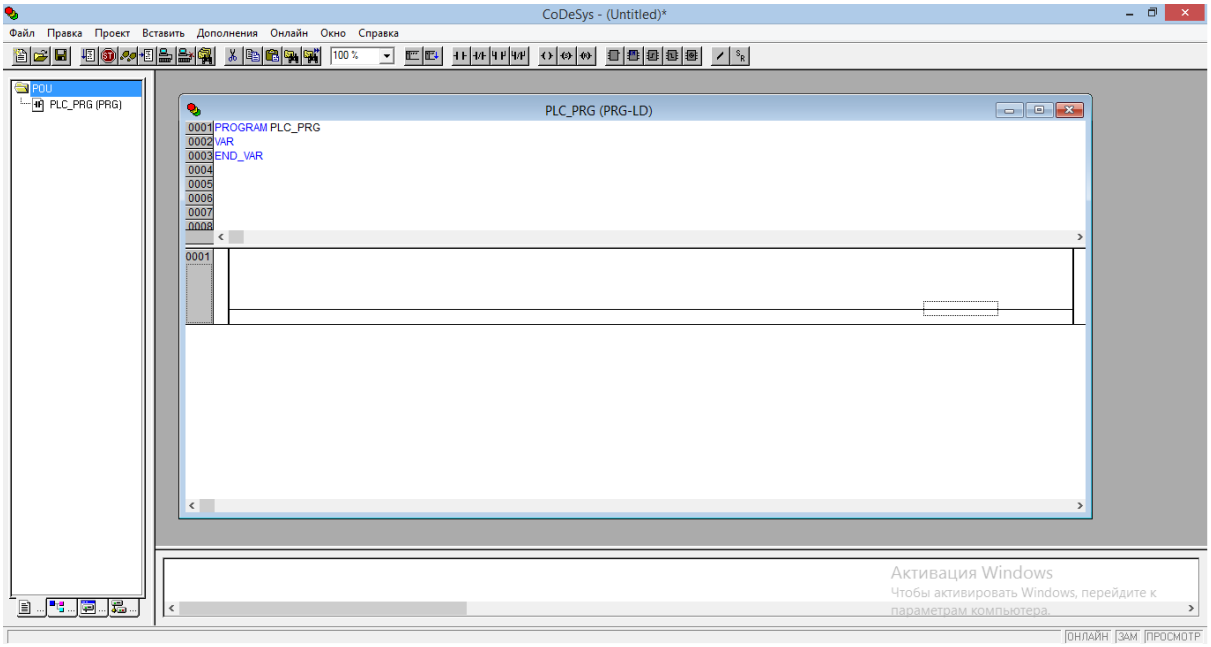


Рис. 3.7. Внешний вид структуры языка *LD*

Для этого введите следующие строки:

```
PROGRAM PLC_PRG
VAR_INPUT
    KK:BOOL;
    STOP:BOOL;
    GO:BOOL;
    RE:BOOL;
END_VAR
```

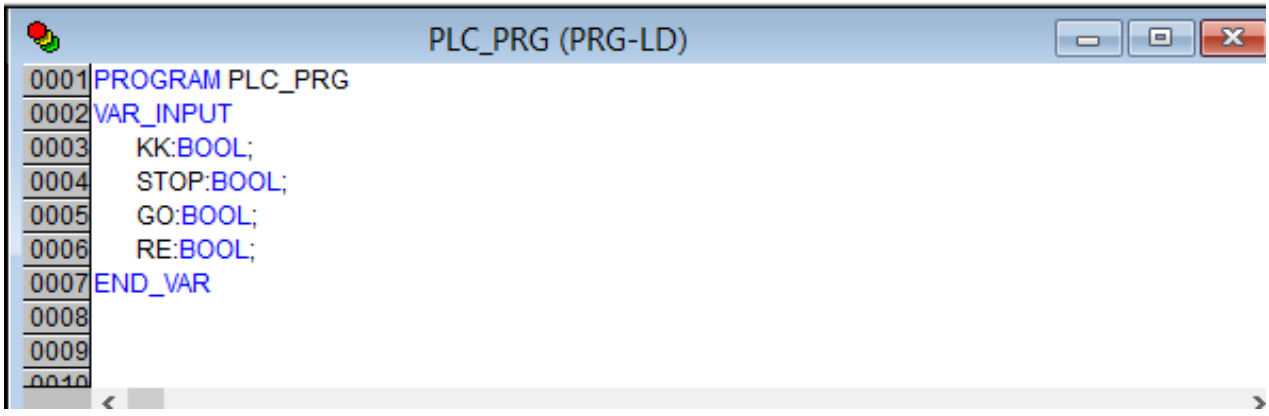


Рис. 3.8. Описание входных переменных

Описание используемых переменных:

KK - входной параметр теплового реле;

STOP - входной параметр кнопки «Стоп»;

GO - входной параметр кнопки «Прямой пуск»;

RE - входной параметр кнопки «Реверс».

2. Помимо «входных», необходимо задать и «глобальные» переменные. Их можно найти, нажав в левом нижнем углу на вкладку «Ресурсы», а затем выбрав папку «Глобальные переменные» и параметр *Global_Variables* (рис. 3.9, рис. 3.10).

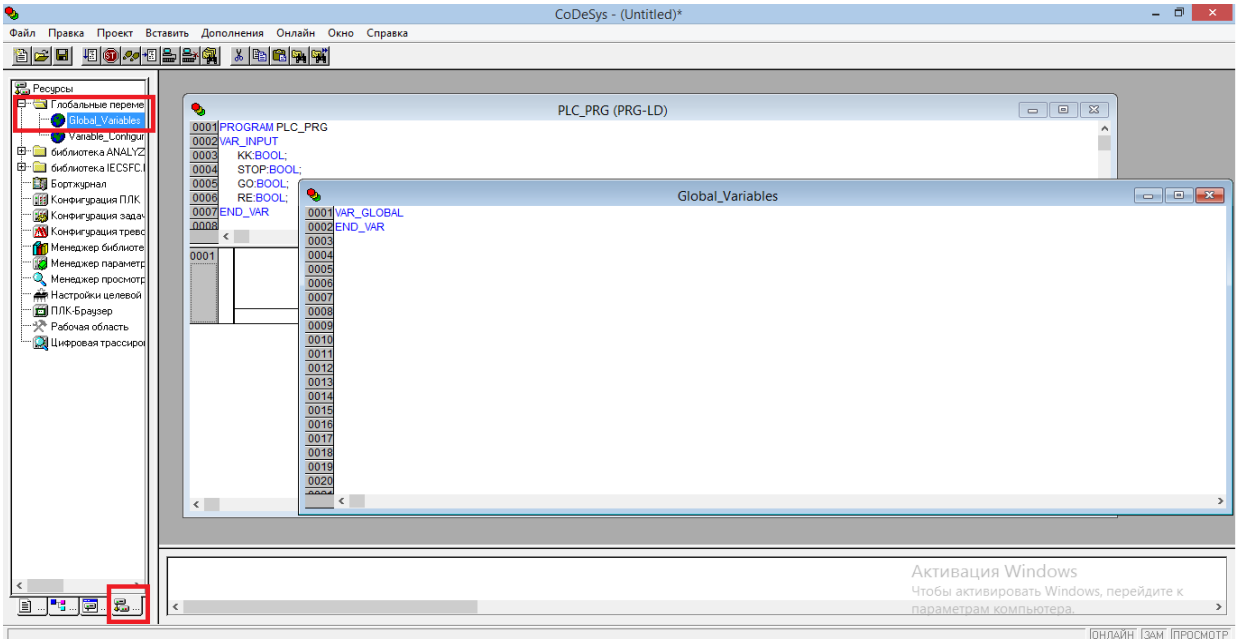


Рис. 3.9. Раздел описания глобальных переменных

В открывшемся текстовом окне опишите «глобальные» переменные. Введите следующие строки:

```
VAR_GLOBAL
    M1:BOOL;
    M2:BOOL;
    X:INT;
    Y:INT;
END_VAR
```

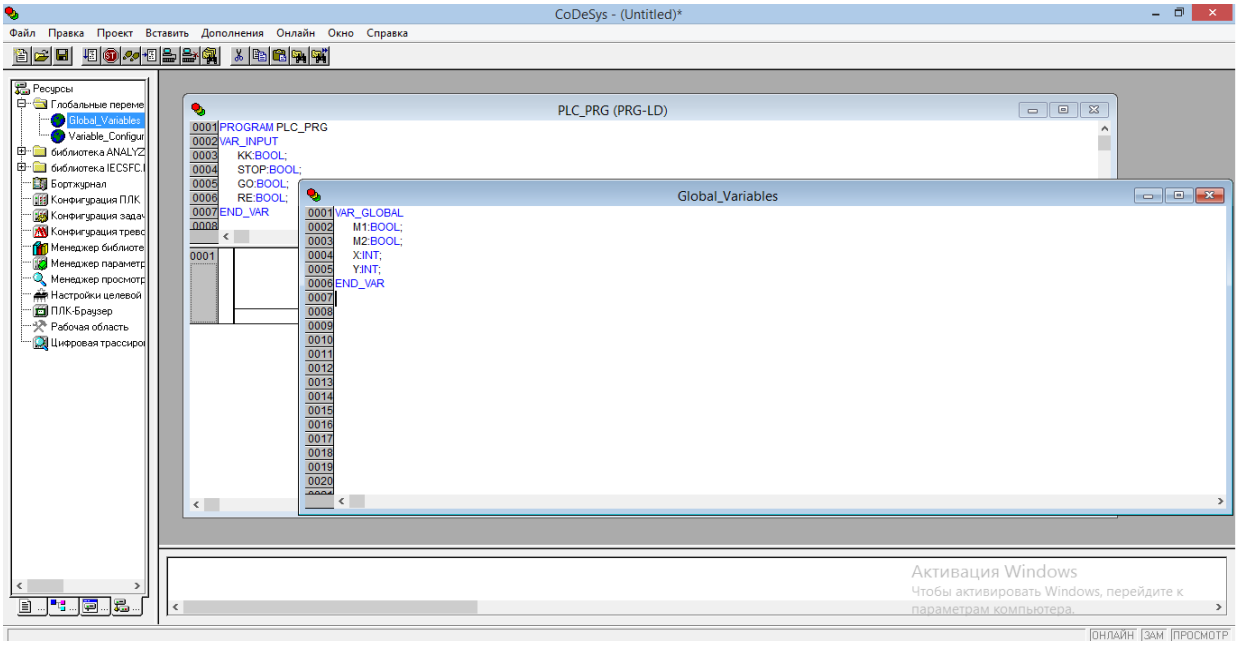


Рис. 3.10. Описание глобальных переменных

Значение используемых глобальных переменных:

M1 - выходной параметр катушки магнитного пускателя *КМ1;*

M2 - выходной параметр катушки магнитного пускателя *КМ2;*

X и *Y* - этот параметр описывает координаты объекта визуализации, который будет создан далее.

3. Можно приступить к нанесению элементов на «лестницу». С помощью «Контактов» и «Инверсных контактов» создайте первую ветвь управления двигателем (рис. 3.11).

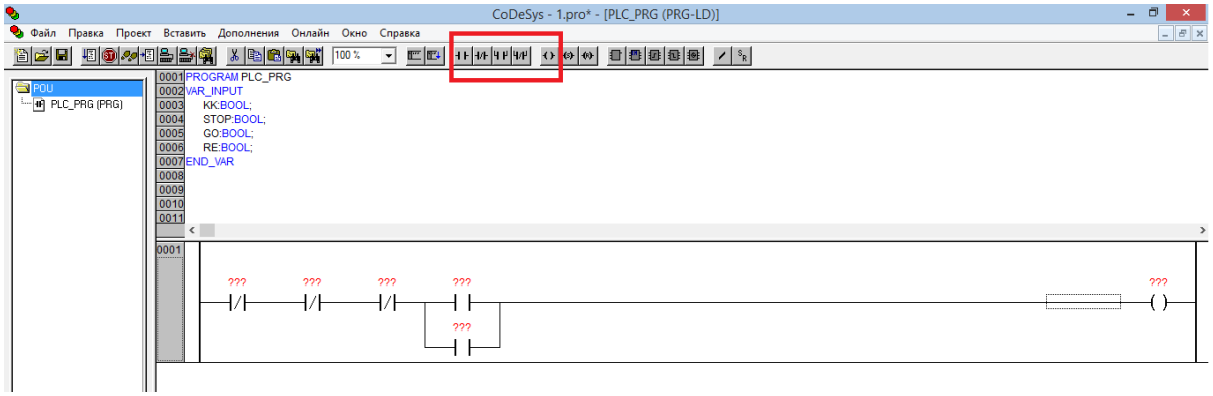


Рис. 3.11. Первая ветвь электрической схемы АД на языке *LD*

4. Для создания второй ветви необходимо нажать правой кнопкой мыши по области «лестницы» и выбрать «Цепь (после)» (рис. 3.12).

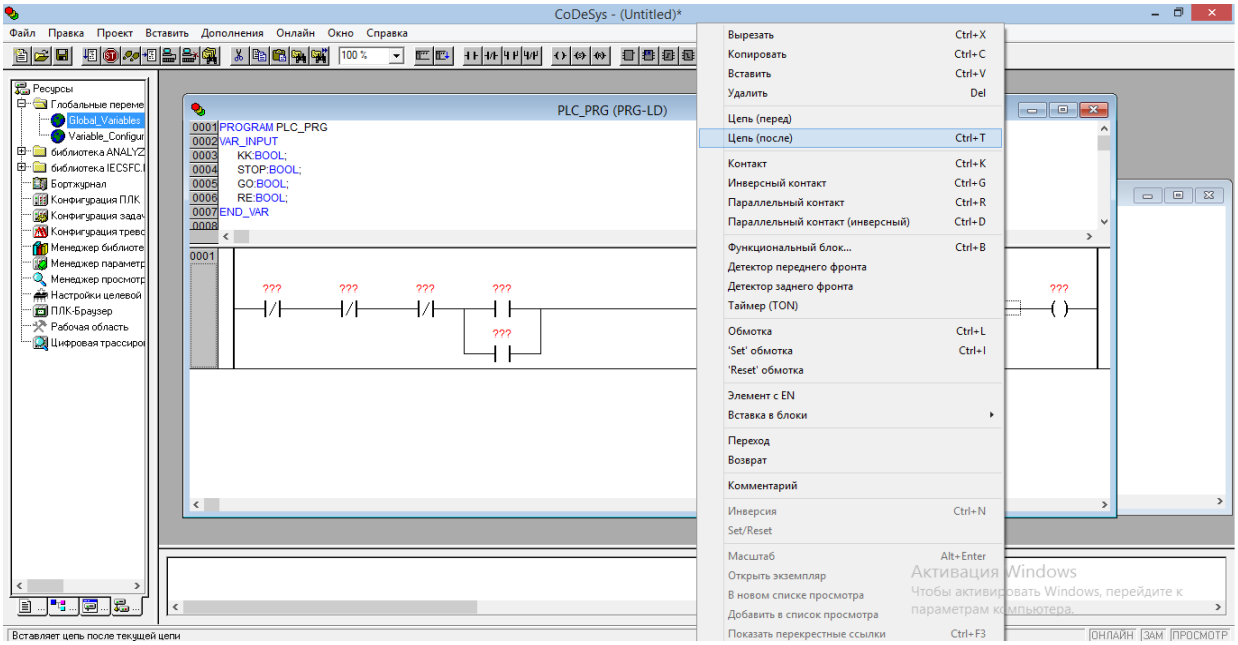


Рис. 3.12. Создание второй ветви электрической схемы

5. Нанесите аналогичным образом элементы схемы на вторую ветвь (рис. 3.13).

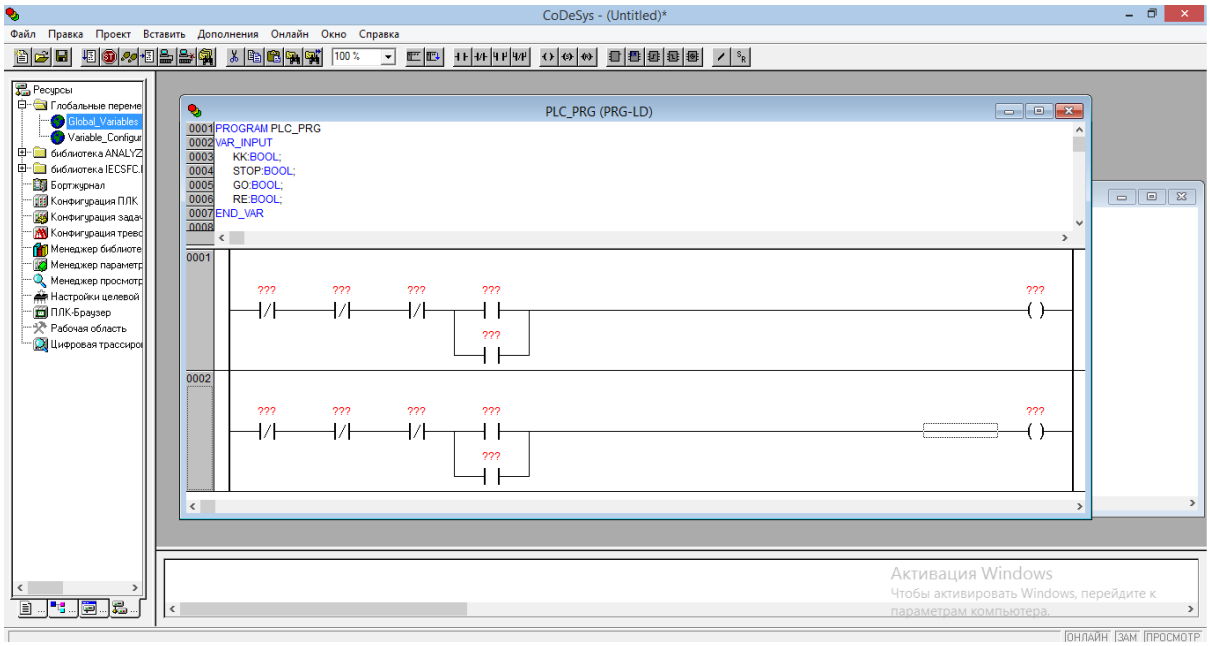


Рис. 3.13. Две ветви электрической сети

6. Присвойте названия элементов, согласно схеме рис. 3.14.

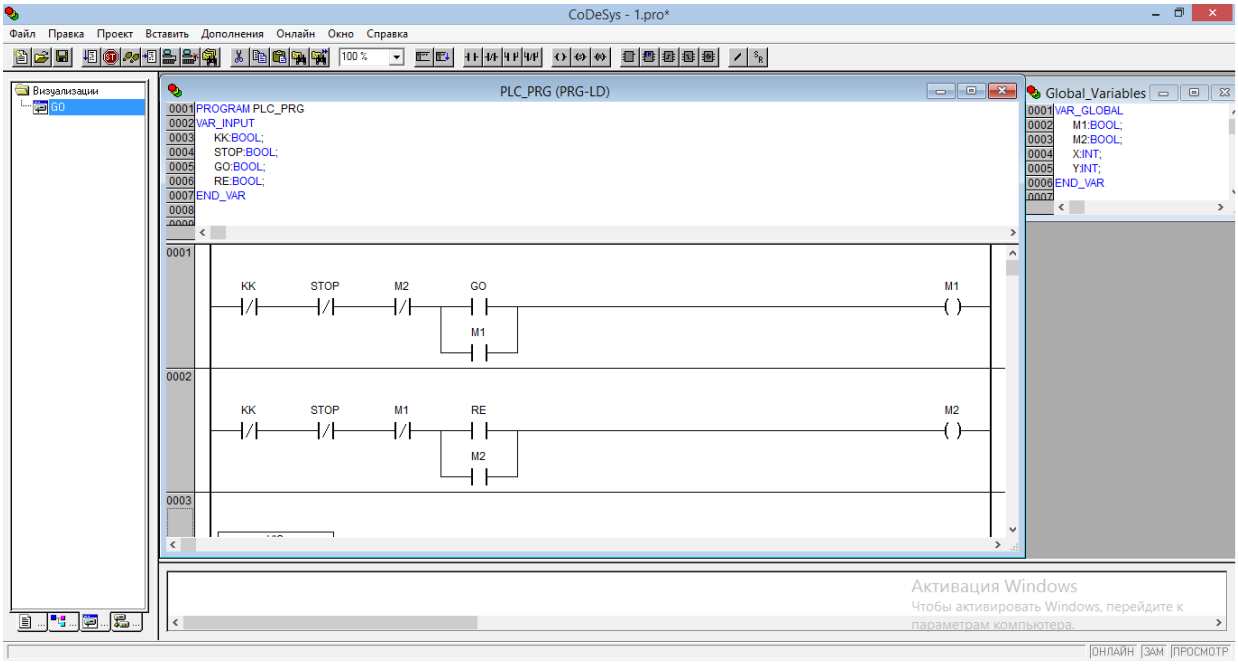


Рис. 3.14. Электрическая схема реверсивного АД на языке *LD*

3.3.2. *Создание программы движения*

1. В левом нижнем углу выберите вкладку «*POU*» и в левой области экрана нажмите на правую кнопку мыши, после чего нажмите «Добавить объект».

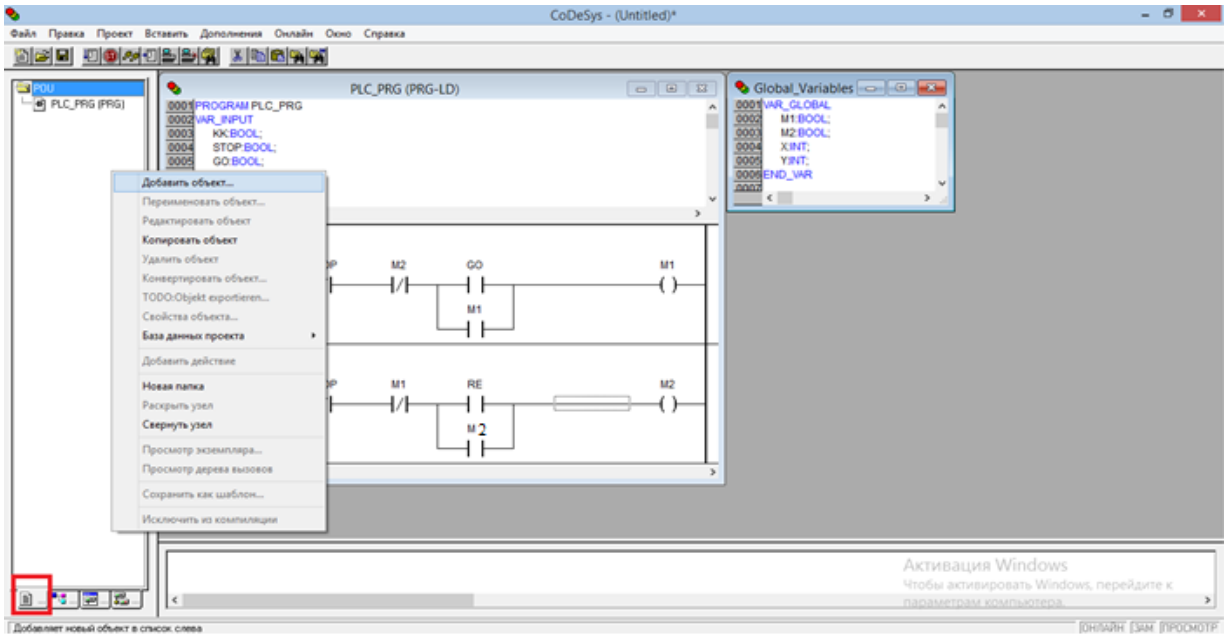


Рис. 3.15. Создание POU визуализации

2. Введите имя подпрограммы и выберете язык реализации *SFC* (рис. 3.16).

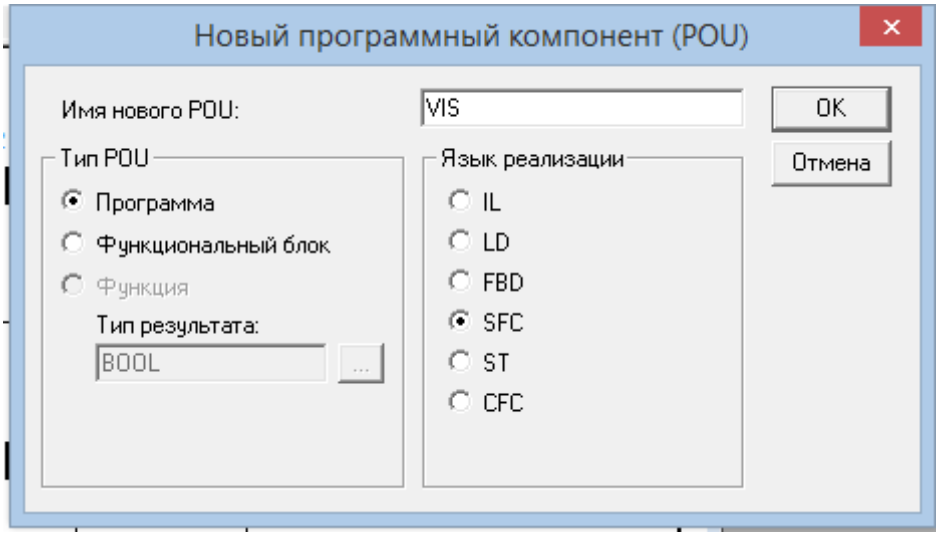


Рис. 3.16. Создание *POU* объекта визуализации работы программы

3. В появившейся программе задайте «входную» переменную, необходимую для объявления программы в главном *POU* (рис. 3.17). В текстовом поле введите следующие строки:

```
PROGRAM VIS
VAR_INPUT
    STARTGO:BOOL;
END_VAR
```

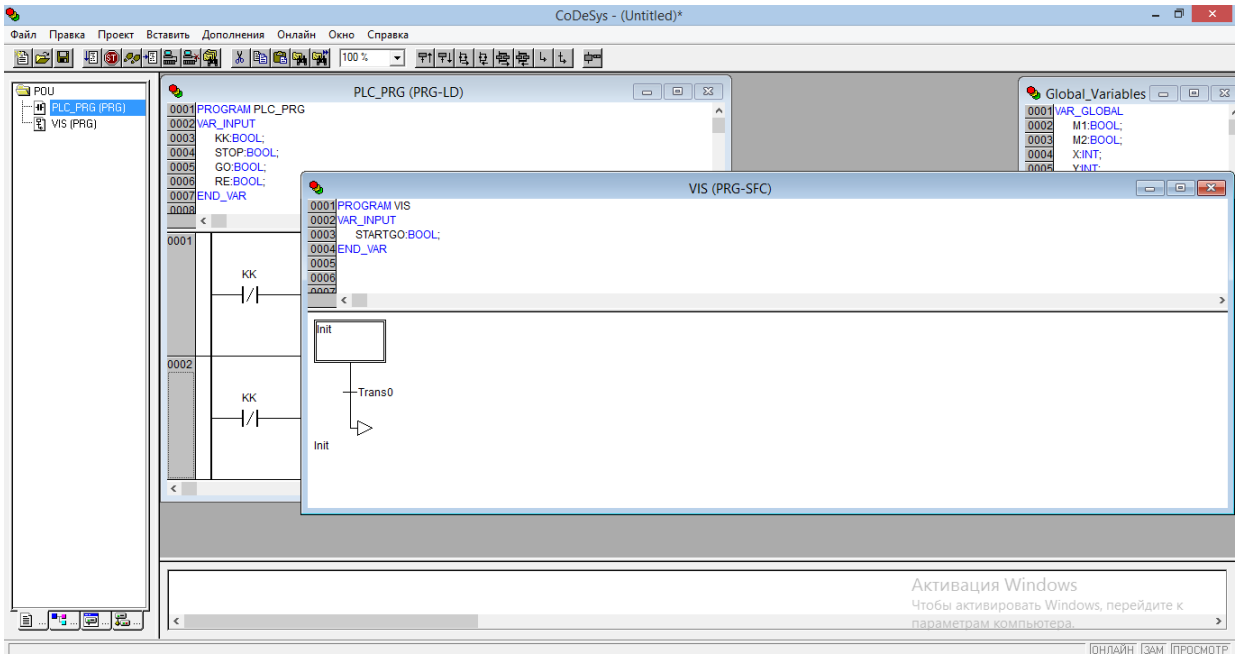


Рис. 3.17. Описание входных переменных в *POU* визуализации

4. Перед написанием кода для этой программы сначала необходимо описать ее в главной программе *PLC_PRG*. Для этого нужно вернуться к

ней, дважды нажав в левой области экрана на соответствующее название программы.

5. Требуется создать еще одну ветвь и разместить на ней «Функциональный блок». В появившемся окне необходимо выбрать вкладку «Пользовательские программы» и программу *«VIS (PRG)»* (рис. 3.18).

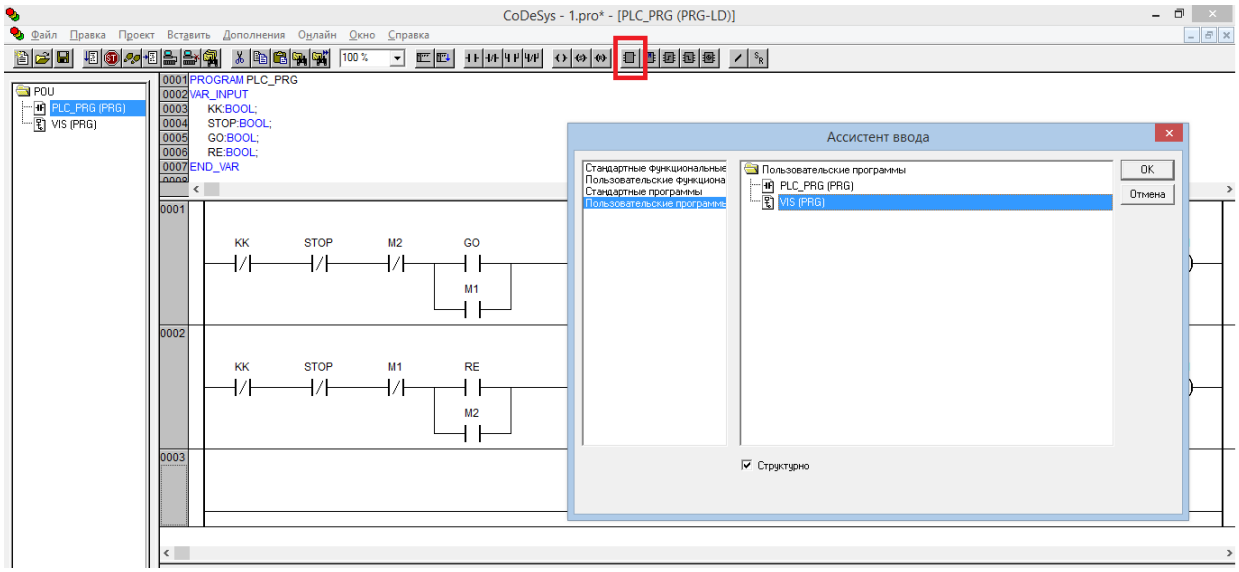


Рис. 3.18. Создание третьей ветви в основном *POU* для функционирования *POU* визуализации

При этом должен появиться функциональный блок с заданным в нем переменной *STARTGO* (рис. 3.19).

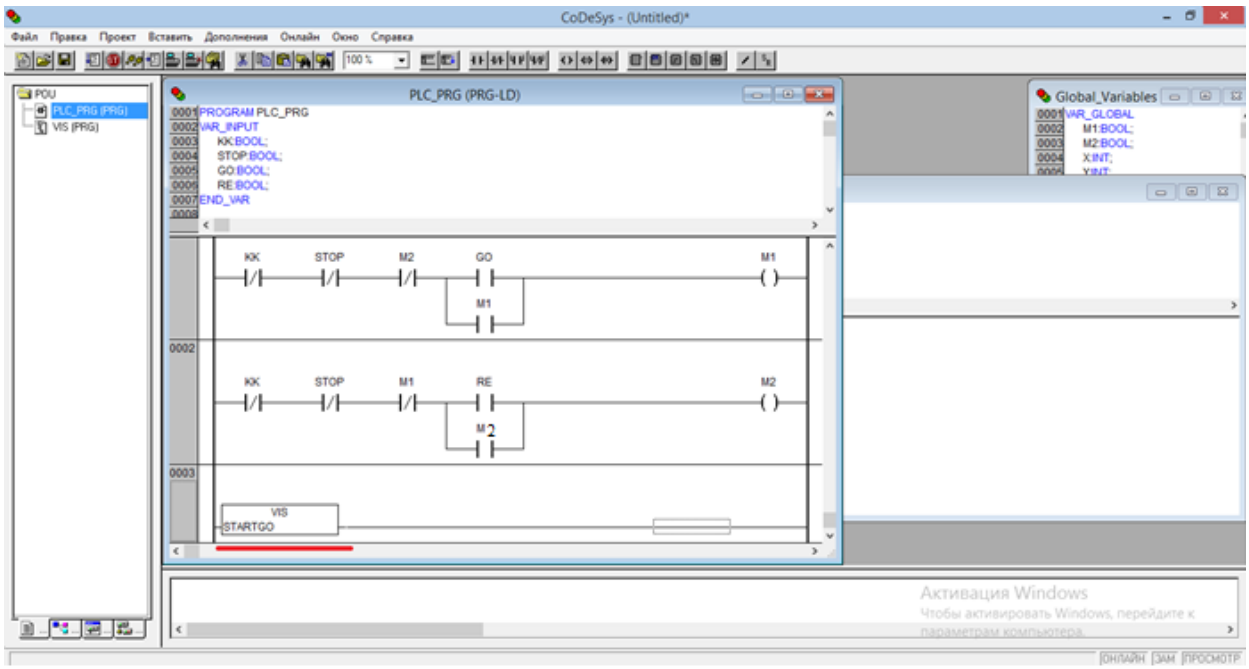


Рис. 3.19. Функциональный блок *POU* визуализации

6. Вернитесь к программе *«VIS (PRG)»*. В области функций нажмите на надпись *«Trans0»*. После того, как она выделится пунктиром, по центру вверху выберите «Альтернативная ветвь (справа)» (рис. 3.20).

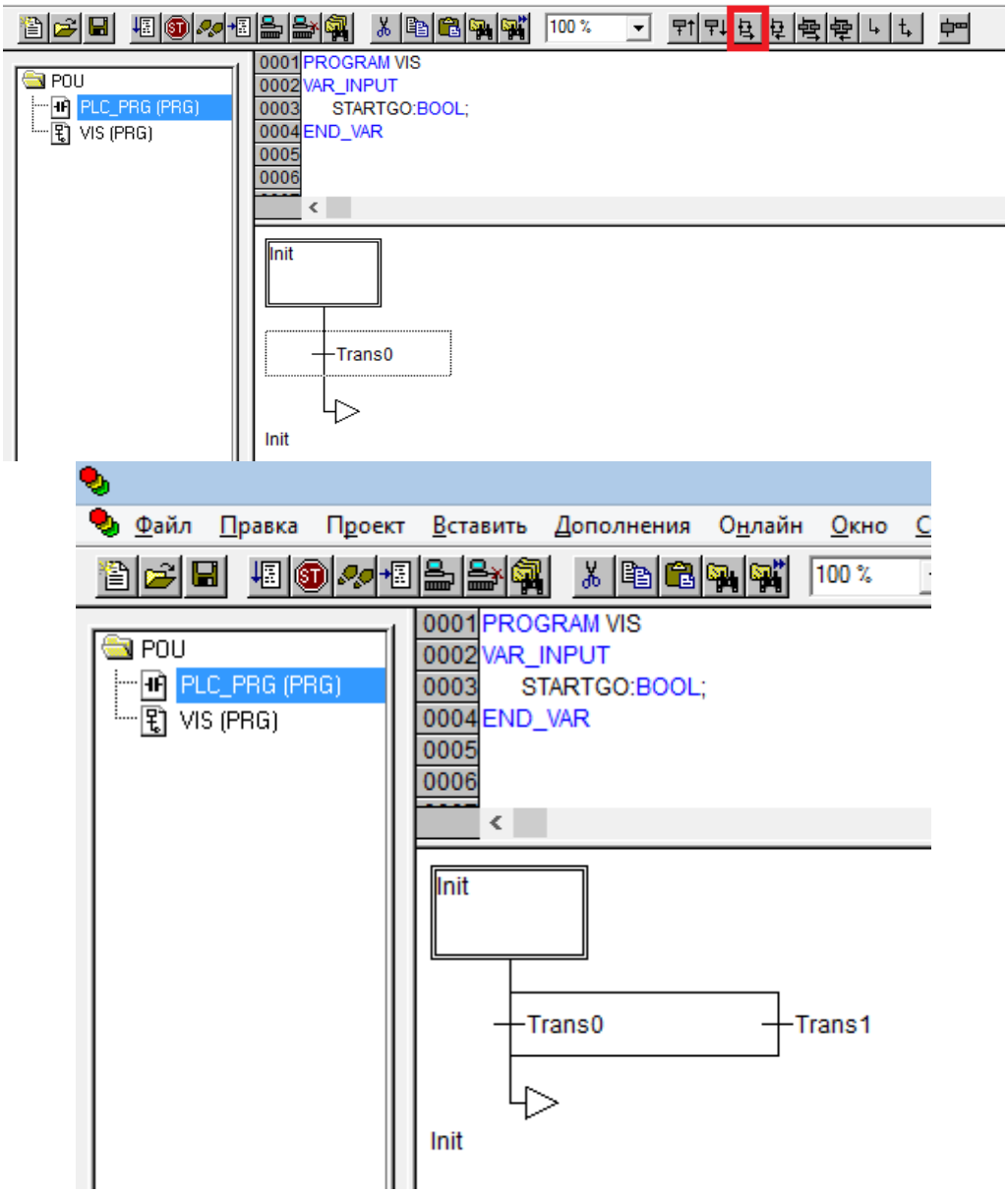


Рис. 3.20. Создание второй параллельной ветви в программе описания визуализации

Левая ветвь будет задавать вращающееся движение двигателя по часовой стрелке, а правая – против часовой стрелки.

7. С помощью «Шаг переход» на получившихся двух ветвях создайте по пять шагов (рис. 3.21).

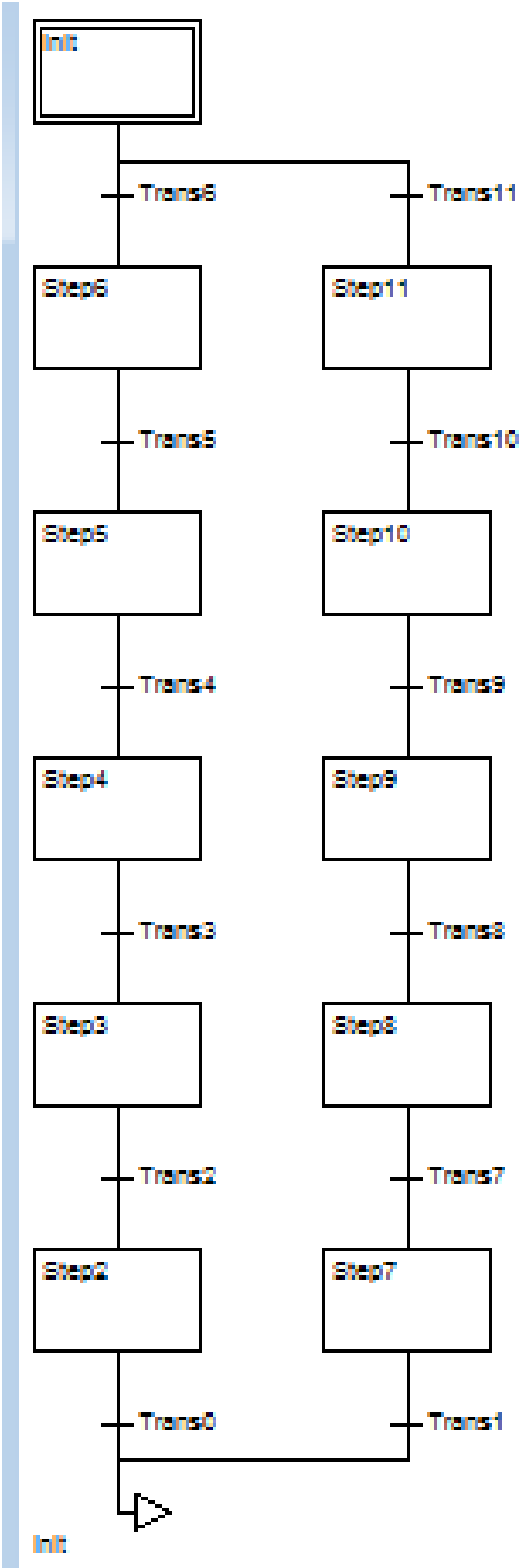


Рис. 3.21. Шаги программы визуализации

8. Переименуйте переходы *«Trans6»* и *«Trans11»* в *«M1»* и *«M2»* соответственно, чтобы активировать ветви при подаче питания на катушки M1 и M2 (рис. 3.22).

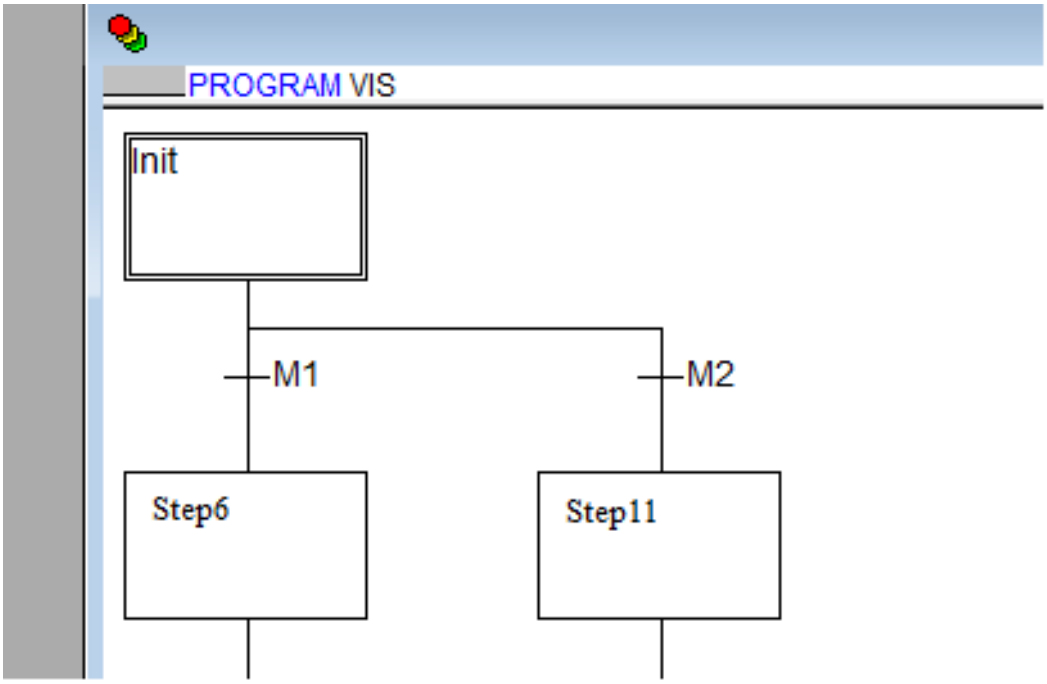


Рис. 3.22. Условие выбора направления вращения объекта визуализации

9. Переименуйте шаги, согласно выполняемого ими действия (рис. 3.23):

«Step6» - «RIGHTGO»

«Step5» - «DOWNGO»

«Step4» - «LEFTGO»

«Step3» - «UPGO»

«Step2» - Оставьте без изменений

«Step11» - «DOWNRE»

«Step10» - «RIGHTRE»

«Step9» - «UPRE»

«Step8» - «LEFTRE»

«Step7» - Оставьте без изменений

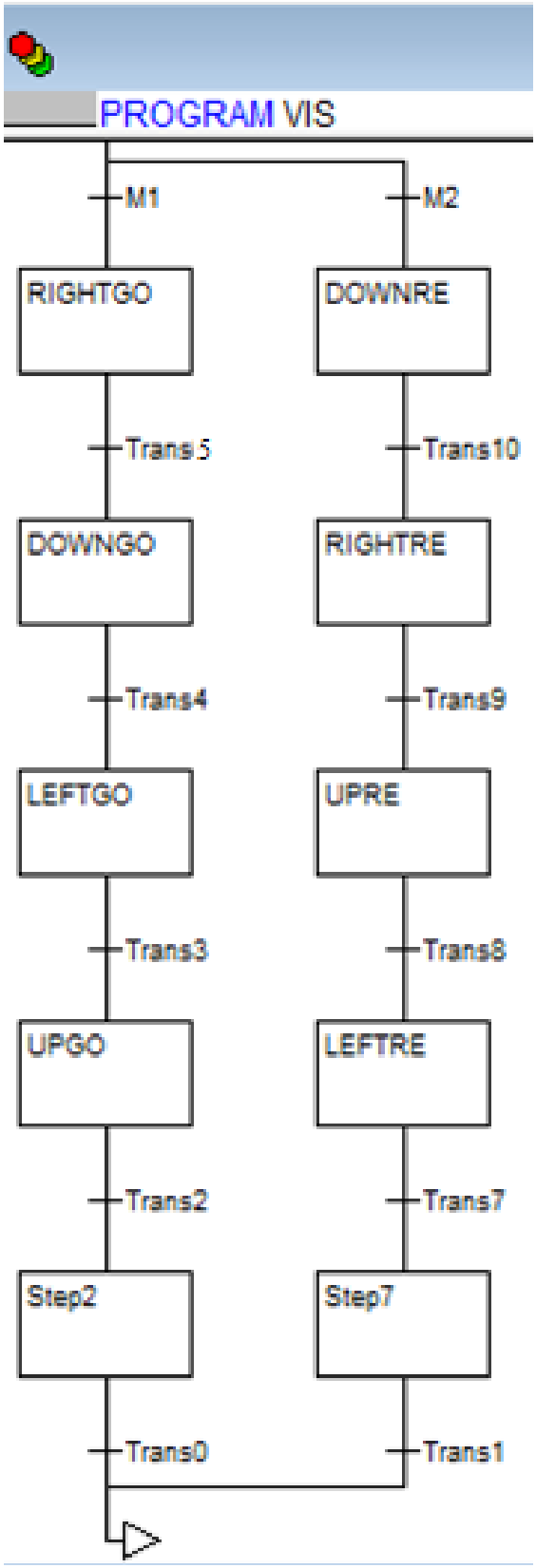


Рис. 3.23. Название шагов

10. Запрограммируйте шаги. Дважды щелкните по *«RIGHTGO»*. В отрывшемся окне выберите язык реализации *«ST»* и нажмите *«OK»* (рис. 3.24).

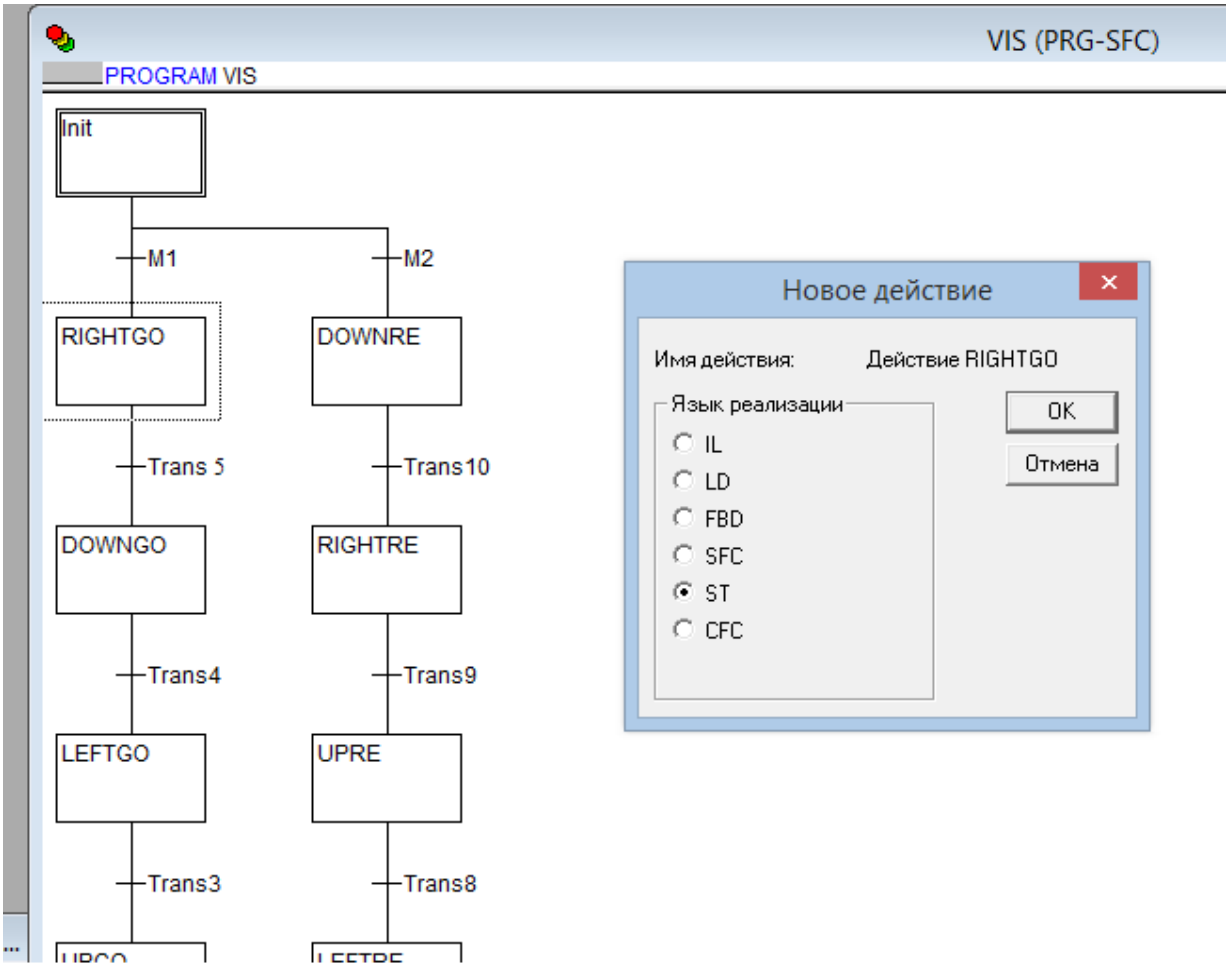


Рис. 3.24. Создание программ для выполнения объектом визуализации необходимых действий

11. В текстовом поле запрограммируйте шаг *«RIGHTGO»*. Введите строки:

```
IF M1=TRUE AND Y=0 THEN
X:=X+1;
ELSE RETURN;
END_IF
```

Здесь используется условие *IF* (если), т.е. если *M1* – верно, а *Y* = 0, то выполняется команда ***X:=X+1*** **(**увеличивается координата по *X*), тем самым выполняя движение вправо. Если хотя бы одно из условий не выполняется, то командой *RETURN* происходит выход из *«RIGHTGO»* (рис. 3.25).

Скорость изменения координат (вращения двигателя) может быть изменена путем задания большего, чем 1, числа. Например: ***X:=X+10.***

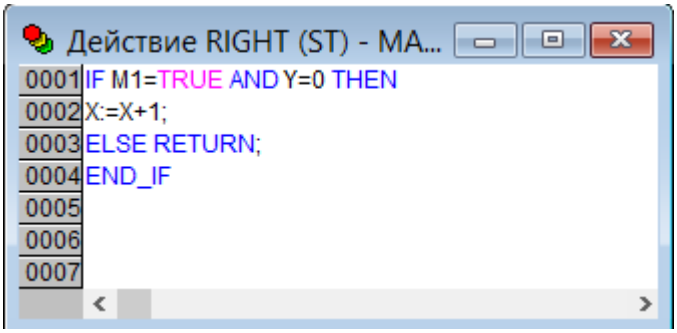


Рис. 3.25. Описание перемещения объекта при движении вправо по часовой стрелке

12. После этого дважды щелкните по *«Trans5»*, тем самым вы запрограммируете выход из цикла *«RIGHTGO»*. Здесь вы должны задать условия, при которых происходит переход на следующий шаг (рис. 3.26).

Дважды щелкнув по *«Trans5»,* в выпадающем окне выбирайте язык *«ST»* и щелкайте *«OK»*.

В текстовом поле необходимо ввести

X=100 OR Y<>0 OR M2=TRUE

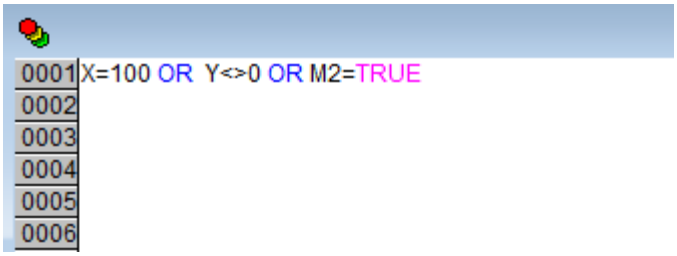


Рис. 3.26. Граничное условие при движении вправо

Здесь задаются три условия перехода на следующий шаг:

- достижение иксом значения 100;
- если *У* не равен 0;
- катушка *М2* активна.

13. По очереди программируется каждый шаг и условие перехода (рис. 3.27):

«DOWNGO» –
IF M1=TRUE AND X=100 THEN
Y:=Y+1;
ELSE RETURN;
END_IF

Переход 4 –
Y=100 OR M2=TRUE OR X<>100

«LEFTGO» –
IF M1=TRUE AND Y=100 THEN
X:=X-1;
ELSE RETURN;
END_IF

Переход 3 –
X=0 OR Y<>100 OR M2=TRUE

«UPGO» –
IF M1=TRUE AND X=0 THEN
Y:=Y-1;
ELSE RETURN;
END_IF

Переход 2 –
Y=0 OR X<>0 OR M2=TRUE

«Step2» – **НЕ ТРОГАЕМ!** Если тронули, то нажимайте по нему правой кнопкой мыши и выбирайте «Очистить действие/переход».

Переход 0 – Переименуйте *«Trans0»* в TRUE.

«DOWNRE» –
IF M2=TRUE AND X=0 THEN
Y:=Y+1;
ELSE RETURN;
END_IF

Переход 10 –
X<>0 OR Y=100 OR M1=TRUE

«RIGHTRE» –
IF M2=TRUE AND Y=100 THEN
X:=X+1;
ELSE RETURN;
END_IF

Переход9 –
X=100 OR Y<>100 OR M1=TRUE

«UPRE» –
IF M2=TRUE AND X=100 THEN
Y:=Y-1;
ELSE RETURN;
END_IF

Переход8 –
X<>100 OR Y=0 OR M1=TRUE

«LEFTRE» –
IF M2=TRUE AND Y=0 THEN
X:=X-1;
ELSE RETURN;
END_IF

Переход 7 –
X=0 OR Y<>0 OR M1=TRUE

«Step7» – **НЕ ТРОГАЕМ!** Если тронули, то нажимайте по нему правой кнопкой мыши и выбирайте «Очистить действие/переход».

Переход 1 – Переименуйте *«Trans1»* в *TRUE.*

Обратите внимание, когда программируется шаг или переход, то появляется черный треугольник. Надпись *«TRUE»* после написания принимает красный цвет.

Примечание:

Задать движение тела можно другим способом. Не изменяя координату по осям X и Y, имитируя движение по квадрату, а изменяя угол командой ***Ang:=Ang+1.0,*** *задавая траекторию движения по кругу.*

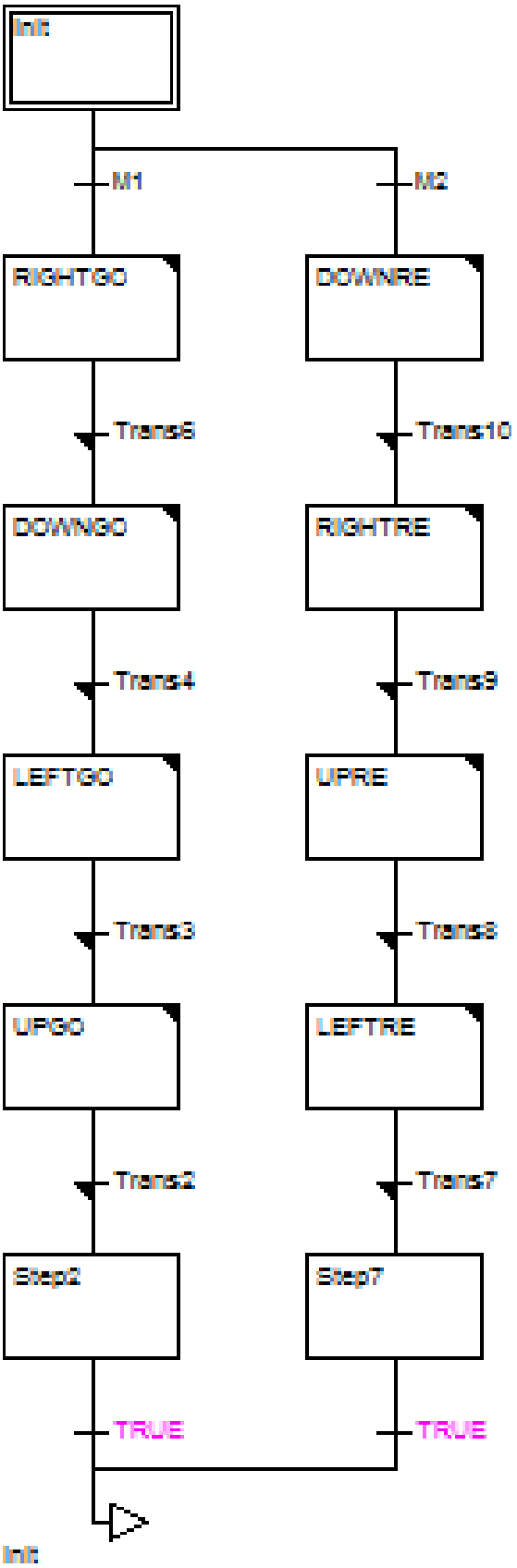


Рис. 3.27. Окончательный вид *POU* визуализации

3.3.3. *Создание визуализации*

1. Создайте окно с визуализацией. Выберите вкладку в левом нижнем углу «Визуализация» и в левой области экрана нажмите «Добавить новый объект». В появившемся окне введите имя. Например, *«GO»* (рис. 3.28).

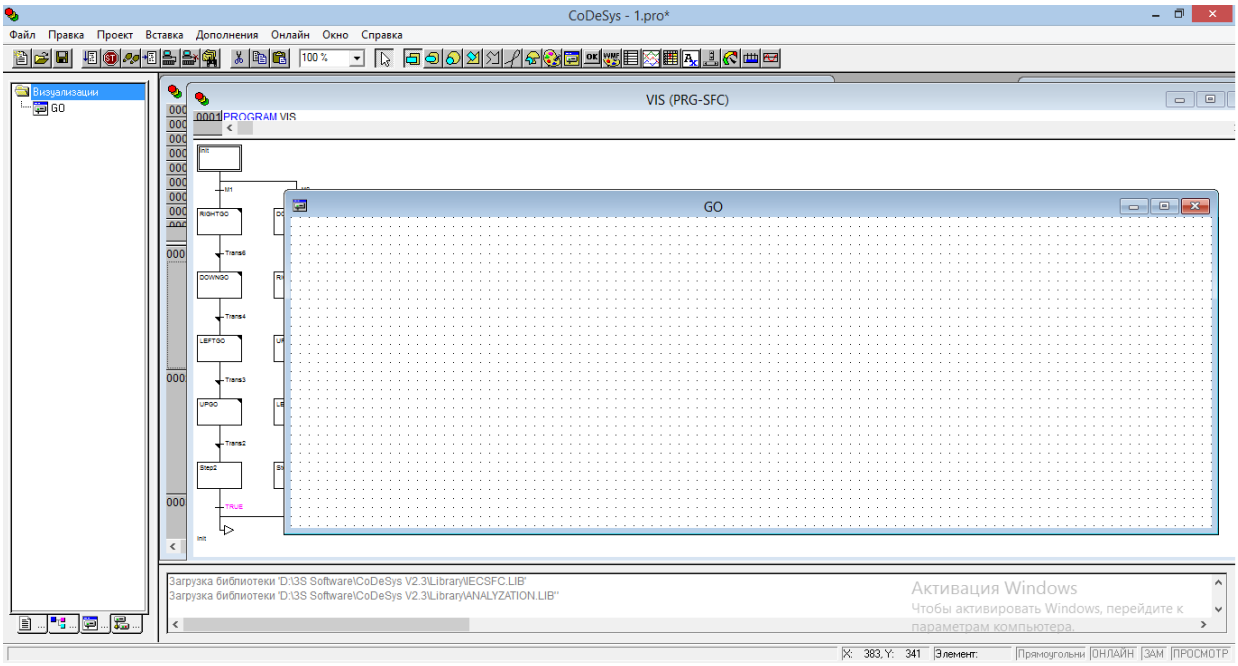


Рис. 3.28. Создание объекта визуализации

2. С помощью инструментов «Прямоугольник» и «Эллипс» создайте четыре кнопки и движимый объект (рис. 3.29).

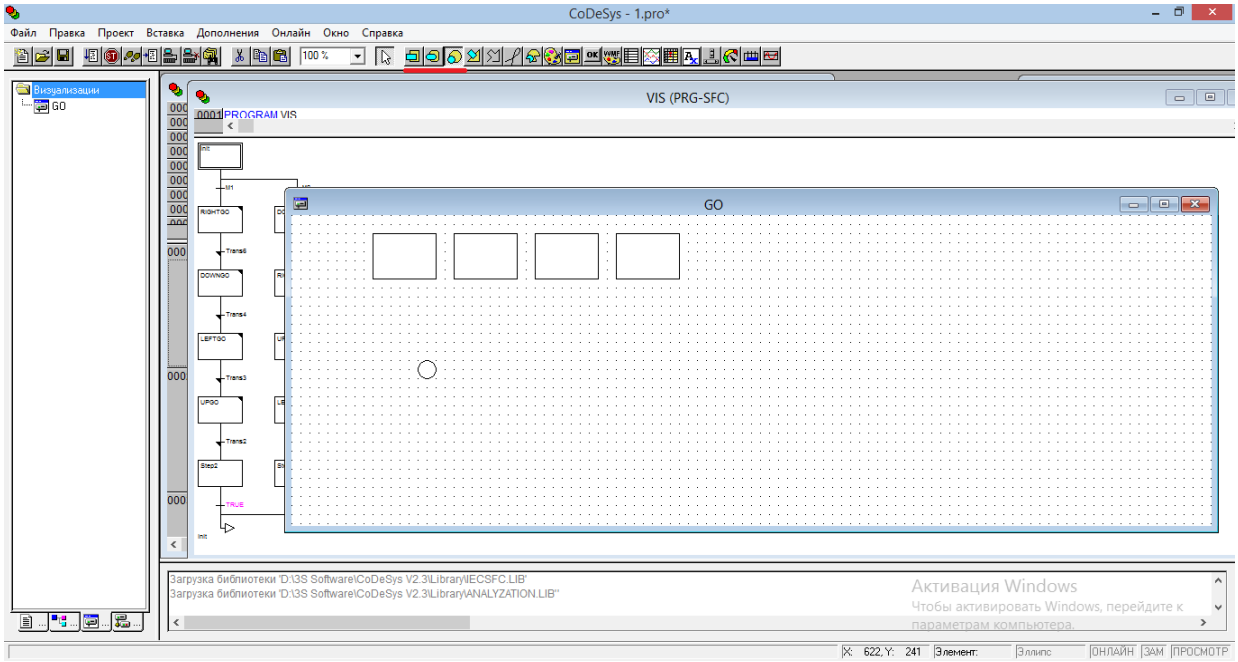


Рис. 3.29. Создание кнопок управления

3. Дважды щелкните по одному из прямоугольников. В появившемся окне выберите вкладку «Текст». В правой части в текстовом поле введите название кнопки (рис. 3.30).

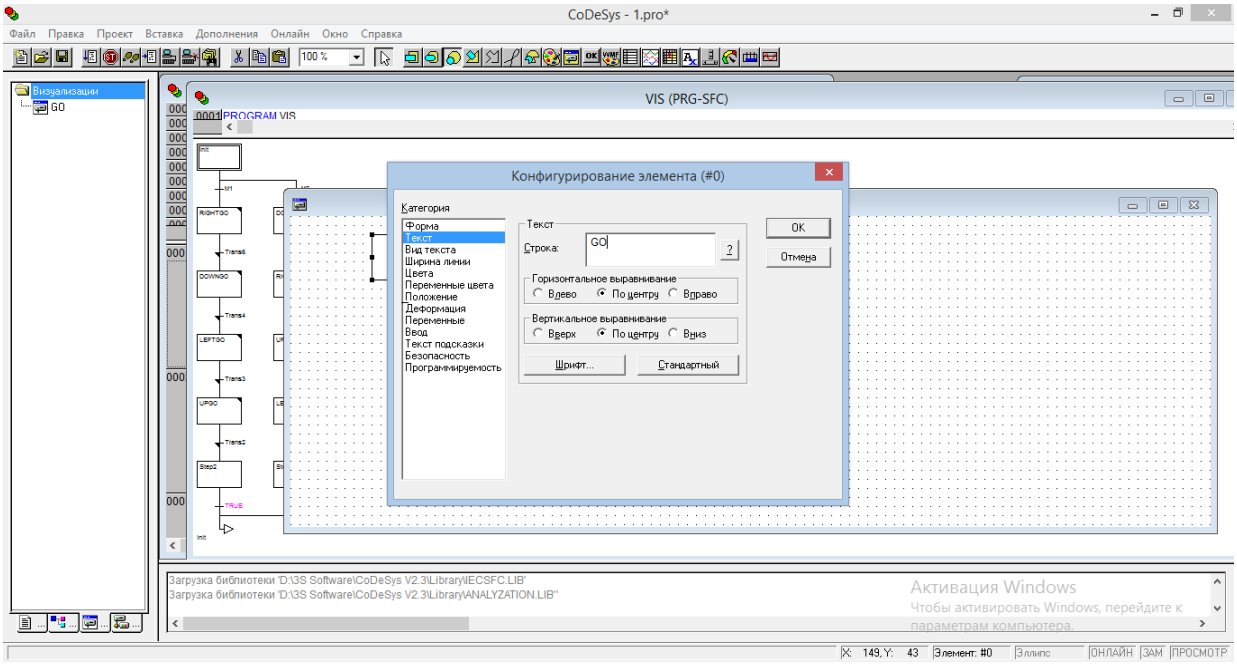


Рис. 3.30. Программирование кнопки *GO*

4. Снова сделайте двойной щелчок по этой кнопке. Выберите вкладку «Ввод». В правой части поставьте галку напротив «Переменная кнопка» и щелкните левой кнопкой по текстовому полю. Нажмите на клавиатуре кнопку «*F2*». В новом окне выберите переменную «*GO (BOOL)*» и нажимаем «ОК» (рис. 3.31, рис. 3.32).

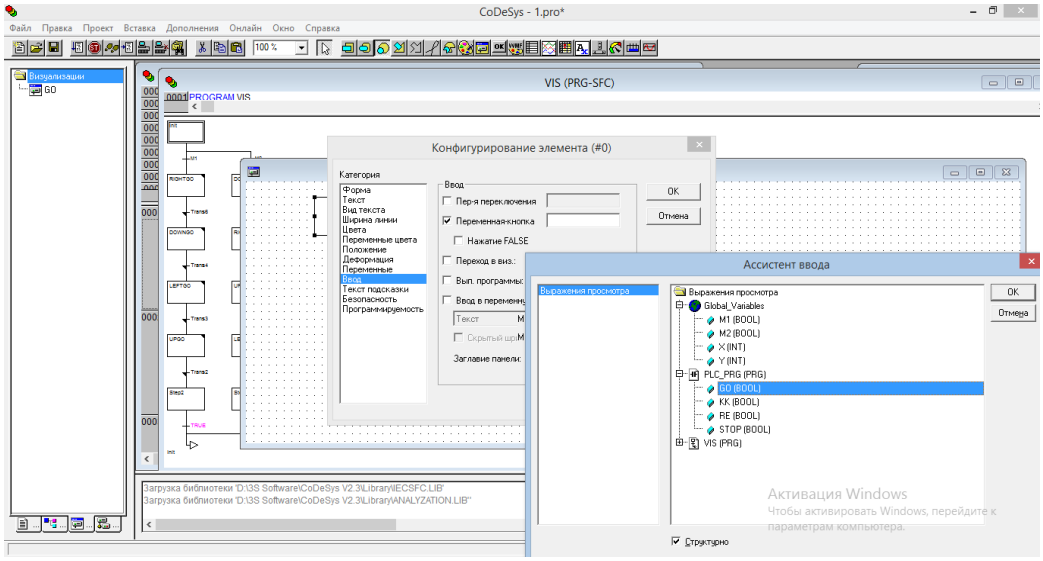


Рис. 3.31. Привязывание кнопки *GO* к соответствующему контакту (первое диалоговое окно)

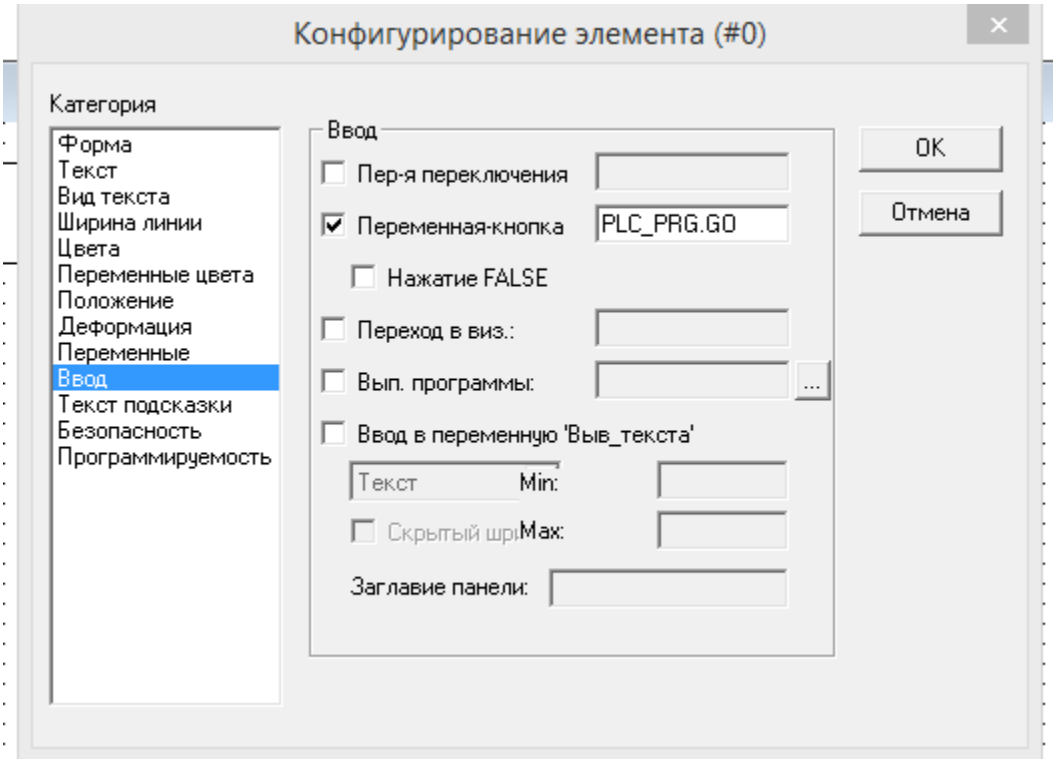


Рис. 3.32. Привязывание кнопки *GO* к соответствующему контакту (второе диалоговое окно)

5. Аналогично создайте кнопки «реверс» и «стоп», выбирая при этом переменные *«RE»* и *«Stop»* (рис. 3.33).

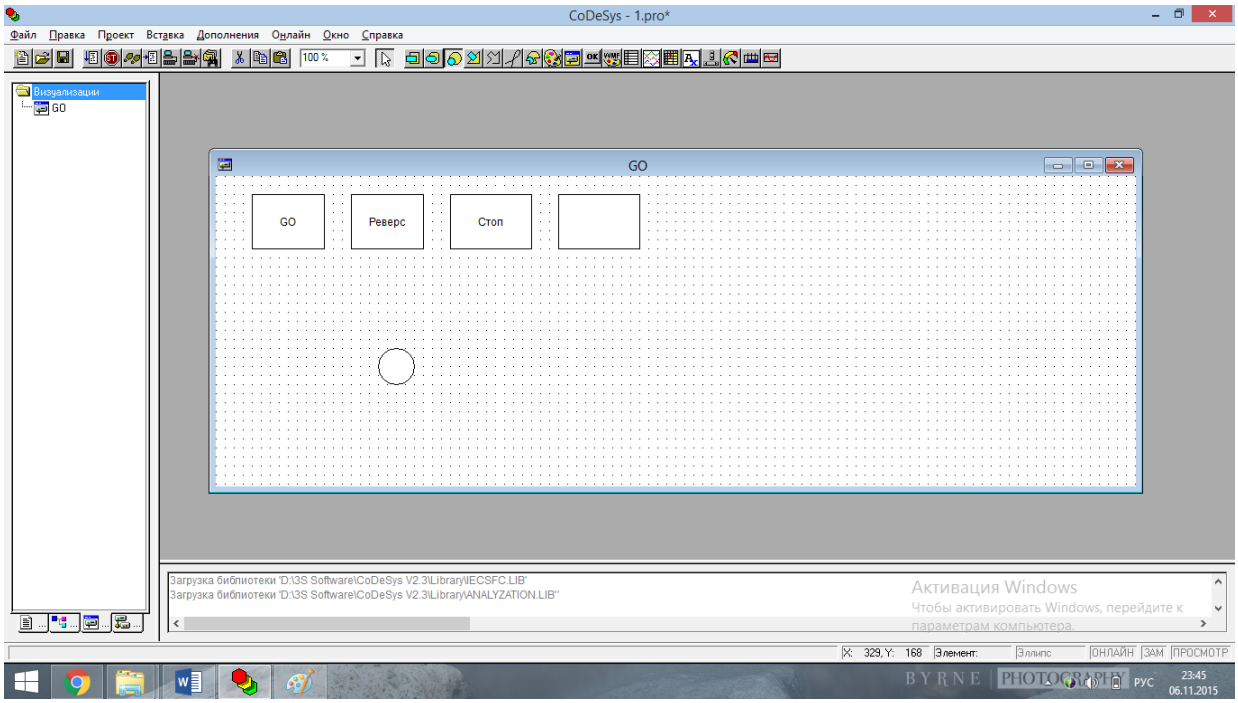


Рис. 3.33. Программирование кнопок управления

6. Создайте кнопку «*КК*», отвечающую за работу теплового реле. Создайте ее аналогично кнопкам «*GO*», «Реверс» и «Стоп», однако следует во вкладке «Ввод» выбрать не «Переменная кнопка», а

«Переменная переключения». Необходимо это для того, чтобы приблизить работу программы к работе теплового реле (рис. 3.34).

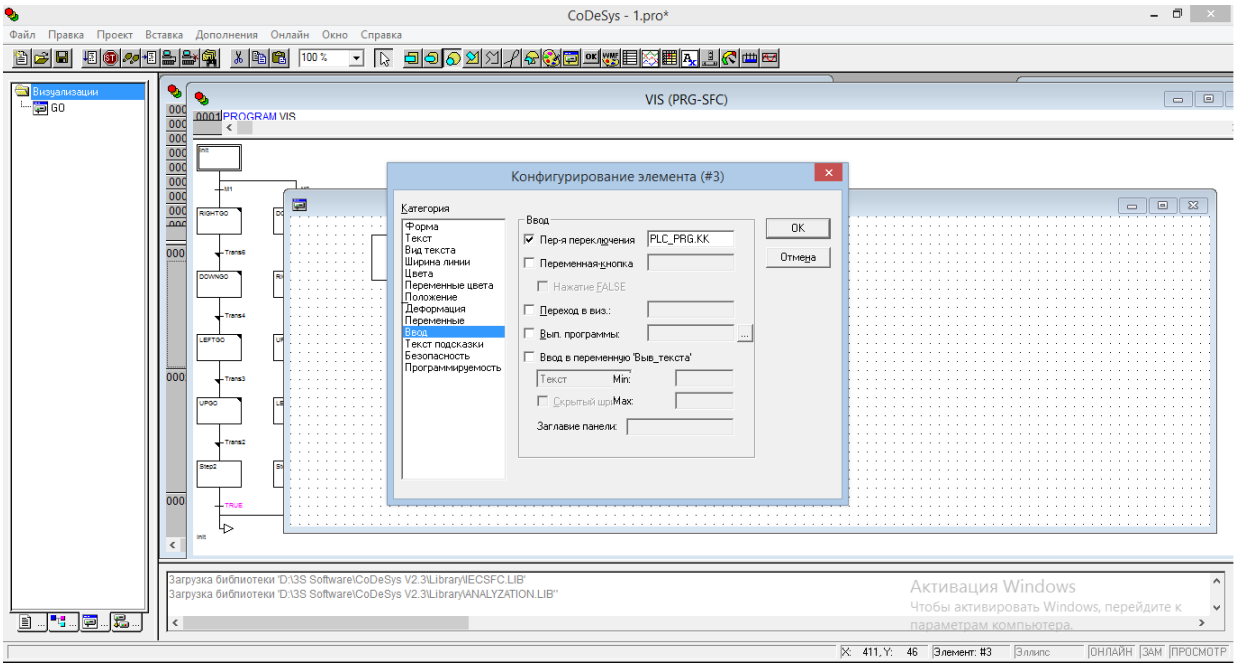


Рис. 3.34. Программирование теплового реле

7. Запрограммируйте вращающееся тело. Щелкните по нему дважды. Выберете вкладку положение. В позициях «*X*» и «*У*» с помощью «*F2*» выберите глобальные переменные «*X*» и «*У*» (рис. 3.35).

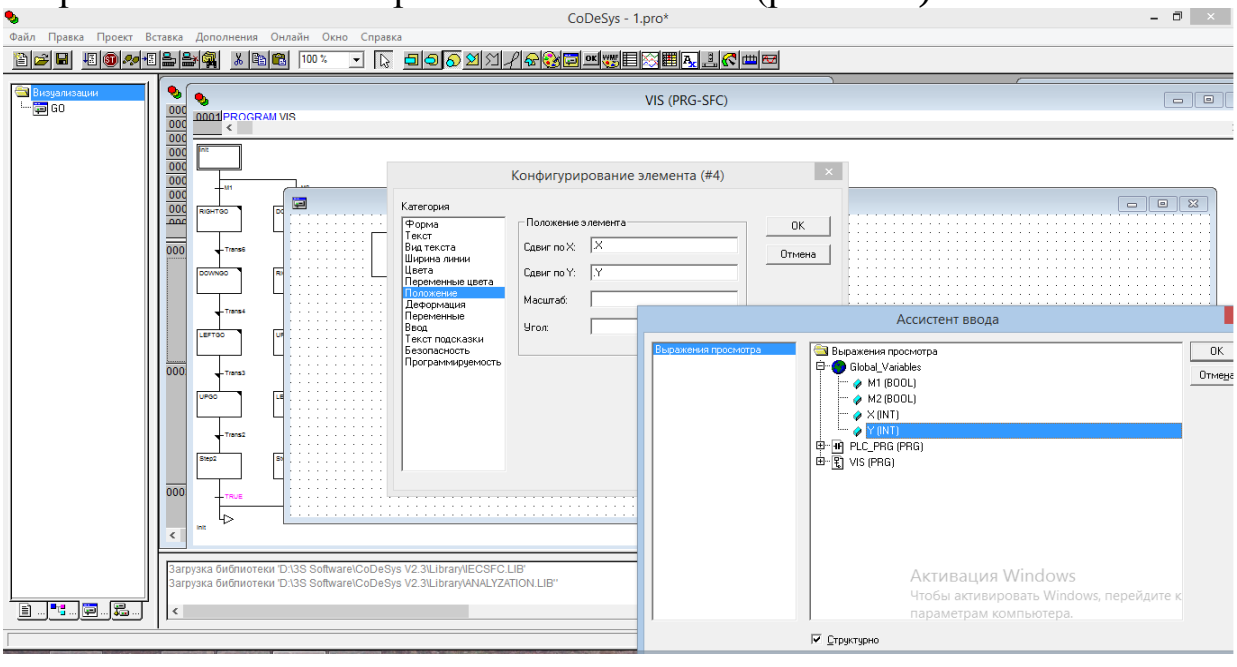


Рис. 3.35. Программирование движущегося объекта

8. Создайте визуализацию нажатия кнопок. Используйте при этом прямоугольники или эллипсы (рис. 3.36).

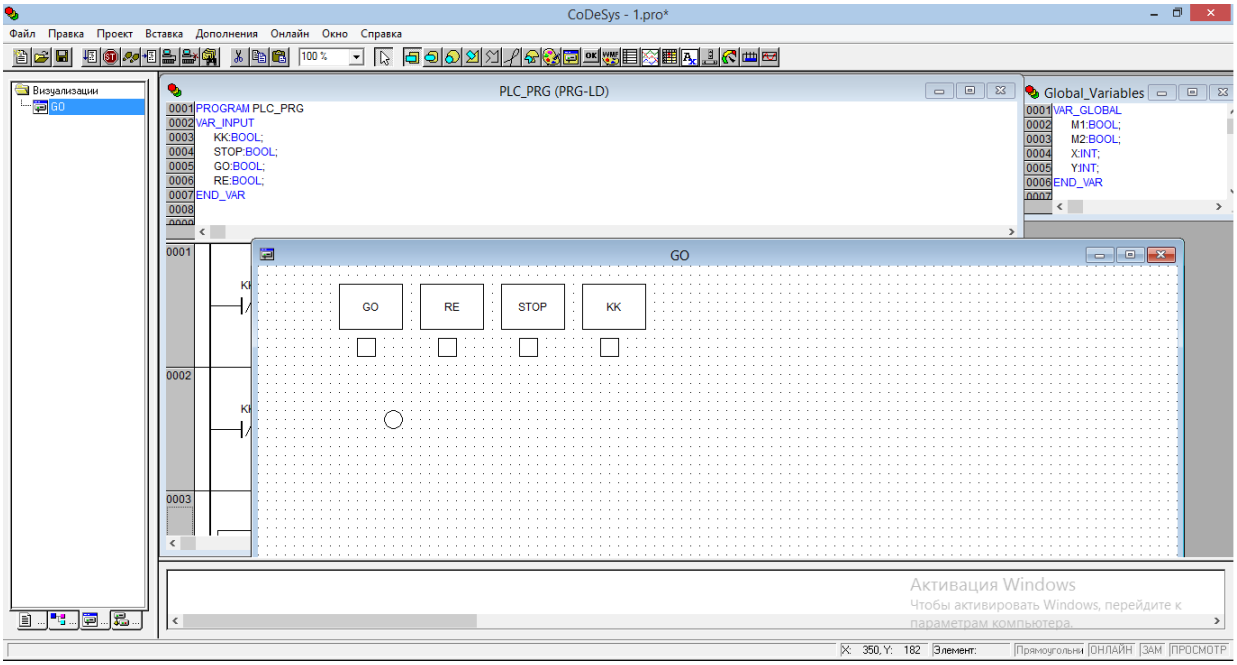


Рис. 3.36. Создание визуализации работы кнопок

Щелкнув дважды по прямоугольнику под кнопкой «*GO*», перейдите в категорию «Цвета». Выберите цвет заливки в нормальном режиме и в тревожном. Например, цвет нормального режима «зеленый», а тревожный – «красный» (рис. 3.37).

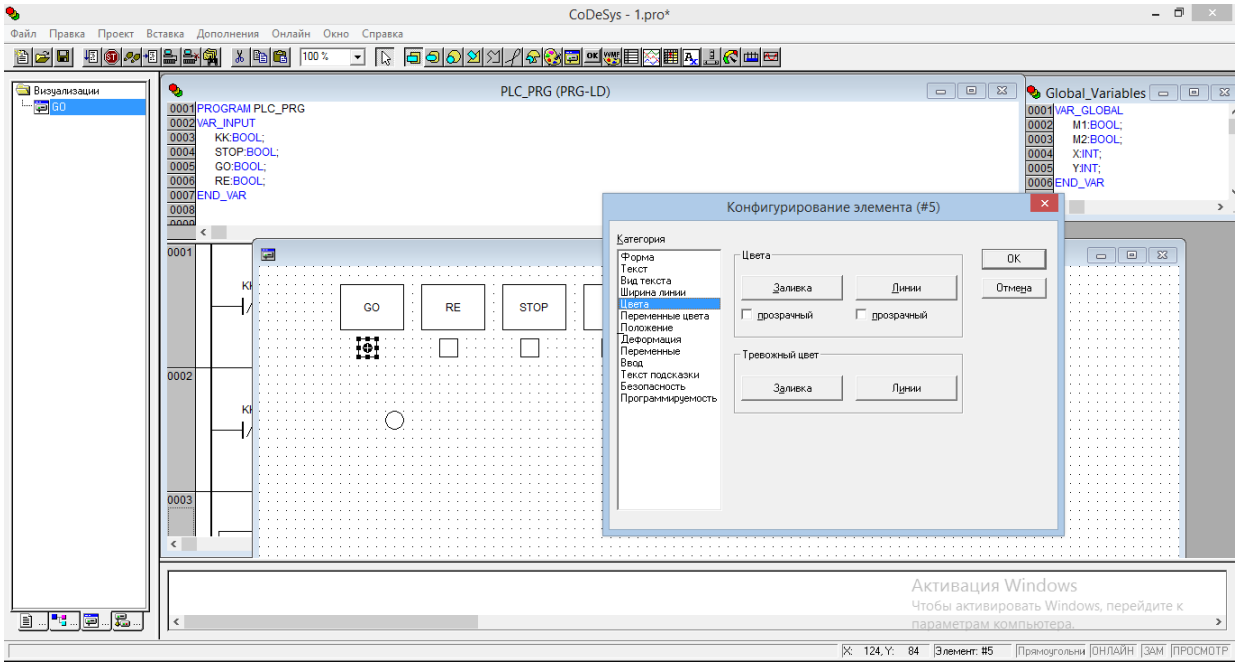


Рис. 3.37. Определение цветов нормального и активированного режима кнопок

После этого перейдите в категорию «Переменные». Выберите в параметре «Изм. Цвета» кнопкой «*F2*» переменную «*M1*» (рис. 3.38).

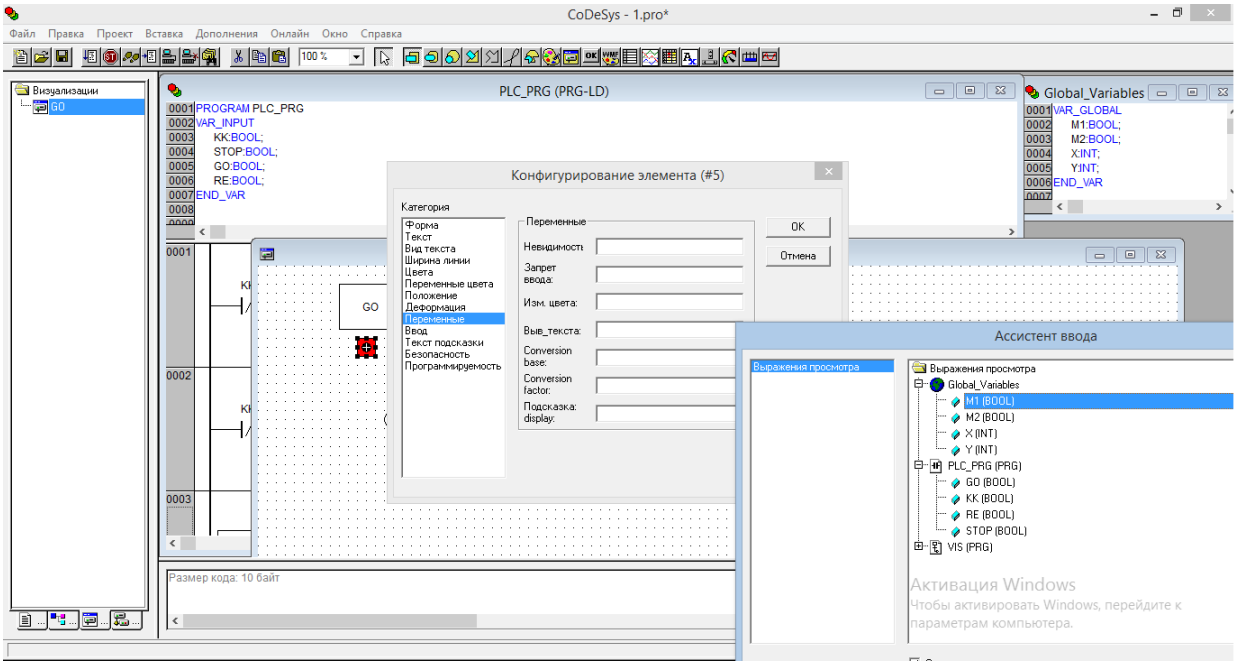


Рис. 3.38. Программирование изменения цвета кнопок в зависимости от состояния катушек

3.3.4. *Запуск программы*

1. В верхней центральной части экрана нажмите «Онлайн» и поставьте галку над «Режим эмуляции».

2. Затем в «Онлайн» выберите «Подключить».

3. В «Онлайн» выберите «Старт» или просто нажмите на *F5*.

4. В режиме эмуляции можно нажимать на кнопки и наблюдать за работой программы.

4. *Практическая работа № 2*
РЕАЛИЗАЦИЯ РАБОТЫ АВР НА КОНТРОЛЛЕРЕ *PLC 150.I-M (CODESYS)*

4.1. Краткие теоретические сведения

Основные требования к схемам автоматического ввода резерва (АВР) секционного выключателя (СВ) (ПУЭ, п. 3.3.30 – 3.3.42) [7]:

1. Схема АВР должна приходить в действие при исчезновении напряжения на шинах потребления по любой причине (аварийное, ошибочное или самопроизвольное отключение рабочего источника);
2. До отключения рабочего источника резервный источник не должен включаться во избежание его включения на КЗ;
3. Для уменьшения длительности перерыва питания потребителей включение резервного источника питания должно производиться сразу после отключения рабочего источника;
4. Действие АВР должно быть однократным, чтобы не допускать нескольких включений резервного источника на неустранившееся КЗ.
5. Для действия АВР при исчезновении питающего напряжения, не сопровождающегося отключением выключателя рабочего источника, схема АВР должна дополняться пусковым органом по напряжению и (или) по частоте.
6. Для ускорения отключения резервного источника при его включении на неустранившееся КЗ должно предусматриваться ускорение защиты резервного источника после АВР.
7. Схема АВР не должна действовать при одновременном исчезновении напряжения на обоих источниках питания.
8. Действие АВР должно быть согласовано с действием РЗ и другой автоматики правильным выбором уставок АВР.
9. Схема АВР должна предусматривать блокировку действия.

В схеме на ПС два вводных выключателя *Q1* и *Q2* и секционный *Q*. На каждой секции трансформаторы напряжения *TV1* и *TV2*, которые питают реле минимального напряжения. При уменьшении напряжения ниже заданного уровня они срабатывают. *KV1, KV2, KV4, KV5* - отслеживают напряжение на рабочем источнике, *KV3, KV6* - напряжение на резервном источнике.

На рис. 4.1 изображена схема АВР двухстороннего действия.

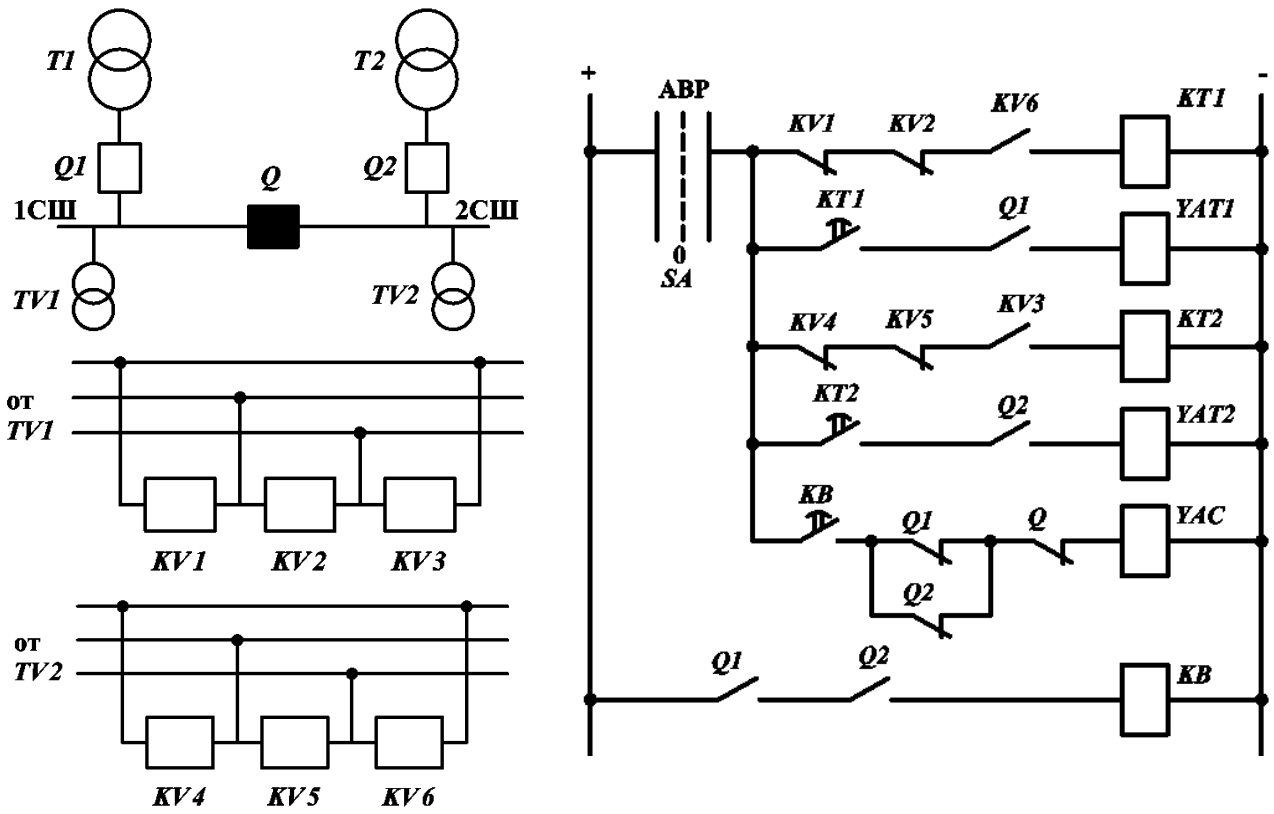


Рис. 4.1. Принципиальная схема устройств АВР на секционном выключателе:

KV1, KV2, KV4, KV5 - реле минимального напряжения рабочего источника; *KV3, KV6* - реле минимального напряжения резервного источника; *Q1, Q2, Q* - вспомогательный контакт выключателя; *KT1, KT2* – реле времени; *YAT1, YAT2* - катушки отключения выключателей *Q1* и *Q2; YAC* - катушка включения выключателя *Q; KB* - реле блокировочное (реле однократности включения); *SA* - ключ управления (переключатель)

Отключение одного из выключателей

При отключении одного из выключателей *Q1* или *Q2* размыкается контакт *Q1* (или *Q2*), блокировочное реле *KB* теряет питание. Его замыкающий контакт с выдержкой времени на размыкание размыкается, но сигнал успевает пройти по цепи, и катушка включения секционного выключателя *YAC* получает питание. Секционный выключатель *Q* включается. Из-за выдержки времени сигнал успевает пройти, но замыкающий контакт «отваливается», и цепь остается без питания. Тем самым обеспечивается однократность включения *Q* (срабатывания АВР).

Условие срабатывания при исчезновении напряжения на шинах потребителей

При исчезновении напряжения на 1 с.ш. реле *KV1, KV2* теряют питание. Размыкающие контакты замыкаются и подают питание на реле времени *KT1*. Замыкается контакт с замедлением на срабатывание и катушка отключения выключателя *Q1* - *YAT1* получает питание. Таким

образом выключатель *Q1* отключается, и далее все происходит по сценарию, описанному выше.

4.2. Задание

1. Создать коммутационную программу для принципиальной электрической схемы работы АВР двухстороннего действия.
2. Создать визуализацию работы АВР.

Например:

Кнопки, имитирующие срабатывание реле минимального напряжения *KV1*, *KV2*, *KV3*, *KV4*, *KV5*, *KV6*. Кнопки отключения выключателей *Q1*, *Q2*, *Q*.

Пример оформления визуализации см. рис. 4.2.

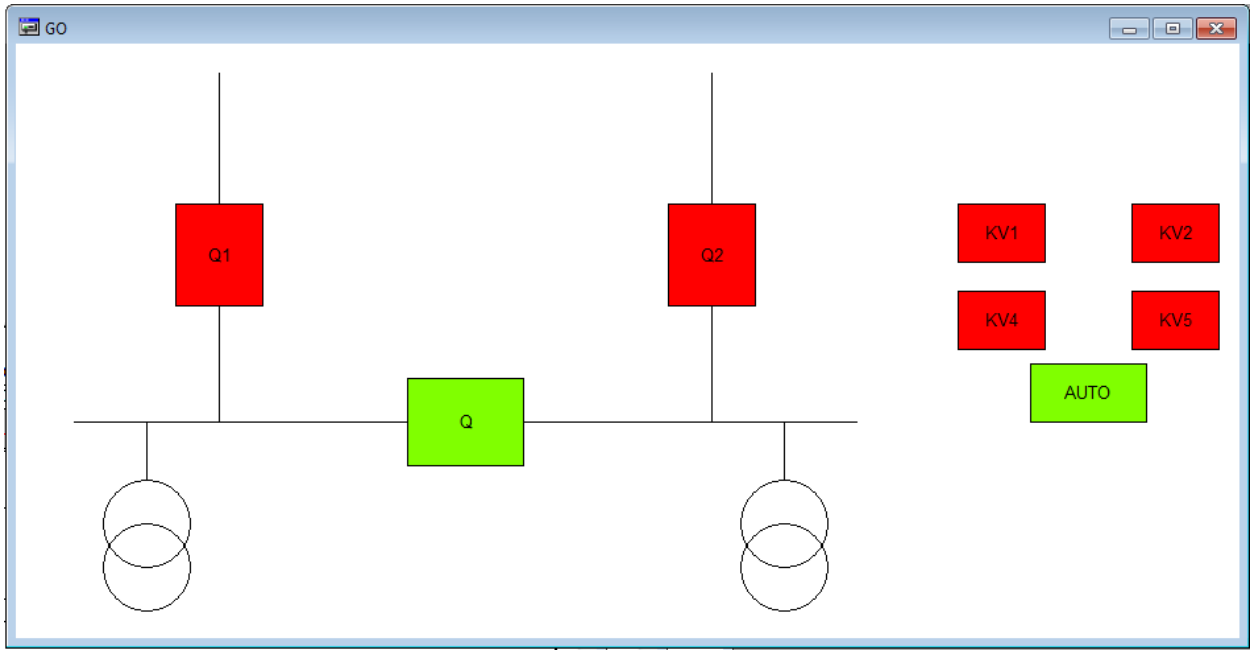


Рис. 4.2. Пример интерфейса (визуализации)

3. Подготовить отчет по плану:

- цель работы;
- требования к схемам автоматики включения резерва СВ;
- принцип действия АВР;
- подробное пошаговое описание коммутационной программы (контактов катушек реле, таймеров);
- описание визуализации. Блок схема. Алгоритм построения визуализации;
- выводы. Преимущества реализации АВР на контроллерах по сравнению с АВР на релейной логике и т.п.

4.3. Выполнение работы в программе *CodeSys*

1. Создайте новый проект в программе *CoDeSys*, выбрав при этом одну из конфигураций ПЛК, например, «*PLC150.I-M*». Не меняя ничего в настройках целевой платформы, нажмите «*Ок*» (рис. 4.3, рис. 4.4).

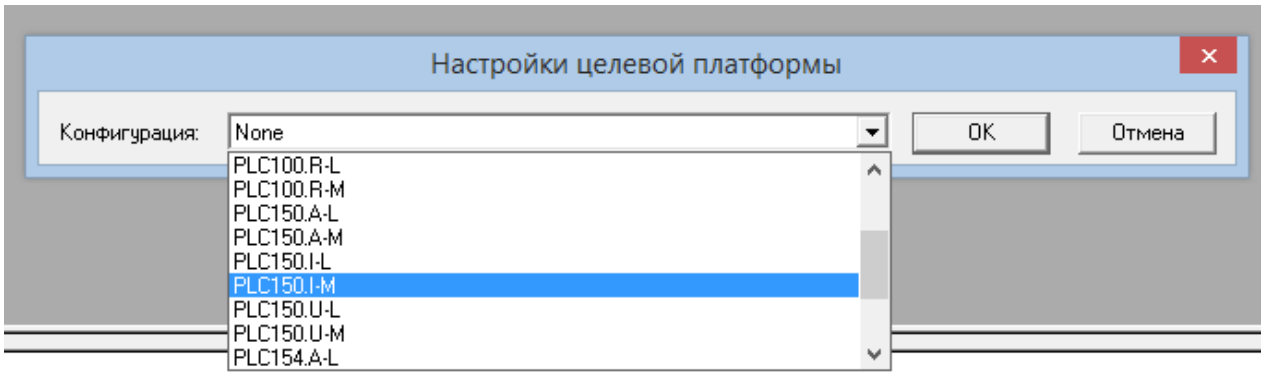


Рис. 4.3. Выбор конфигурации ПЛК

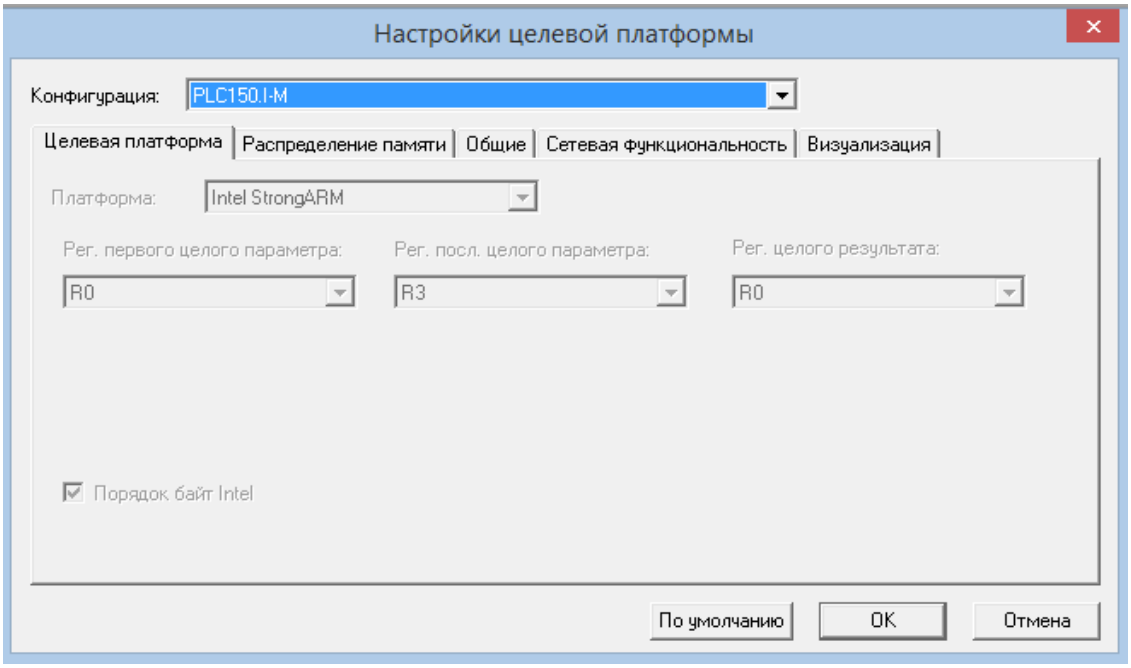


Рис. 4.4. Настройка целевой платформы ПЛК

2. Введите в работу первый программный компонент *POU* типа «Программа» на языке реализации «*LD*», при этом не изменяйте «Имя нового *POU*» (рис. 4.5).

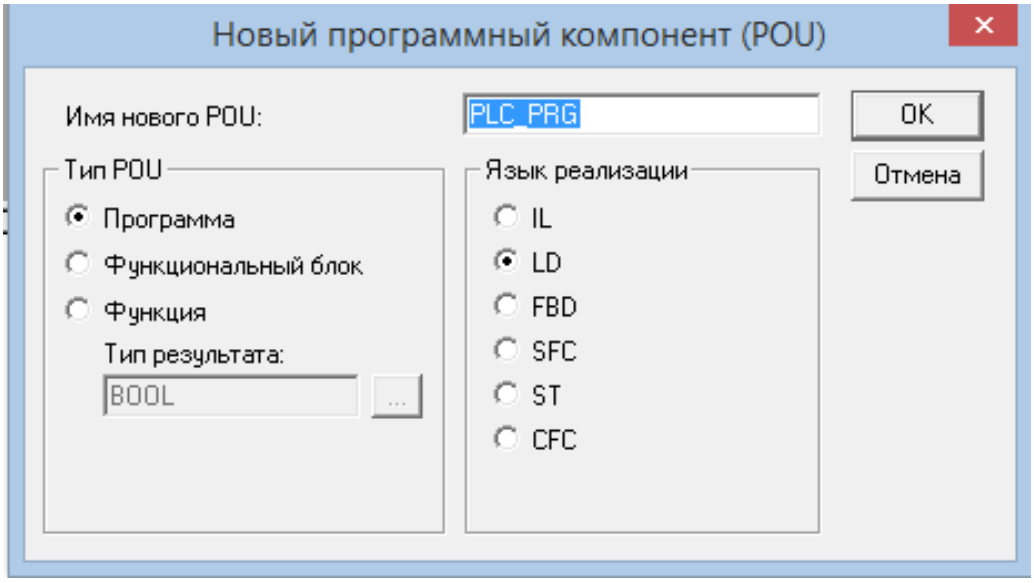


Рис. 4.5. Создание нового программного компонента *POU*

3. Создайте на лестничной диаграмме элементы, необходимые для работы программы (рис. 4.6). Используйте такие элементы, как:

- контакт;
- инверсный контакт;
- параллельный контакт;
- параллельный инверсный контакт;
- катушка;
- таймер (находится в той же строке, но ближе к концу).

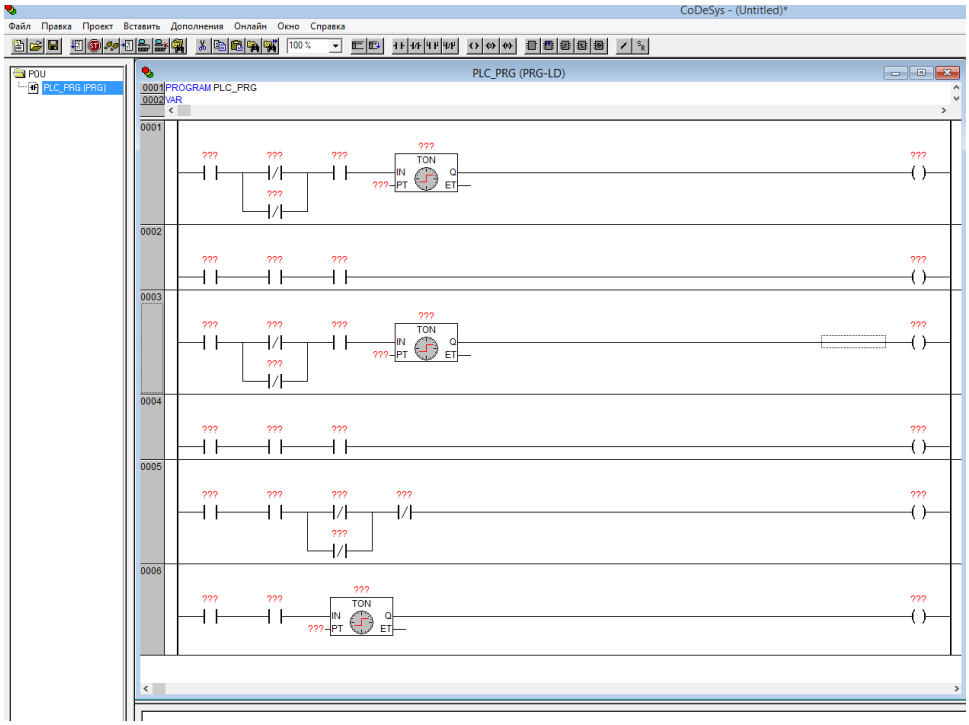


Рис. 4.6. Нанесение элементов электрической схемы в программе на языке *LD*

4. Опишите используемые переменные в программе *PLC_PRG* (рис. 4.7).

```
PROGRAMPLC_PRG
VAR
        KV1: BOOL:=TRUE;
        KV2: BOOL:=TRUE;
        KV3: BOOL:=TRUE;
        KV4: BOOL:=TRUE;
        KV5: BOOL:=TRUE;
        KV6: BOOL:=TRUE;
        T1: BOOL;
        T2: BOOL;
        KB: BOOL;
        TOF1: TOF;
        TON1: TON;
        TON2: TON;
END_VAR
```

Опишите глобальные переменные в *Global_Variables*.

```
VAR_GLOBAL
        Q1: BOOL:=TRUE;
        Q2: BOOL:=TRUE;
        Q:BOOL;
        YAT1: BOOL;
        YAT2: BOOL;
        YAC:BOOL;
        AUTO: BOOL;
END_VAR
```

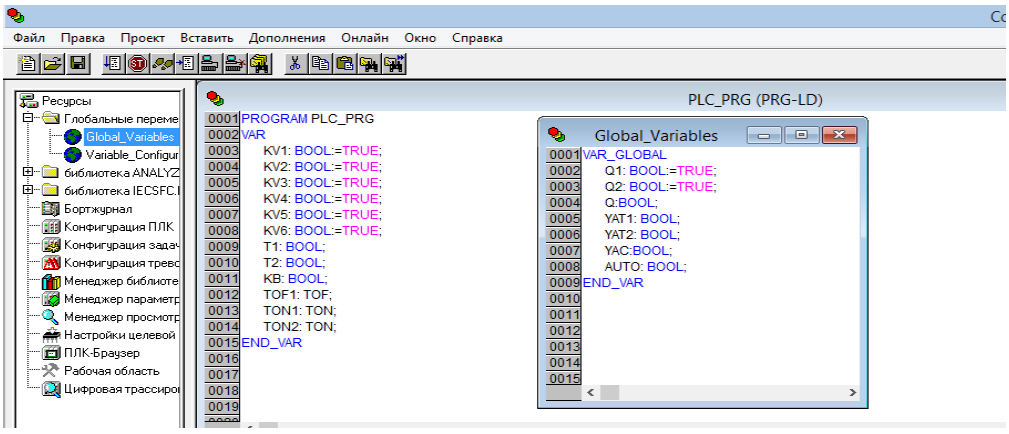


Рис. 4.7. Описание переменных

5. Нанесите названия на соответствующие элементы схемы (рис. 4.8).

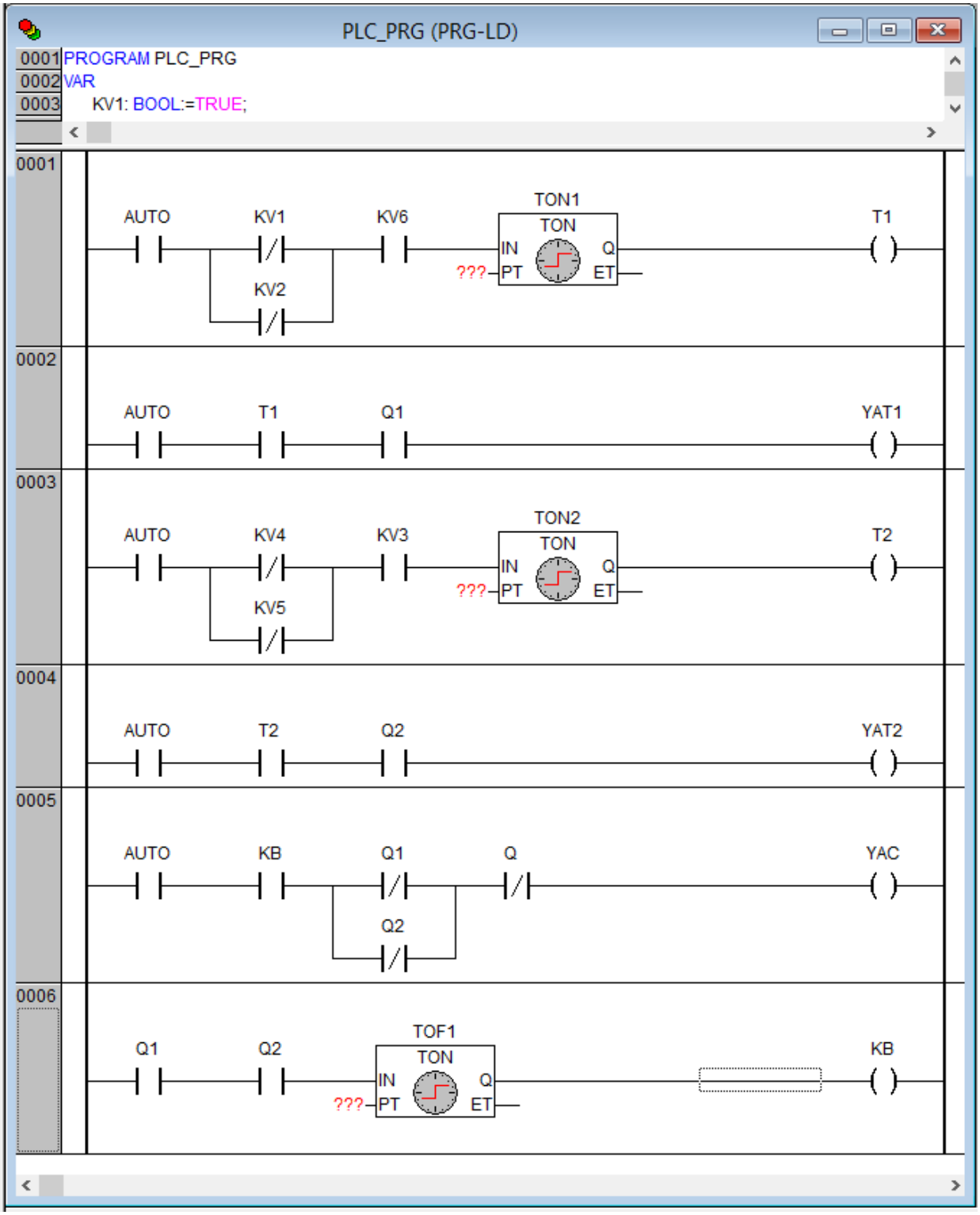


Рис. 4.8. Электрическая схема на языке *LD*

6. Установите значение таймеров *TON1*, *TON2* и *TOF1* (рис. 4.9) Для этого нажмите на вход таймера *PT* по знакам вопроса и введите там следующий текст:

T#1s

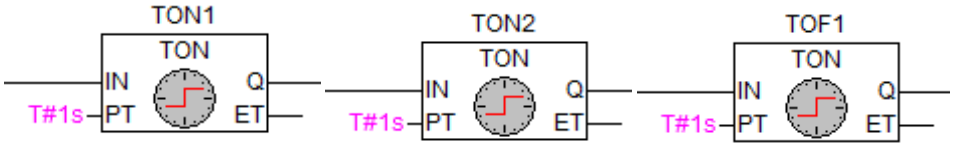


Рис. 4.9. Настройка таймеров

7. В таймере *TOF1* нажмите на надпись «*TON*» и измените ее на «*TOF*» (рис. 4.10).

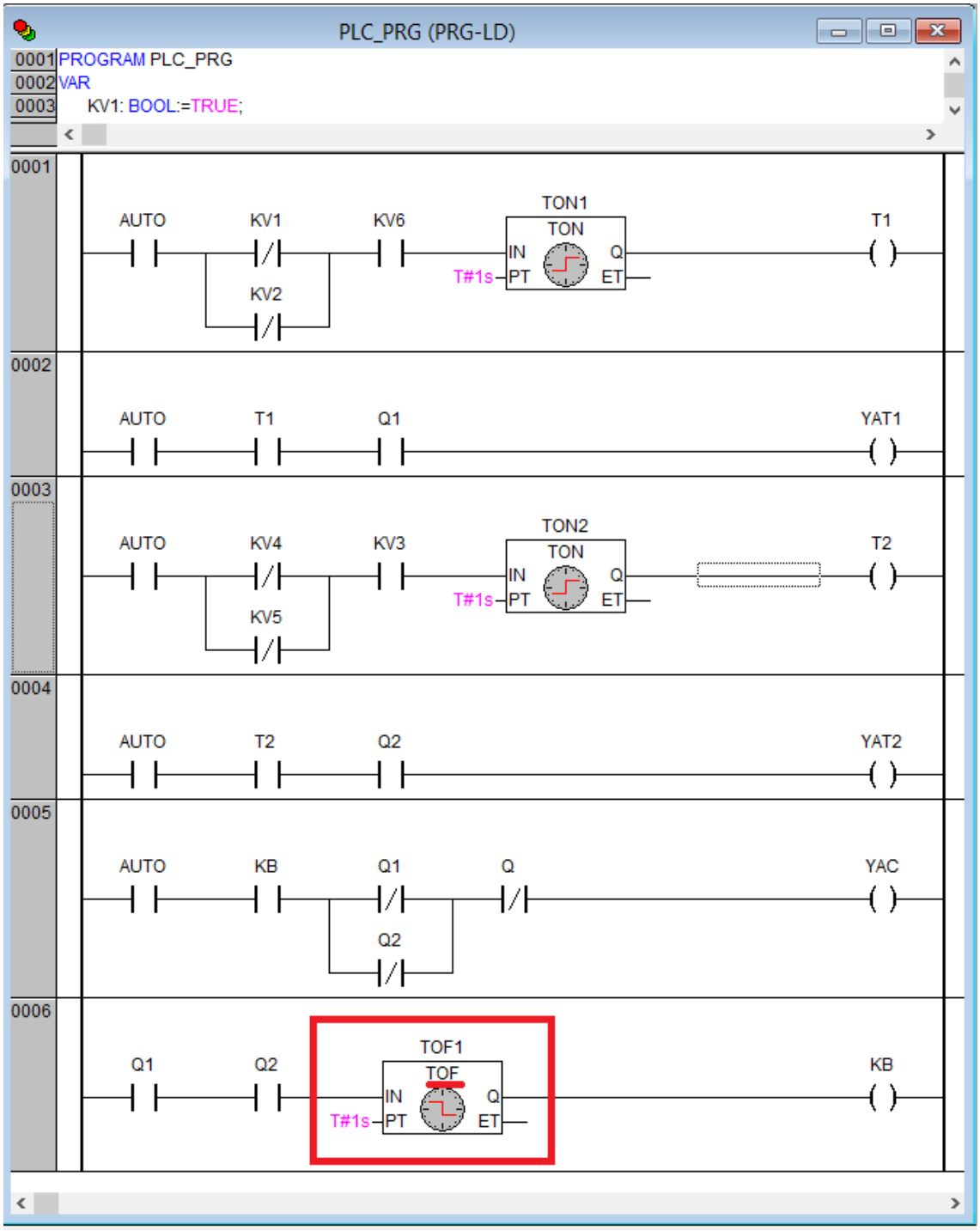


Рис. 4.10. Создание таймера, срабатывающего при отключении

8. Необходимо создать вторую программу с произвольным именем на языке *SFC* для программирования логики работы выключателей. Для этого нажмите правой кнопкой мыши в папке *POU* и нажмите «Добавить объект» (рис. 4.11).

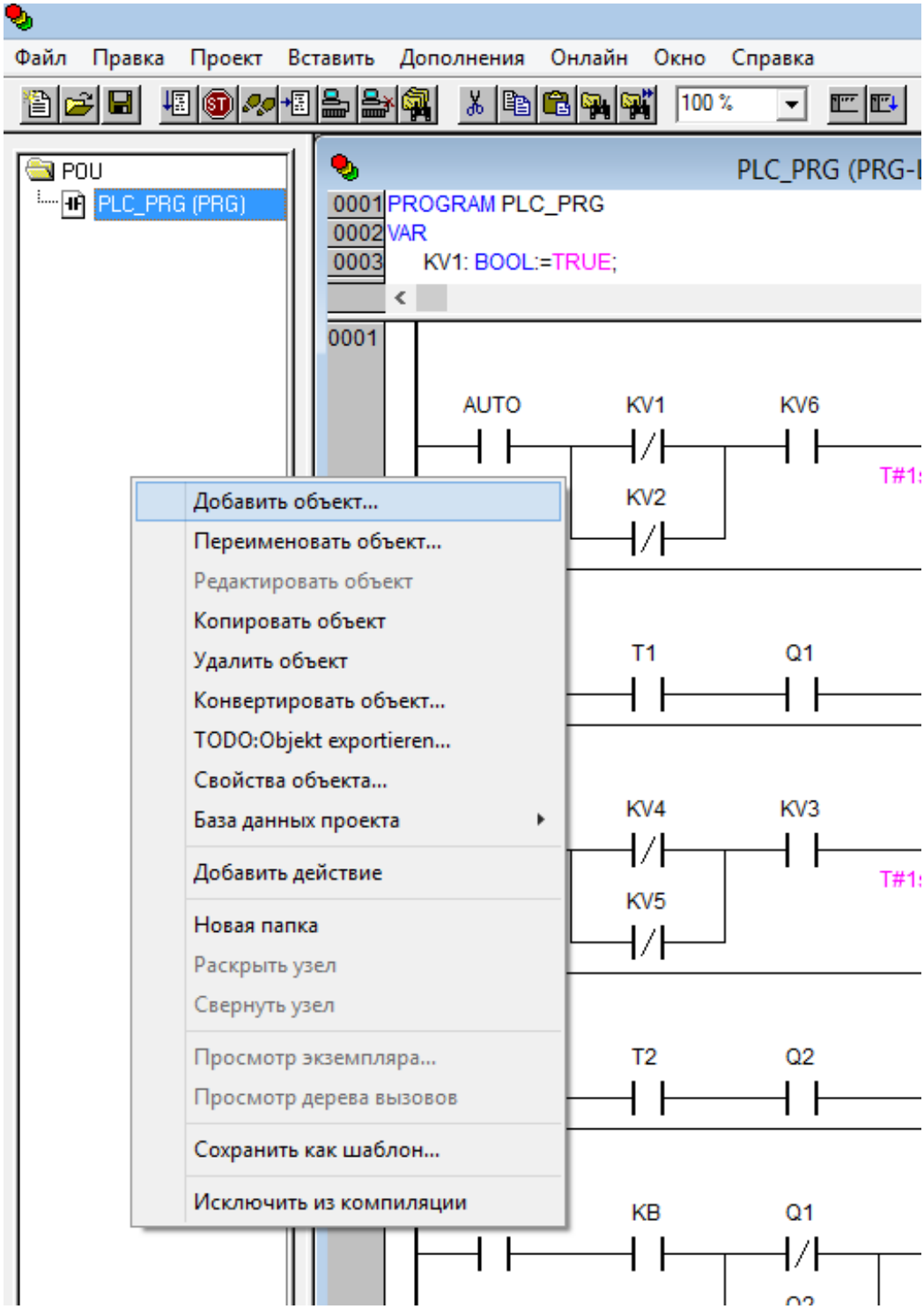


Рис. 4.11. Создание *POU* на языке *SFC*

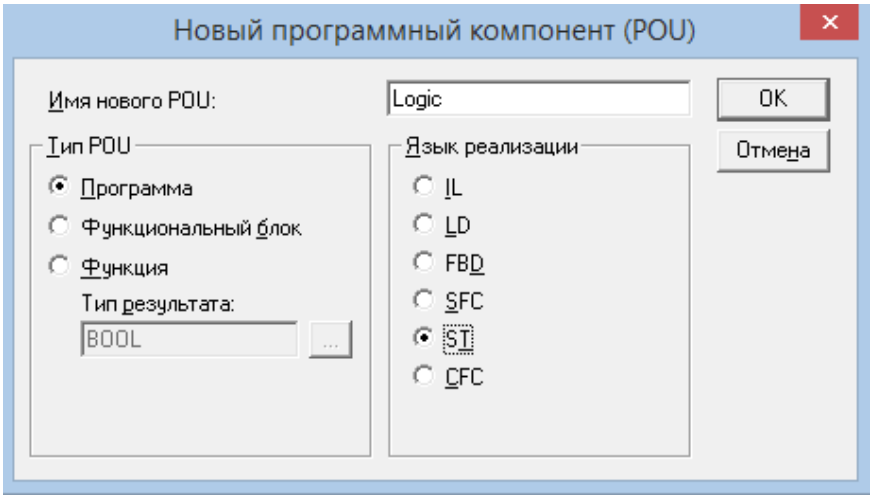


Рис. 4.12. Новый программный компонент *POU* на языке *SFC*

10. В созданной программе опишите входную переменную, которая будет отвечать за работу этой подпрограммы (рис. 4.13).

```
PROGRAM Logic
VAR_INPUT
    POWER:BOOL;
END_VAR
```

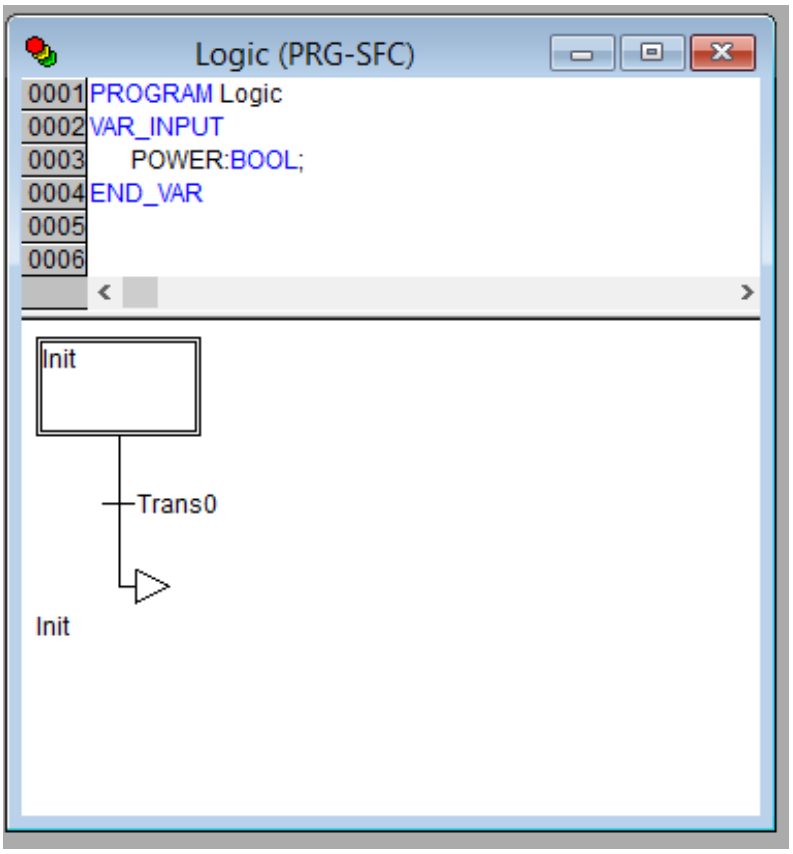


Рис. 4.13. Описание входной переменной

11. Нажав на «*Trans0*», создайте «Альтернативную ветвь (слева)» несколько раз так, чтобы получилось четыре параллельные ветви (рис. 4.14).

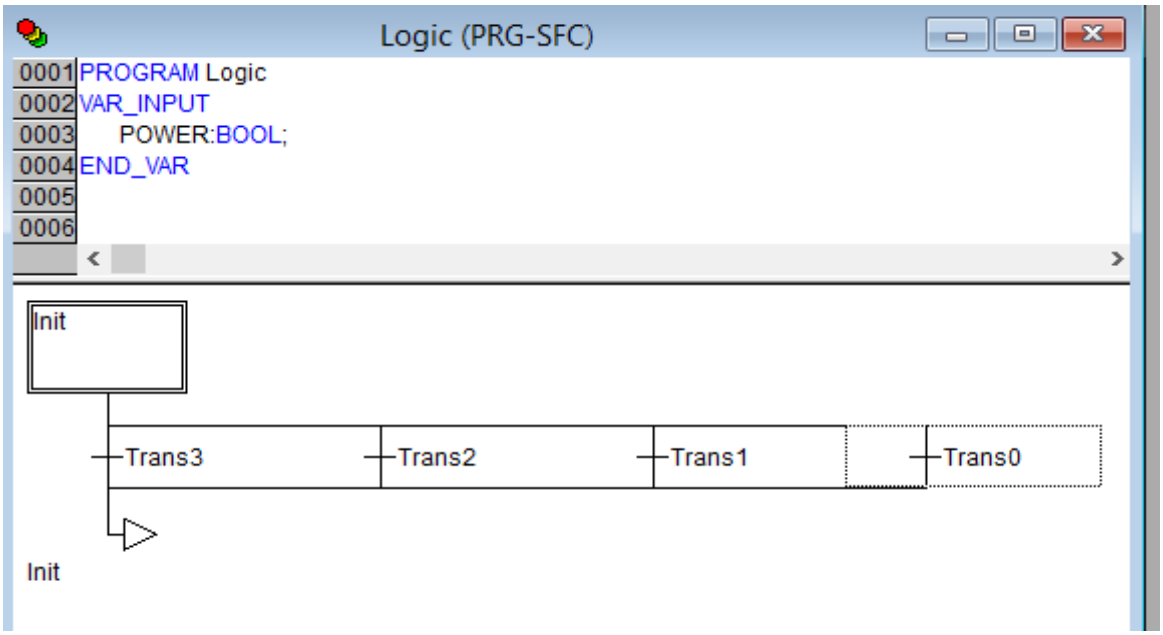


Рис. 4.14. Создание альтернативных ветвей

12. С помощью «Шаг переход (снизу)» создайте по одному шагу в каждой ветви (рис. 4.15).

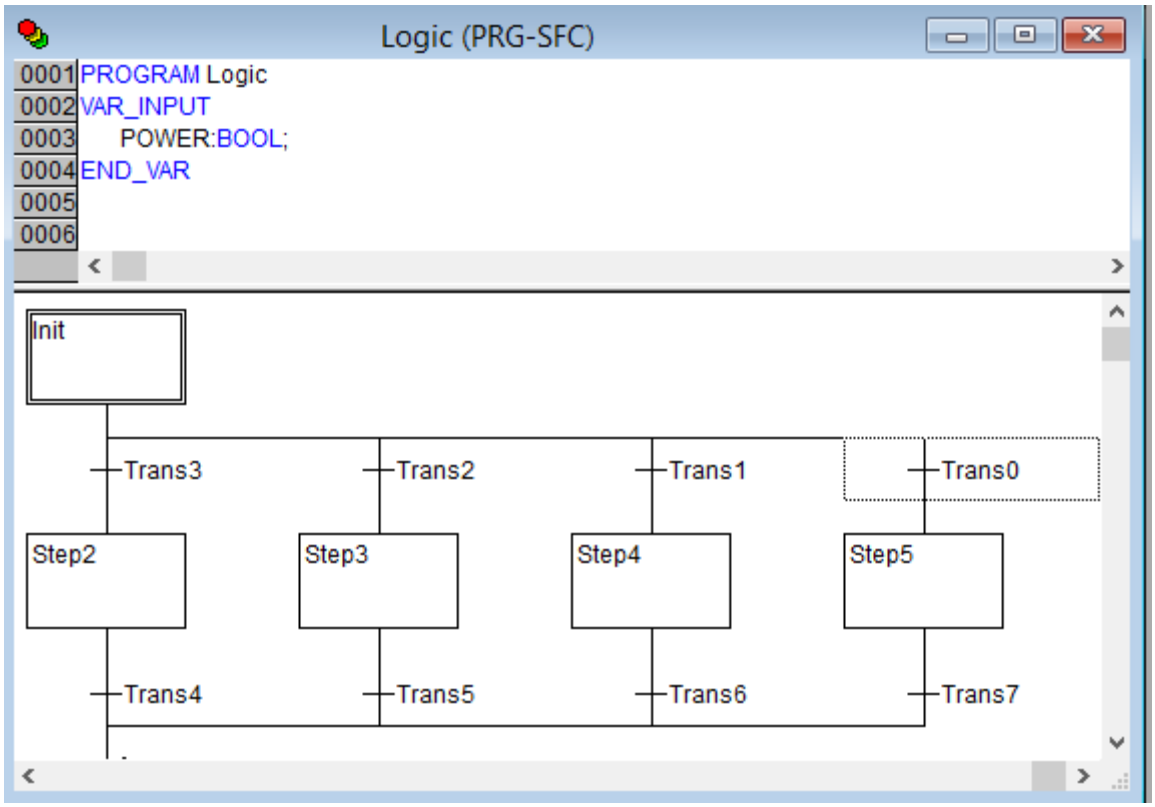


Рис. 4.15. Создание шагов для программирования

13. Переименуйте переходы и шаги. Так, «*YAT1*» и «*YAT2*» - это катушки отключения выключателей «*Q1*» и «*Q2*» соответственно. «*YAC*» - катушка включения секционного выключателя (рис. 4.16).

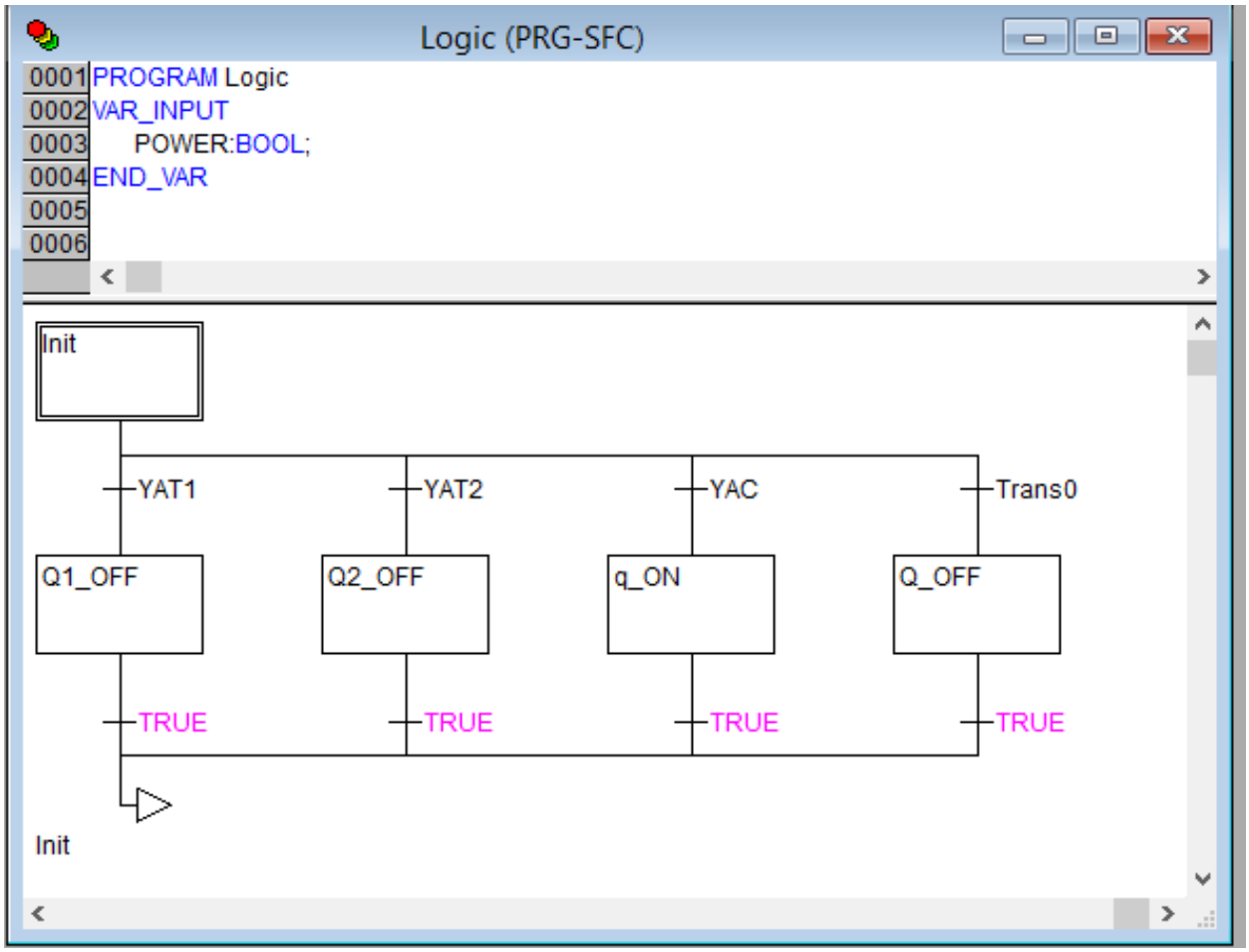


Рис. 4.16. Переименование шагов и переходов, согласно выполняемой функции

14. Запрограммируйте переходы и шаги на языке реализации *ST* (рис. 4.17).

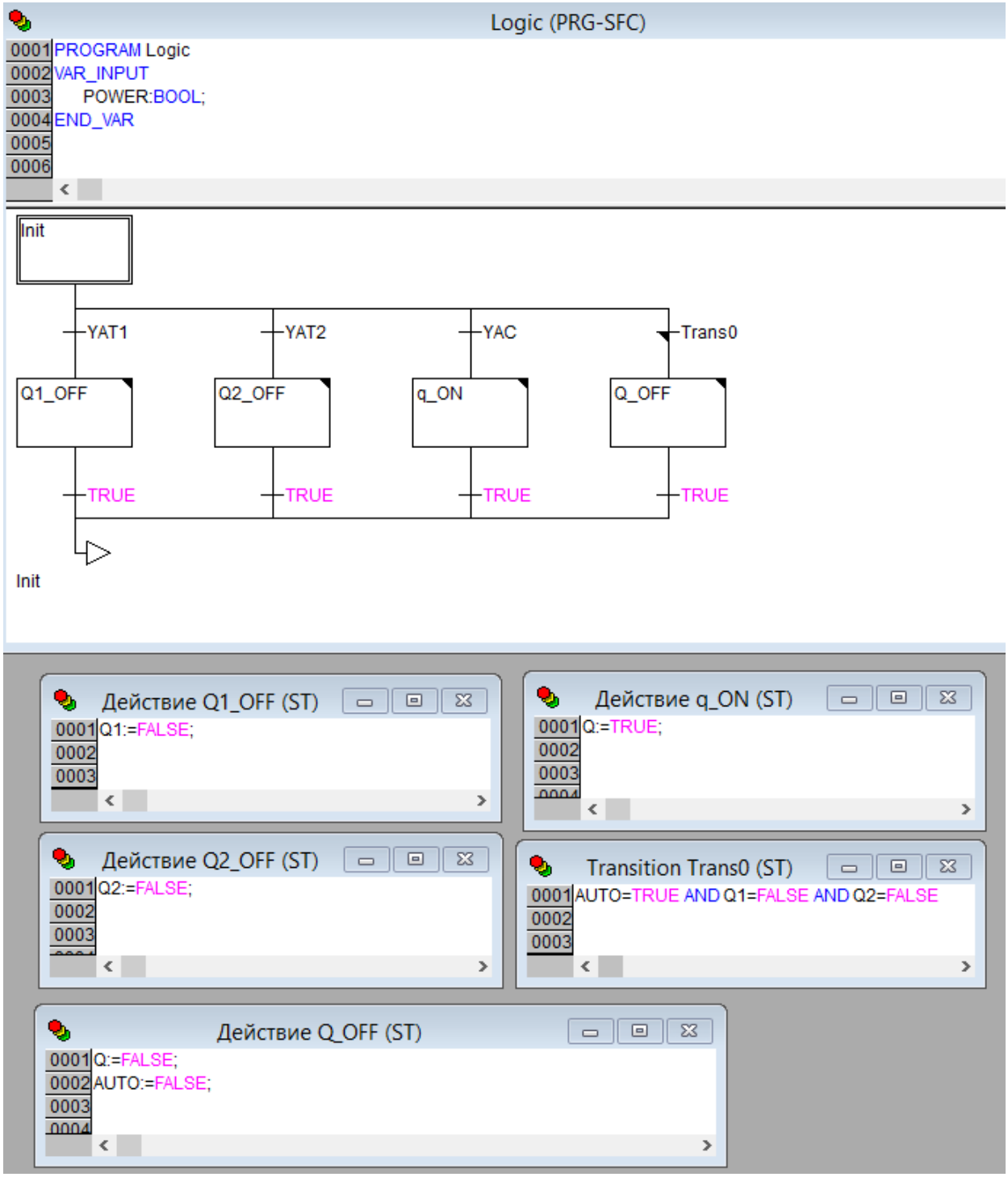


Рис. 4.17. Программирование шагов и переходов

15. В программе *PLC_POU* создайте седьмую цепь. На нее добавьте «Функциональный блок». В разделе «Пользовательские программы» выберите программу, написанную вами на языке *SFC* (рис. 4.18).

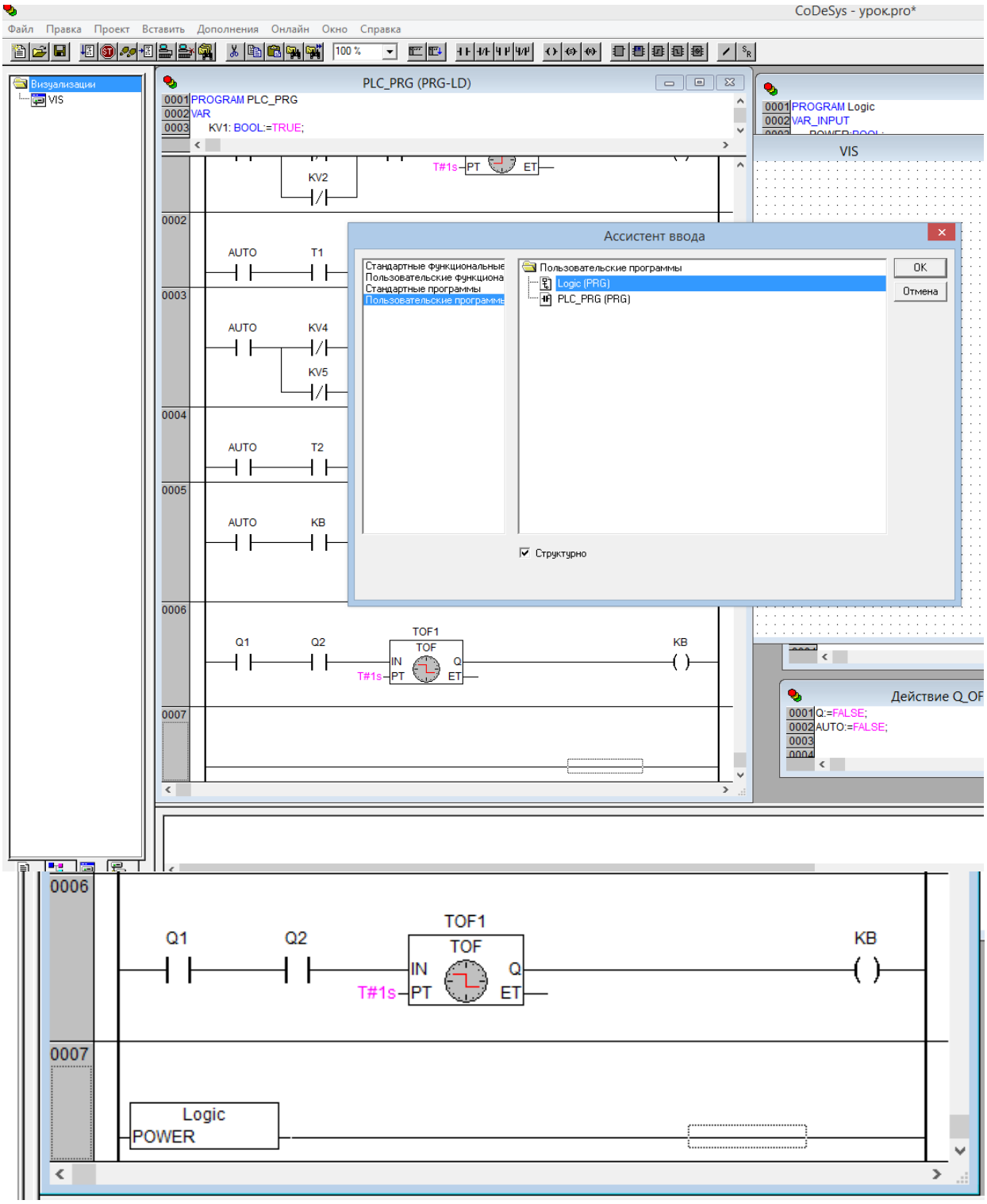


Рис. 4.18. Создание функционального блока программы на языке *SFC*

16. Создайте объект визуализации с произвольным именем и поместите на него несколько элементов, обозначающих выключатели, реле напряжения и кнопку АВР (рис. 4.19).

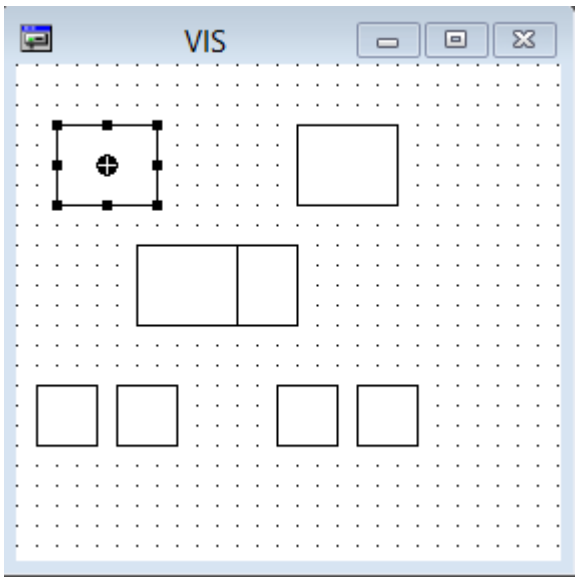


Рис. 4.19. Создание объектов визуализации

17. Запрограммируйте кнопки:

- присвойте название в категории «Текст» (рис. 4.20);
- выберите цвет заливки во включенном и отключенном состоянии в категории цвет (рис. 4.21);
- запрограммируйте изменение цвета относительно соответствующей переменной (рис. 4.22);
- задайте переменную переключения (для выключателей *Q1*, *Q2*, *Q*, *AUTO*) (рис. 4.23);
- задайте переменную кнопку для катушек *KV1*, *KV2*, *KV4*, *KV5* с изменением на ***FALSE*** при нажатии.

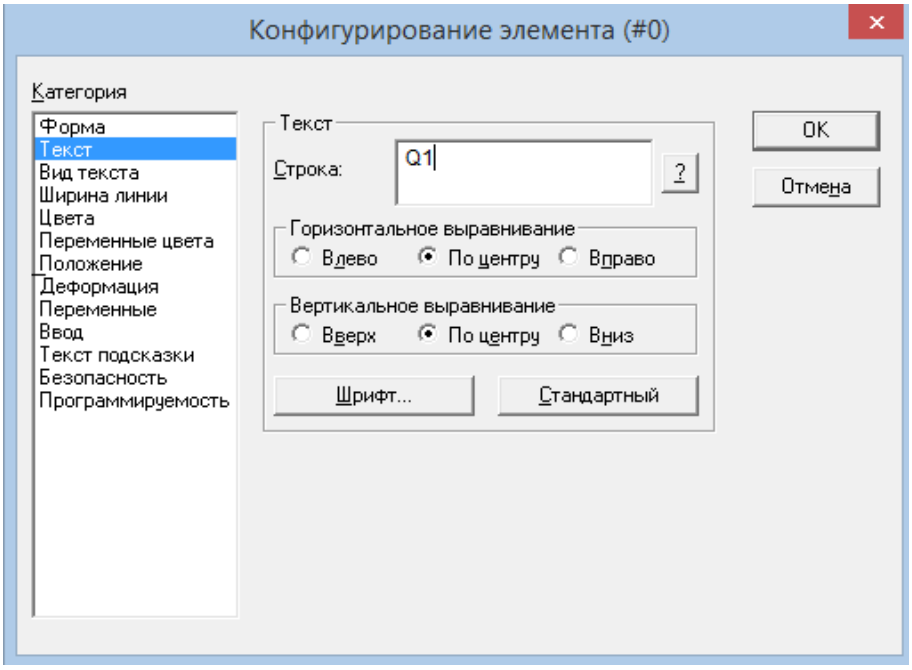


Рис. 4.20. Присвоение названия кнопкам

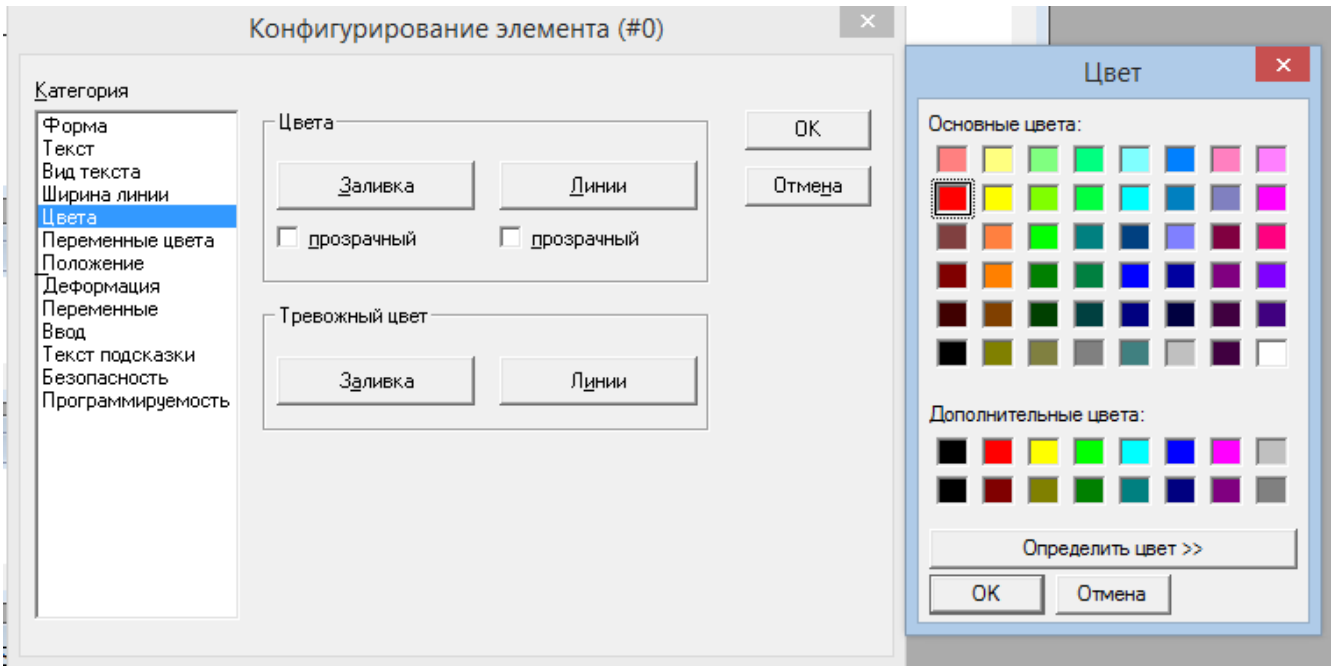


Рис. 4.21. Изменение цвета заливки кнопок

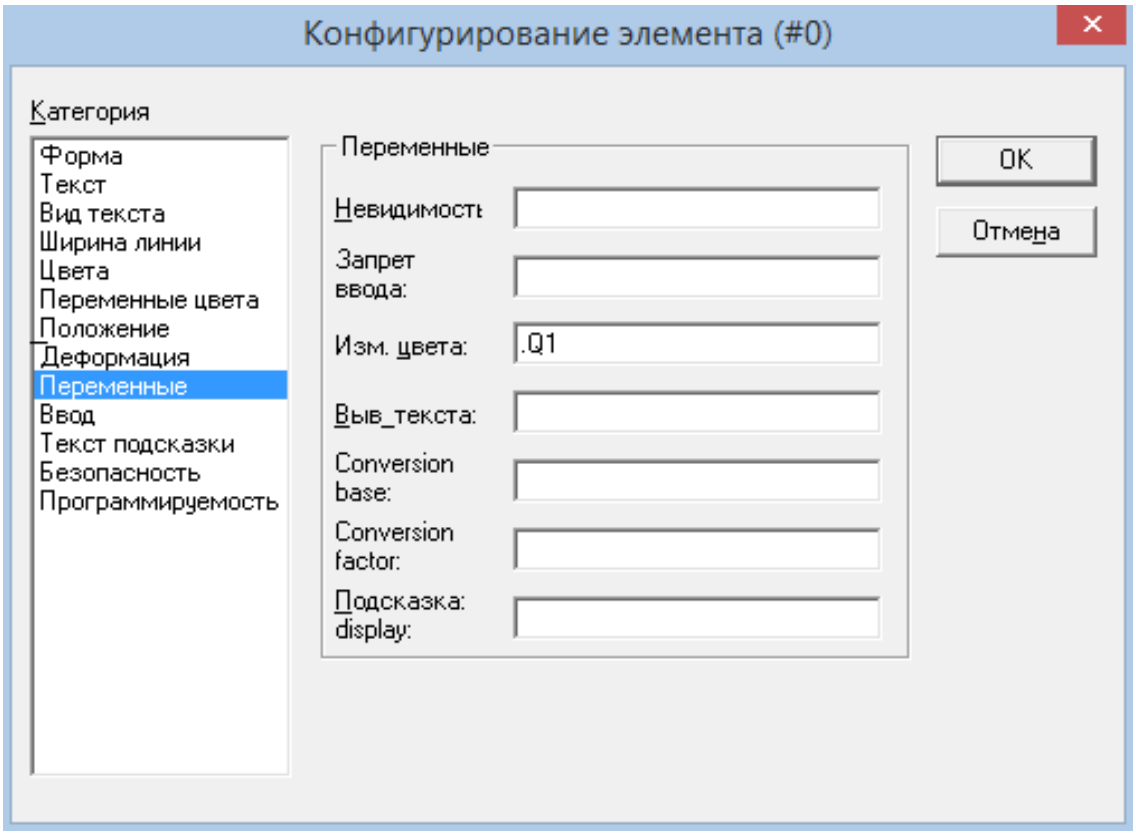


Рис. 4.22. Назначения события изменения цвета

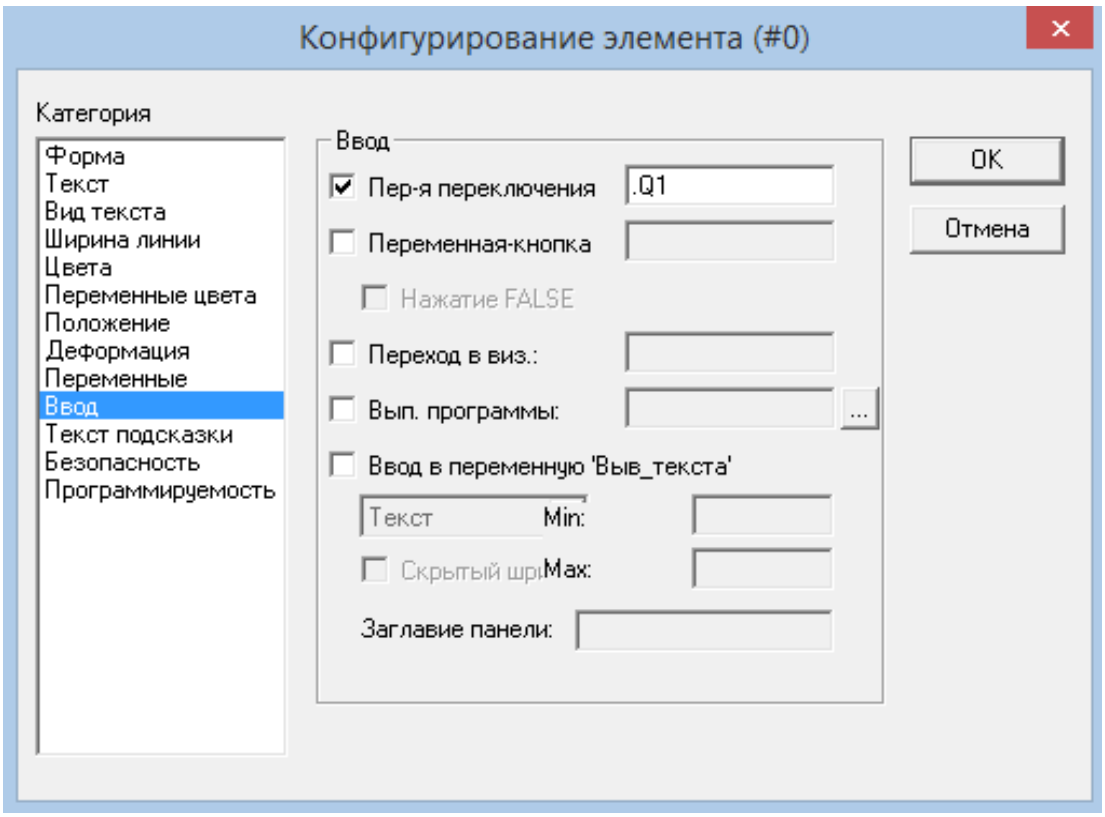


Рис. 4.23. Назначение переменной переключения

Пример оформления визуализации работы программы приведен на рис. 4.24.

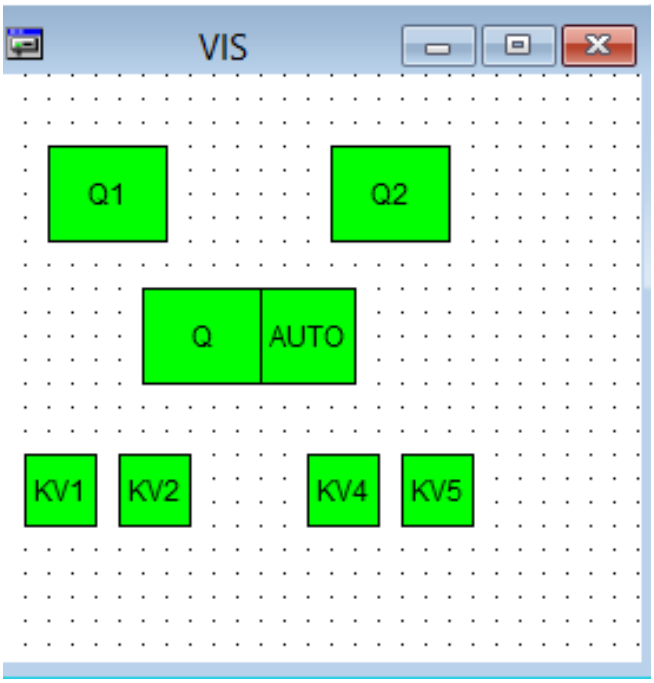


Рис. 4.24. Визуализация работы программы

18. Запуская программу, не забудьте включить режим эмуляции и нажать на «СТАРТ».

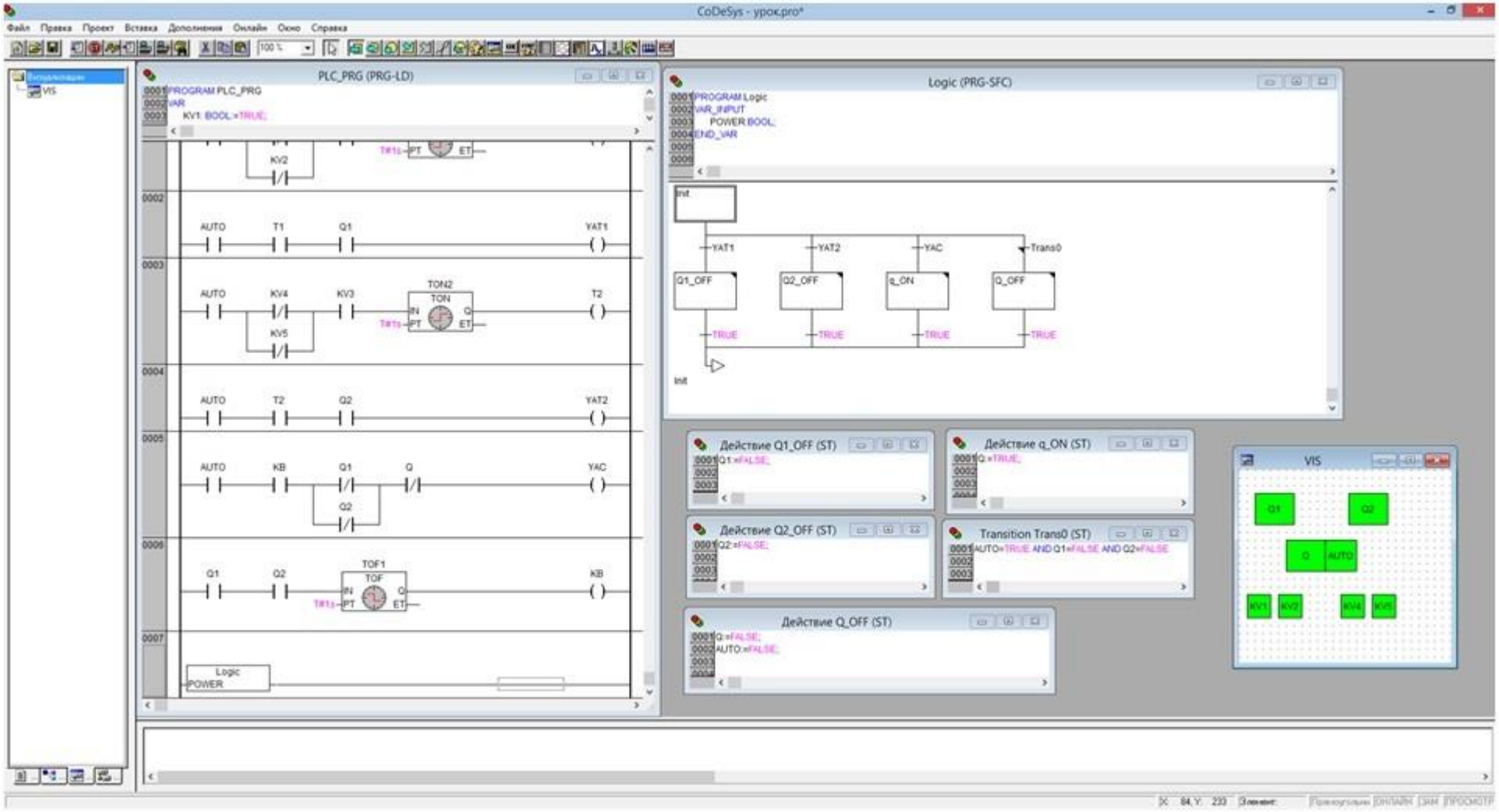


Рис. 4.25. Окончательный вид программы

5. *Практическая работа № 3*
РЕАЛИЗАЦИЯ РАБОТЫ АЧР НА ПЛК

5.1. Краткие теоретические сведения

Частота переменного тока является одним из основных показателей качества электроэнергии. Отклонение частоты в нормальных режимах от номинального значения $f_{НОМ}$ = 50 Гц не должно превышать $\Delta f = \pm$ 0,1 Гц. Допускается кратковременное отклонение частоты не более $\Delta f = \pm$0,2 Гц. Регулирование частоты тока возможно проводить только при наличии в энергосистеме резерва активной мощности (т. е. генераторы загружены не полностью). При возникновении дефицита активной мощности в системе электроснабжения происходит снижение частоты тока, вырабатываемого генераторами. Работа потребителей электроэнергии при пониженной частоте тока (напряжения) приводит к снижению частоты двигателей, а следовательно, снижению их производительности, нарушению технологического процесса производства, браку продукции.

Для восстановления баланса активной мощности часть потребителей на некоторое время должна быть отключена устройствами автоматической частотной разгрузки (АЧР). Устройство АЧР является важным и необходимым средством автоматизации системы электроснабжения.

Автоматическая частотная разгрузка предусматривает отключение потребителей небольшими частями по мере снижения частоты (АЧР1) или по мере увеличения продолжительности существования пониженной частоты (АЧР2).

Объемы отключения нагрузки устанавливают исходя из обеспечения эффективности при любых возможных дефицитах мощности: очередность отключения выбирают так, чтобы уменьшить ущерб от перерыва электроснабжения (применяют большое число устройств и очередей АЧР; более ответственные потребители подключают к более дальним по вероятности срабатывания очередям).

Устройства АЧР предусматриваются по требованию энергоснабжающей организации на подстанциях и распределительных пунктах промышленных предприятий для отключения части электроприемников при возникновении в питающей энергосистеме дефицита активной мощности, сопровождающегося снижением частоты с целью сохранения генерирующих источников и возможно быстрой ликвидации аварии.

В России установлены три категории частотной разгрузки:

1. **АЧР1** — быстродействующая (время действия 0,25 - 0,3 с), имеющая в пределах энергосистемы и отдельных ее узлов различные

уставки по частоте срабатывания и предназначенная для прекращения снижения частоты до опасного уровня (46 Гц). Граничные уставки по частоте: верхний предел не выше 48,5 Гц, нижний — не ниже 46,5 Гц; в отдельных районах страны 49,0 Гц.

2. **АЧР2** — с общей уставкой по частоте и различными уставками по времени, предназначенная для подъема частоты после действия АЧР1 и для предотвращения ее "зависания" на уровне ниже 49 Гц. Единая уставка по частоте обычно принимается равной верхней уставке АЧР1 или на 0,5 Гц больше. Верхний предел не выше 48,8 Гц, а в некоторых районах страны 49,9 Гц. Начальная уставка по времени $t_{\text{Н}}$ = 5 - 10 с, а конечная $t_{\text{К}}$= 60 - 90 с.

3. Третья категория — дополнительная, действующая при возникновении местного глубокого дефицита активной мощности (например, при отделении от энергосистемы энергоемкого потребителя, питаемого местной электростанцией небольшой мощности) и предназначенная для ускорения и увеличения объема частотной разгрузки.

Внутри каждой из первых двух категорий могут назначаться отдельные очереди. В АЧР1 две последовательные очереди отличаются друг от друга уставками срабатывания, но, как правило, не более 0,05-0,1 Гц.

Минимальные интервалы уставок по времени очередей АЧР2 в пределах энергосистемы или района могут составлять до 3 с.

В ряде случаев используется совмещение различных категорий АЧР, когда очереди АЧР1 и АЧР2 действуют на отключение одних и тех же потребителей.

Автоматическое повторное включение (ЧАПВ) приемников и потребителей электроэнергии, отключенных при АЧР, может осуществляться только с разрешения энергоснабжающей организации в тех случаях, когда время восстановления питания действием оперативного персонала или средствами диспетчерского управления недопустимо велико с точки зрения потерь производства. ЧАПВ допустимо на тех присоединениях, внезапное включение которых не может вызвать непредвиденные последствия и опасность для эксплуатационного персонала, самой электроустановки и для механизмов, связанных с ней. К таким присоединениям относятся линии к электроприемникам, включение которых не требует подготовки технологических схем либо полностью автоматизировано, включая и предварительную подготовку соответствующих технологических агрегатов. Включение выключателей устройством ЧАПВ следует производить поочередно (с интервалом не менее 1 с) во избежание перегрузки источников оперативного тока и наложения переходных процессов в системе электроснабжения,

возникающих из-за включения нагрузки. Учитывая, что режим дефицита активной мощности, как правило, связан и с ограниченными возможностями энергосистемы в обеспечении потребителей реактивной мощностью, ЧАПВ целесообразно осуществлять с контролем нормального уровня напряжения на шинах, к которым подключаются группы электроприемников.

Общим измерительным органом в центральном блоке электромеханических устройств АЧР и ЧАПВ служит полупроводниковое реле понижения частоты типа РЧ-1, способное правильно работать в условиях сопровождающего дефицит активной мощности снижения напряжения при $U=0,2 \cdot U_{НОМ}$. Диапазон уставок срабатывания реле 45 - 50 Гц, а уставок возврата — 46 - 51 Гц.

Реле имеет встроенный элемент выдержки времени срабатывания со ступенчатой регулировкой 0,15; 0,3; 0,5 с. Время возврата реле не превышает 0,15 с.

Реле срабатывает сначала на уставке определенной очереди АЧР, после чего перестраивается на уставку возврата, соответствующую частоте сети, при которой разрешается подключение потребителей и электроприемников, отключенных при АЧР. Реле подключается к трансформатору напряжения той секции шин, для которой предусмотрен данный блок устройств АЧР. При отключении этого трансформатора напряжения (например, для проведения ремонтно-восстановительных работ) реле временно присоединяется к трансформатору напряжения другой секции.

Действие АЧР согласовывают с работой устройств автоматического повторного включения (АПВ) и автоматического ввода резерва (АВР). Так, когда АЧР применяют на подстанциях без постоянного дежурного персонала, на которых отсутствует телеуправление, используют АПВ потребителей при восстановлении частоты (ЧАПВ). Частотное АПВ предусматривают также в сетях, где возможно кратковременное снижение частоты при КЗ.

Мощность, отключаемую устройствами АЧР, определяют с учетом того, что в общем случае мощность, потребляемая нагрузкой, зависит от частоты и снижается вместе с ней. Это явление называют регулирующим эффектом нагрузки, оно характеризуется следующим коэффициентом:

$$K_{р.э.н} = \Delta P_{н} / \Delta f,$$

где $\Delta P_{н}$ - изменение снижения суммарной нагрузки, %; Δf - снижение частоты, %;

$$K_{р.э.н} = 1,5\text{-}2,0.$$

На основании приведенной формулы можно определить мощность *Р*отк, отключаемую для восстановления частоты при ее снижении $f_{НОМ}$ до $f < f_{НОМ}$, т. е.

$$P_{ОТК} = (50 - f)\, K_{Р.Э.Н} \cdot P_{Н.НОМ}/50,$$

где $P_{Н.НОМ}$ - мощность нагрузки системы электроснабжения при $f_{НОМ} = 50$Гц.

Устройства ЧАПВ используют для уменьшения перерыва питания отключенных потребителей в условиях восстановления частоты. При размещении устройств и распределении нагрузки по очередям ЧАПВ учитывают степень ответственности потребителей, вероятность их отключения действием АЧР, сложность и длительность неавтоматического восстановления электропитания. Очередность включения нагрузки от ЧАПВ является обратной по отношению к принятой для АЧР [8].

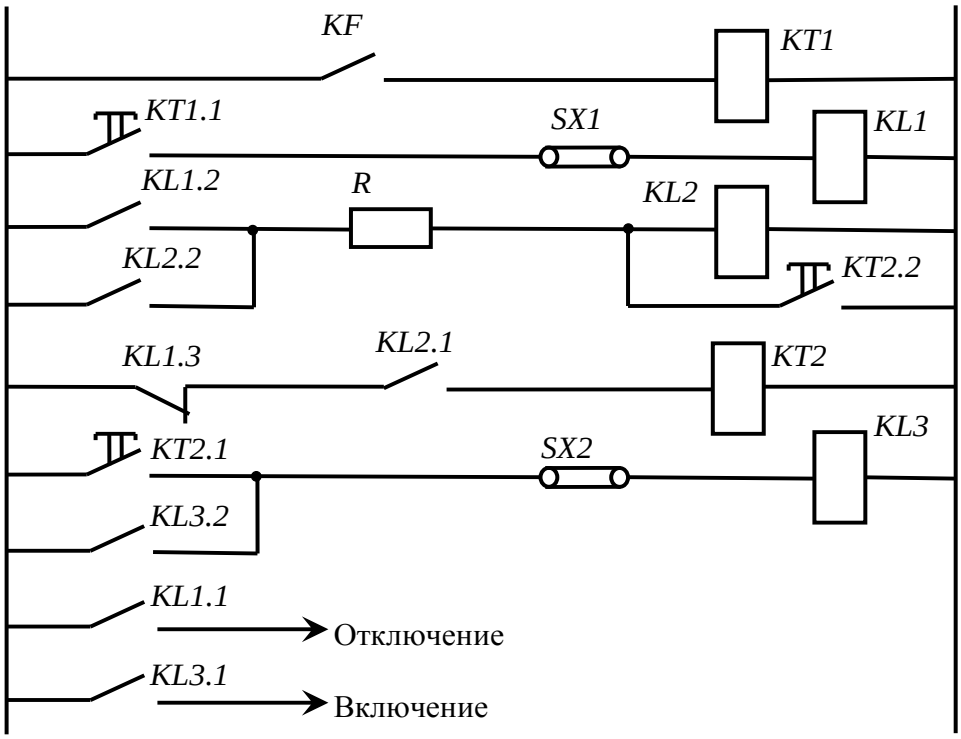


Рис. 5.1. Принципиальная схема одной очереди АЧР с ЧАПВ

На рис. 5.1 приведена принципиальная схема одной очереди АЧР с ЧАПВ. Срабатывание реле частоты *KF* при заданном снижении частоты обеспечивает пуск реле времени *KT1*. Это реле через замыкающий контакт *KT1.1* запускает промежуточное реле *KL1*, выполняющее следующие функции: контакт *KL1.1* обеспечивает отключение потребителей, контакт *KL1.2* используется для запуска реле *KL2* (оно самоудерживается через

контакт *KL2.2),* контакт *KL1.3* находится в цепи реле времени *KT2,* контакт *KL1.4* (на рис. 5.1 не показан) производит изменение уставки реле *KF* (повышает ее). Обычно перенастраивают реле *KF* на уставку 49,5-50 Гц. Следовательно, пока частота не восстановится до указанного уровня, реле *KF* будет держать свой контакт замкнутым.

Если частота восстановится, то реле *KF, KT1, KL1* вернутся в исходное положение. Возврат реле *KL1* и замыкание его контакта *KL1.3* обеспечивают пуск реле *KT2.* С выдержкой времени контактом *KT2.1* будет запущено реле *KL3,* которое производит повторное включение потребителей. Для пуска *KL3* используется проскальзывающий контакт *KT2.1,* реле *KL3* самоудерживается *(KL3.2).* Упорный контакт *KT2 с* выдержкой времени *(KT2.2)* большей, чем у *KT2.1* (на 1-2 с), возвращает схему в исходное положение, шунтируя обмотку *KL2.* Используя накладки *SX1* и *SX2,* схема может быть настроена только на АЧР или же на АЧР и ЧАПВ и т. д.

5.2. Задание

1. Создать коммутационную программу для принципиальной электрической схемы работы одной очереди АЧР
2. Создать визуализацию работы АЧР.
3. Подготовить отчет по плану:

- цель работы;
- описание принципа работы АЧР и электрической схемы;
- описание визуализации. Блок схема. Алгоритм построения визуализации;
- выводы.

5.3. Выполнение работы в программе *CodeSys*

1. Создайте программу *PLC_PRG* на языке *LD.*
2. В области описания переменных опишите локальные переменные:

```
PROGRAM PLC_PRG
VAR
    Q1: BOOL;
    Q2: BOOL;
    Q3: BOOL;
    TON1:TON;
    TON2: TON;
    TON3: TON;
    TOF1: TOF;
```

```
        TOF2: TOF;
        TOF3: TOF;
    END_VAR,
```

где *Q1-Q3* - состояние выключателей питающих линий потребителей; *TON1-TON3* - таймер выдержки времени включения выключателя при восстановлении частоты; *TOF1-TOF3* - таймер выдержки времени отключения выключателя при срабатывании ЧАПВ.

3. В пункте «Глобальные переменные» объявите:

```
VAR_GLOBAL
        YAC1:BOOL;
        YAC2:BOOL;
        YAC3:BOOL;
        YAC11: BOOL;
        F:REAL:=50;
END_VAR,
```

где *YAC1-YAC3* - катушки отключения выключателей при срабатывании АЧР; *YAC11* - катушки отключения выключателей при срабатывании ЧАПВ; *F* – варьируемое значение частоты с начальным значением 50 Гц;

4. Добавьте новый объект *POU* программу на языке реализации *ST*.

5. Опишите локальные переменные подпрограммы:

```
PROGRAM katushki
VAR_INPUT
        GO:BOOL;
END_VAR
VAR
        FN:REAL:=50;
        F1:REAL:=49;
        F2:REAL:=48;
        F3:REAL:=47;
END_VAR,
```

где *GO* (название переменной может быть любым) – входная переменная, необходимая для работы подпрограммы; *FN*, *F1-F3* – значение уставок частоты для срабатывания разных очередей АЧР.

6. Согласно заданным уставкам, напишите текст программы для питания катушек отключения выключателей при достижения определенной уставки АЧР и ЧАПВ:

```
IFF<FNTHEN//этим блоком мы программируем ЧАПВ//
YAC11:=TRUE;
ELSE YAC11:=FALSE;
END_IF
    IFF>=F2 ANDF<F1 THEN//отключаем первую очередь//
    YAC1:=TRUE;
    ELSE
        IFF>=F3 ANDF<F2 THEN// отключаем вторую очередь//
        YAC2:=TRUE;
        YAC1:=TRUE;
        ELSE
            IFF<F3 THEN//отключаем третью очередь//
            YAC3:=TRUE;
            YAC2:=TRUE;
            YAC1:=TRUE;
            END_IF
    END_IF
END_IF

IFF>=FNTHEN//этим блоком мы производим включение потребителей//
    YAC3:=FALSE;
                YAC2:=FALSE;
                YAC1:=FALSE;
    YAC11:=FALSE;
    END_IF
```

7. Создайте цепи в основной программе *PLC_PRG*. Не забудьте добавить функциональный блок, в котором необходимо описать программу на языке *ST* (рис. 5.2, 5.3). Обратите внимание на таймеры: параметры *TON* и *TOF* необходимо менять согласно выполняемой функции.

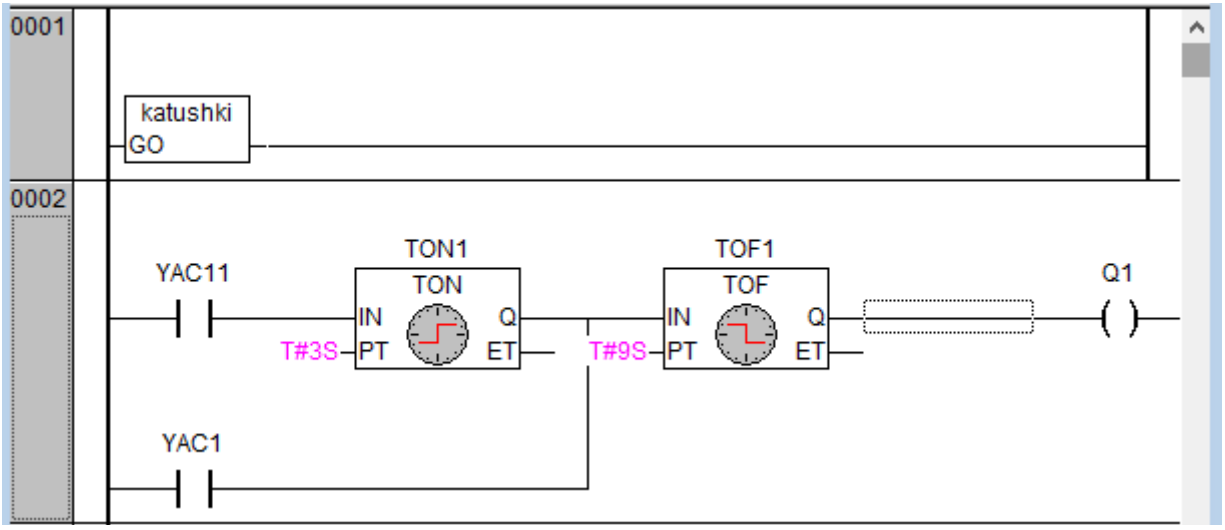


Рис. 5.2. Функциональный блок программы на языке *ST* и первая ветвь электрической схемы

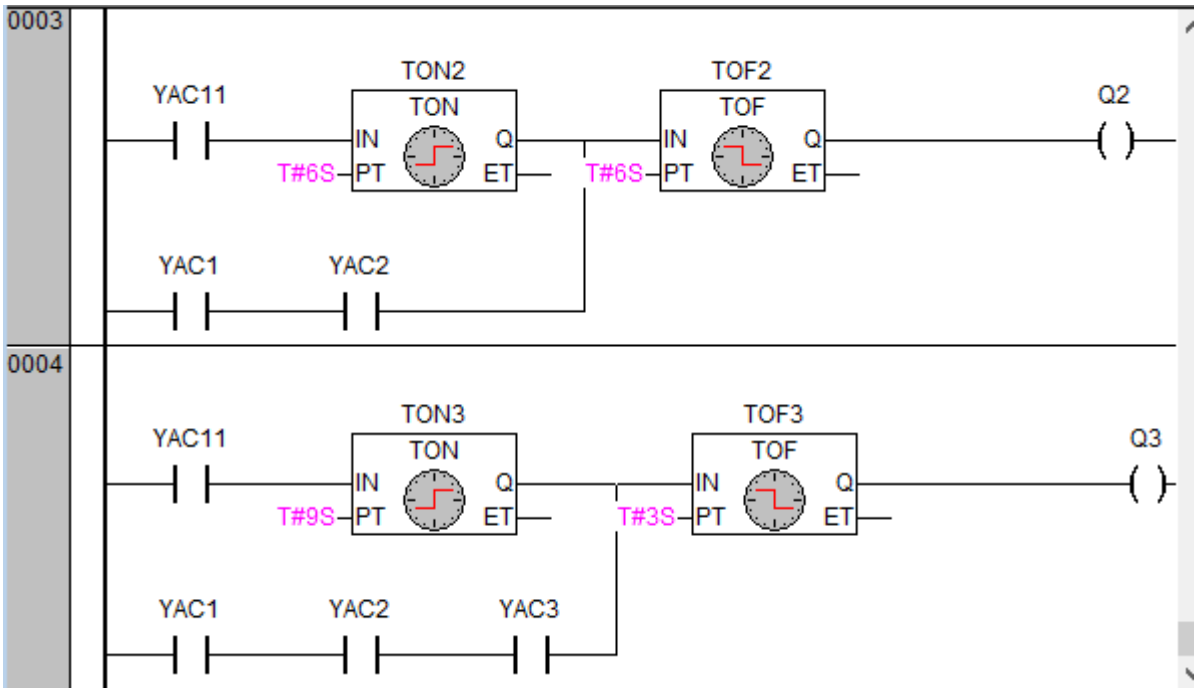


Рис. 5.3. Вторая и третьи ветви программы на языке *LD*

8. Создайте визуализацию работы программы:

- создайте три прямоугольника с изменением цвета относительно переменных *Q1*, *Q2*, *Q3* (не программируйте ввод с этих элементов, они нужны только для индикации) (рис. 5.4);
- создайте «ползунок» для изменения частоты (рис. 5.4);

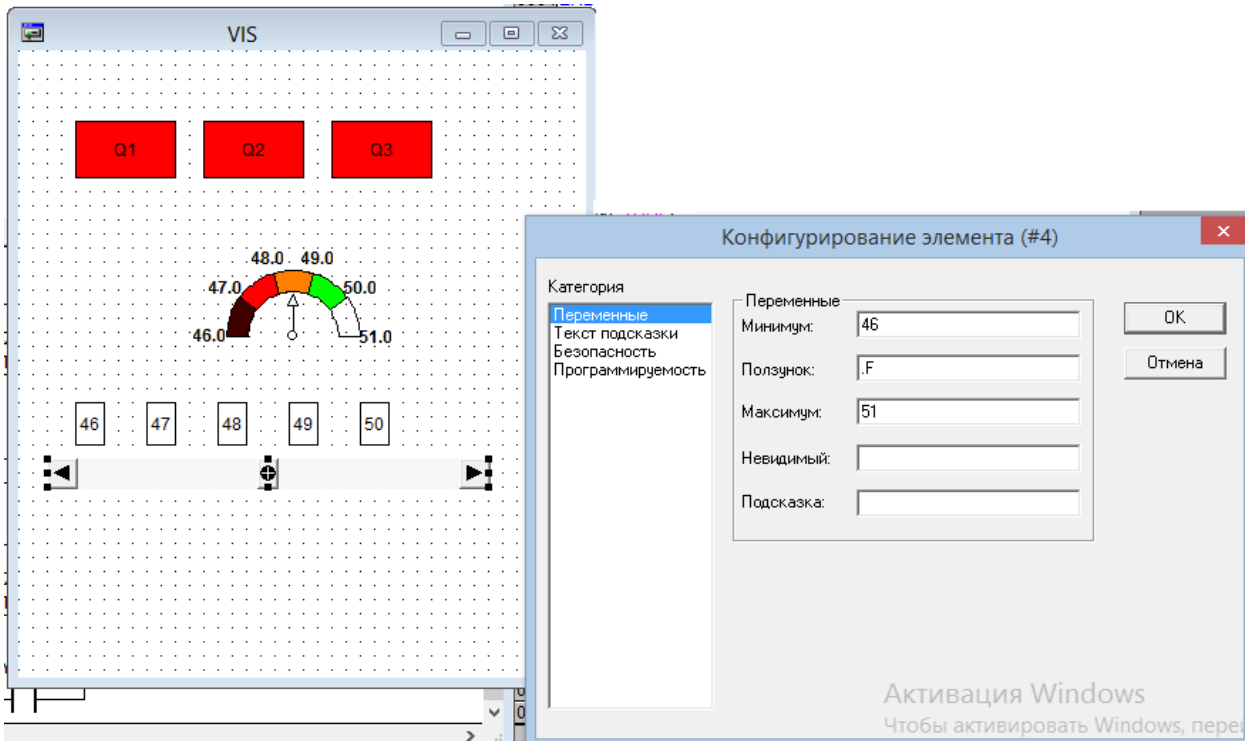


Рис. 5.4. Настройка элемента «ползунок»

- создайте «индикатор» для индикации значения частоты (рис. 5.5).

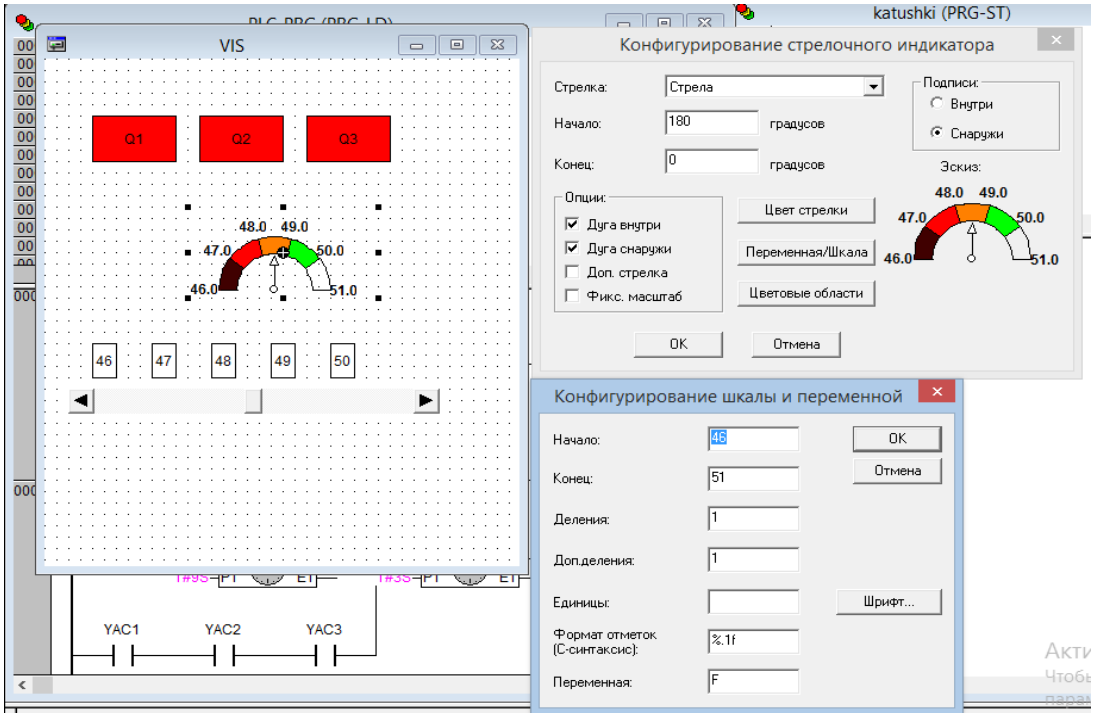


Рис. 5.5. Настройка элемента «индикатор»

9. Запуская программу, обратите внимание что выдержка времени стоит как на отключение от ЧАПВ (3 с, 6 с и 9 с), так и включение при возвращении частоты к номинальному значению.

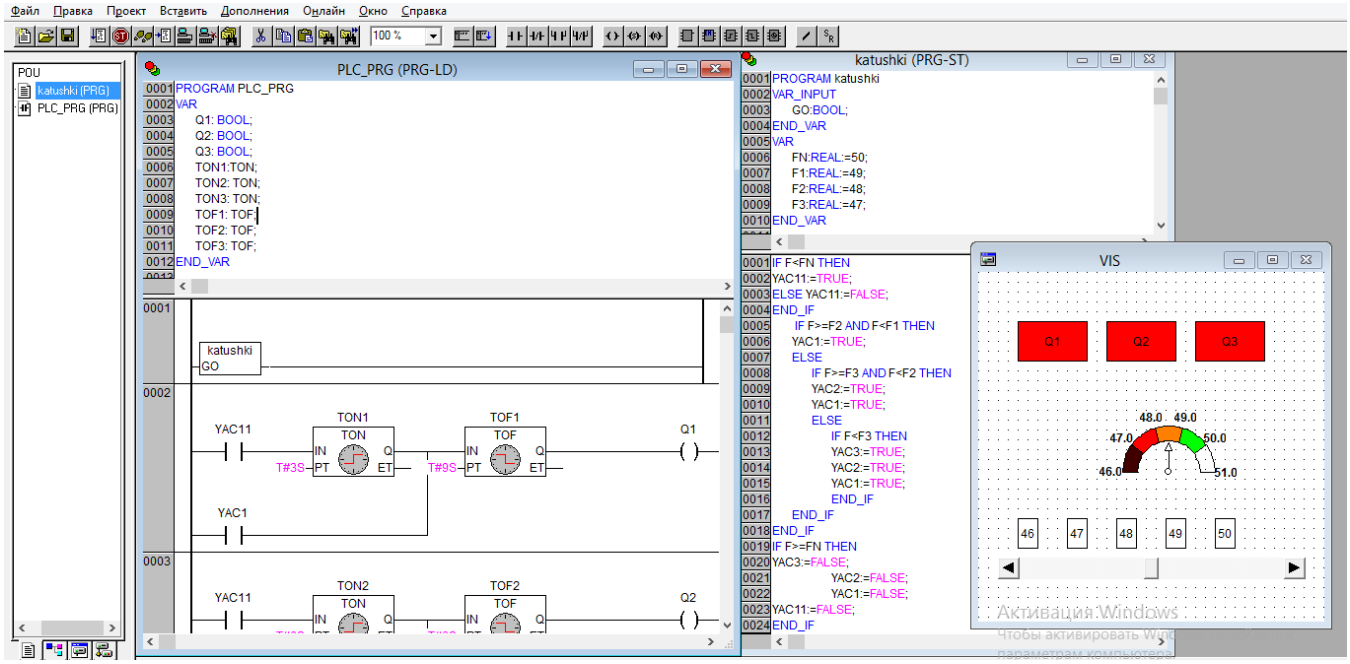


Рис. 5.6. Внешний вид программы

6. *Практическая работа № 4.*
РЕАЛИЗАЦИЯ АВТОМАТИЧЕСКОГО ВКЛЮЧЕНИЯ УСТРОЙСТВ КОМПЕНСАЦИИ РЕАКТИВНОЙ МОЩНОСТИ

6.1. Краткие теоретические сведения

Для повышения коэффициента мощности и улучшения качества электроэнергии широкое применение на промышленных предприятиях находят конденсаторные батареи (КБ) высокого и низкого напряжения. Многие промышленные предприятия в течение суток имеют неравномерный график активной и реактивной нагрузок. В связи с этим изменяется и потребность в реактивной мощности, вырабатываемой КБ для поддержания на предприятиях требуемого коэффициента мощности. Неравномерный график нагрузки и отсутствие автоматического регулирования мощности компенсирующих устройств (КУ) может вызвать повышение напряжения в отдельных участках сети, что недопустимо для некоторых приемников электроэнергии и связано с излишним расходом электроэнергии и дополнительными потерями в сетях.

Для обеспечения экономичной работы конденсаторных установок применяют автоматическое регулирование мощности КБ. Регулирование может быть одноступенчатым и многоступенчатым. При одноступенчатом регулировании мощности КБ уменьшение нагрузки вызывает автоматическое отключение всей конденсаторной установки. При многоступенчатом регулировании происходит автоматическое включение или отключение отдельных батарей или секций, каждая из которых снабжена своим выключателем.

Для регулирования реактивной мощности используется автоматическое регулирование возбуждения синхронных машин и регулирование КБ. Автоматическое регулирование КБ может осуществляться в функции напряжения, времени суток, реактивной мощности и по комбинированным схемам в зависимости от нескольких факторов.

Регулирование мощности КБ в зависимости от напряжения на шинах подстанции используют в тех случаях, когда конденсаторные установки, наряду с основной своей функцией, используют также и для регулирования напряжения. Автоматическое управление КБ в зависимости от напряжения осуществляется при помощи реле максимального и минимального напряжения. При снижении нагрузки и повышении напряжения реле максимального напряжения отключает всю батарею или часть ее. При увеличении нагрузки и снижении напряжения реле минимального напряжения снова включает КБ. Во избежание

ложных переключений при кратковременных изменениях напряжения включение и отключение производится с выдержкой времени около 15 с.

На рис. 6.1 показана принципиальная схема одноступенчатого управления конденсаторной установкой в функции времени [9].

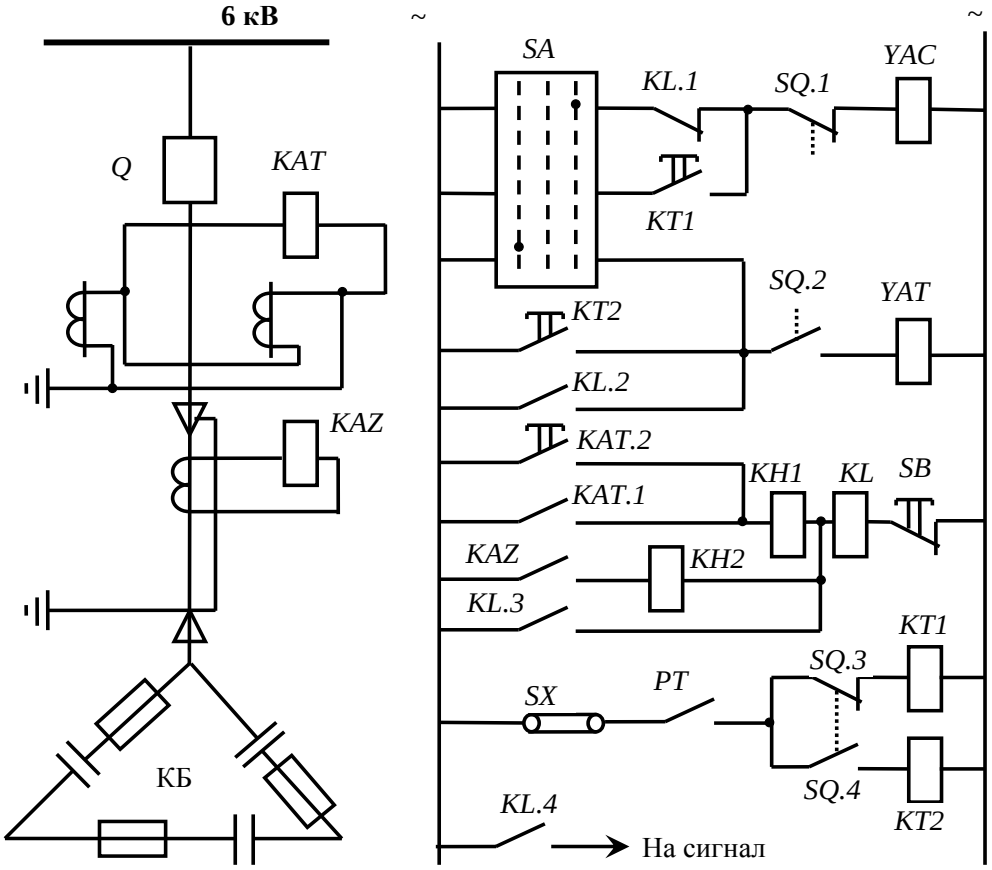


Рис. 6.1. Принципиальная схема одноступенчатого управления конденсаторной установкой в функции времени:

YAC, YAT - электромагниты включения и отключения соответственно; *KAT* - защита конденсаторной установки от КЗ (электромагнитный элемент *KAT.1*) и от перегрузки токами высших гармоник (индукционный элемент *KAT.2*) на реле РТ-80; *KAZ* - защита конденсаторной установки

6.2. Задание

1. Создать коммутационную программу для принципиальной электрической схемы одноступенчатого управления конденсаторной установкой.

2. Создать визуализацию работы КРМ.

3. Подготовить отчет по плану:

- цель работы;
- описание принципа работы системы автоматического управления КРМ и электрической схемы;

- описание визуализации. Блок схема. Алгоритм построения визуализации;
- выводы.

6.3. Выполнение работы в программе *CodeSys*

Для реализации программы создайте программу на языке *LD*, опишите локальные переменные, нанесите элементы принципиальной электрической схемы на лестницу и присвойте имена переменных по схеме (рис. 6.2).

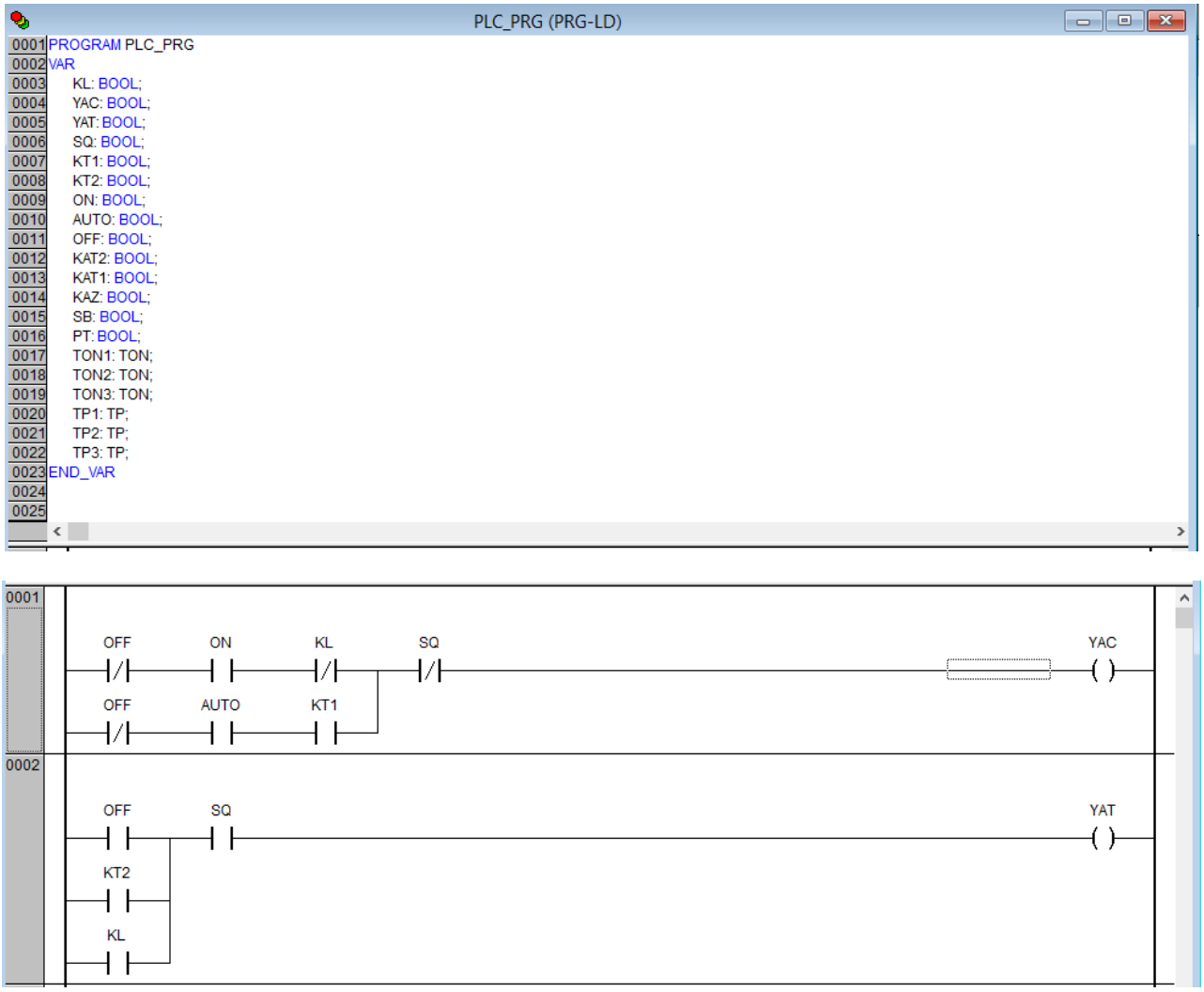


Рис. 6.2. Реализация принципиальной схемы в программе *PLC_PRG* на языке *LD* (начало)

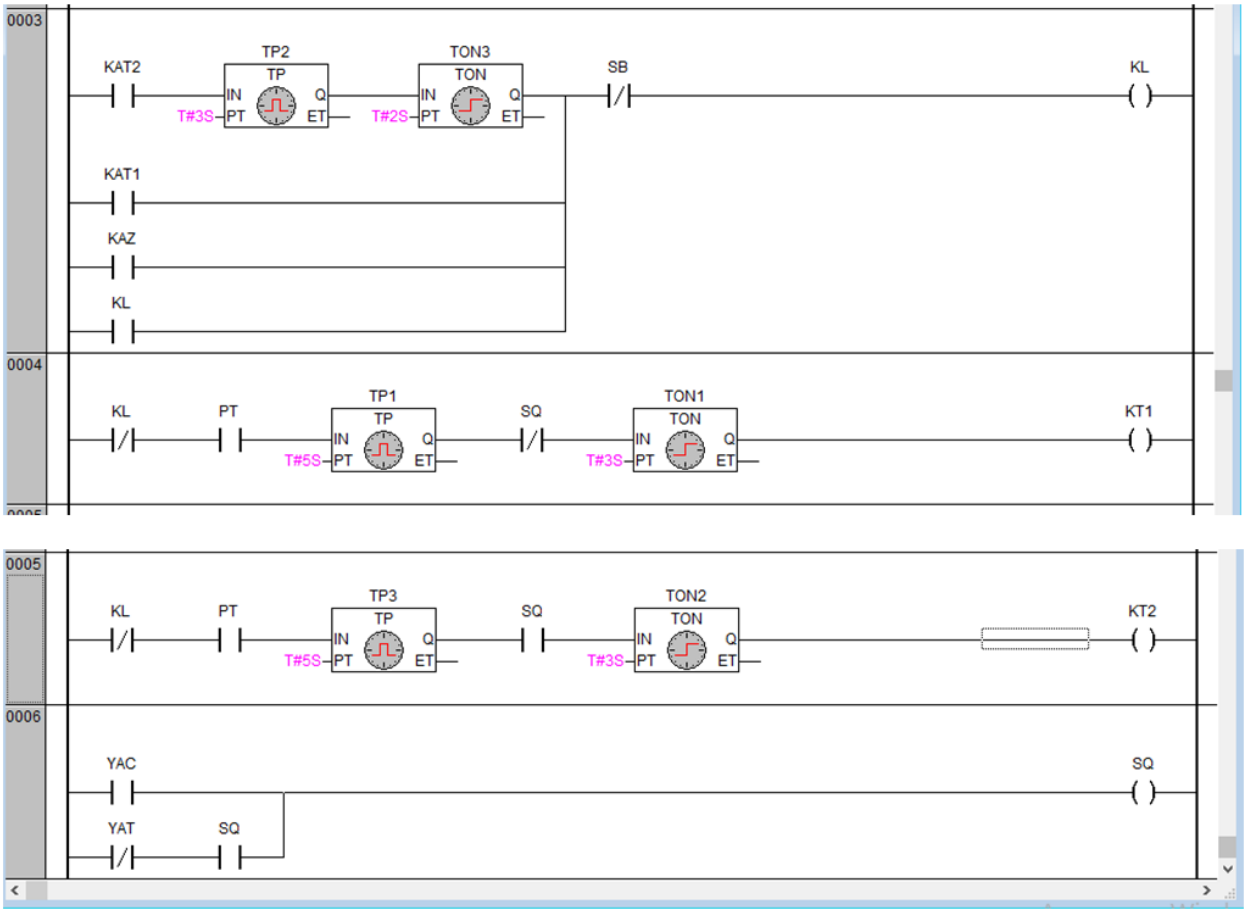


Рис. 6.3. Реализация принципиальной схемы в программе *PLC_PRG* на языке *LD* (окончание)

В области описания локальных переменных опишите компоненты схемы:

```
PROGRAMPLC_PRG
VAR
    KL: BOOL;//Промежуточное реле для отключения
    YAC: BOOL; //Катушка включения выключателя
    YAT: BOOL; //Катушка отключения выключателя
    SQ: BOOL;//Состояние контактов выключателя
    KT1: BOOL;//Реле выдержки времени
    KT2: BOOL;
    ON: BOOL; //Кнопка ручного включения
    AUTO: BOOL; //Кнопка автоматического управления
    OFF: BOOL;//Кнопка отключения
    KAT2: BOOL; //Защита от перегрузки токами высших гармоник
    KAT1: BOOL; //Защита конденсаторной установки от КЗ
```

KAZ: BOOL;
SB: BOOL; //Кнопка сброса самоудержания
PT: BOOL; //Контакты часов
TON1: TON;//Таймер выдержки времени
TON2: TON;
TON3: TON;
TP1: TP;//Таймер
TP2: TP;
TP3: TP;
END_VAR

Визуализация

Визуализацию работы программы реализуем с помощью кнопок ручного и автоматического управления, кнопок активации защитных реле и кнопки активации контактов часового механизма (рис. 6.3). Если КБ активна, то горит надпись «Включено».

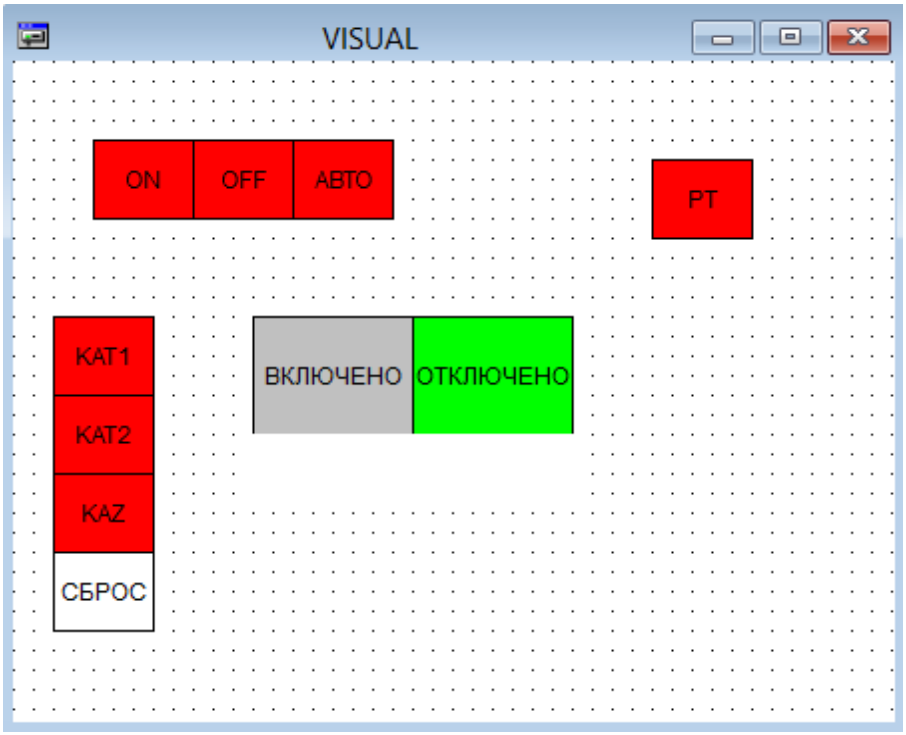


Рис. 6.3. Визуализация

При срабатывании реле защиты необходимо с помощью кнопки «Сброс» отключить катушку самоудерживания. Все кнопки реализуются именно как кнопки, а «*OFF*» и «*AUTO*» как переменные переключения (рис. 6.4).

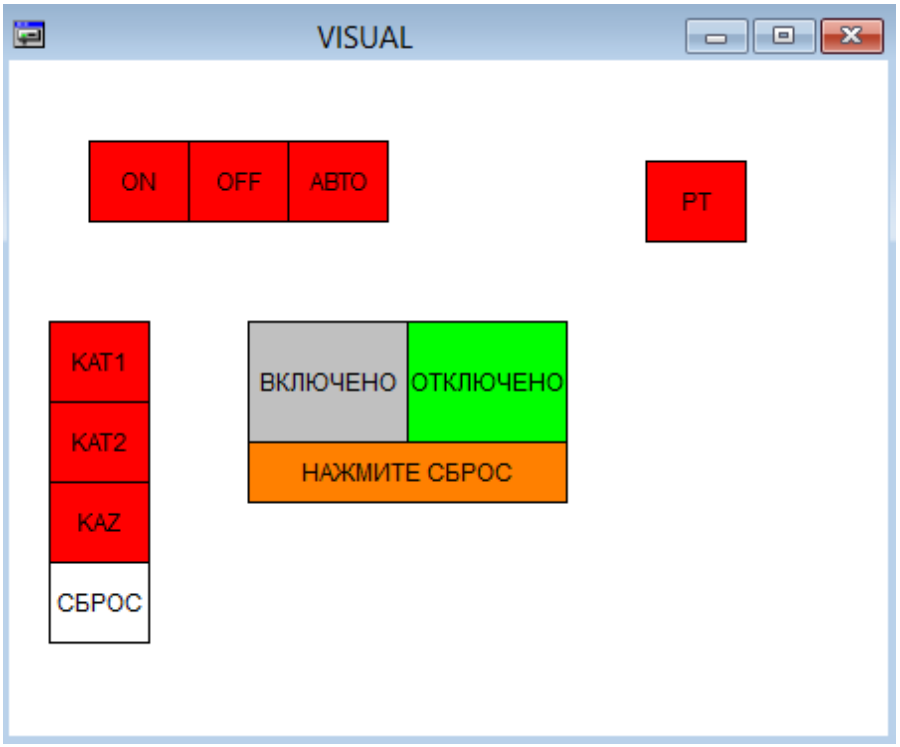


Рис. 6.4. Визуализация срабатывания промежуточного реле

Работа программы

В зависимости от положения выключателя *Q* работает либо реле времени *KT1*, либо *KT2*. Контакты *PT* электрических часов (замыкаются на *t*=5с), замыкаясь, включают *KT1* или *KT2*. При отключенном *Q* работает *KT1*, при включенном *Q* - *KT2*. Выдержки времени обоих реле выбраны одинаковыми 3 с. При отключенном выключателе *Q* его вспомогательные контакты *SQ.2* и *SQ.4* будут разомкнуты, а *SQ.1* и *SQ.3* - замкнутыми. При замыкании контакта *PT* получит питание реле *KT1* и с выдержкой времени замкнет свои контакты в цепи электромагнита включения выключателя *Q*, выключатель включится, а следовательно, включится конденсаторная установка.

Несмотря на то, что после включения выключателя *Q* получит питание, реле *KT2 (SQ.1* и *SQ.3* разомкнутся; *SQ.2* и S*Q.4* — замкнутся; *PT* будет еще замкнут), но оно не успеет сработать, так как контакт *PT* разомкнется раньше, чем замкнется контакт *KT2* в цепи электромагнита отключения привода выключателя *Q*. Таким образом, конденсаторная установка останется подключенной к шинам 0,4 кВ до очередного замыкания *PT*.

При срабатывании защит *KAT* или *KAZ* получает питание промежуточное реле *KL*, самоудерживается *KL.3* и разрывает цепь электромагнита включения привода выключателя *Q* (контакт *KL.1).* Самоудерживание может быть снято кнопкой «Стоп» *SB.*

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие типы данных существуют?
2. Что такое локальные и глобальные переменные?
3. Назначение Target-файла.
4. Какова структура *LD*-программы?
5. Как в *LD*-программе задают проверку состояния входных дискретных сигналов?
6. Какими командами в *LD*-программе формируют выходные дискретные сигналы?
7. Каким образом можно фиксировать выходные дискретные сигналы?
8. Как выполнить конфигурацию входных и выходных переменных?
9. Как проверить правильность *LD*-программы?
10. Как загрузить код программы в ПЛК?
11. Как запустить программу в контроллере?
12. Таймеры. Способ задания временного интервала в *CoDeSys*.
13. Отличие таймеров *TON* от *TOF*.
14. Типы компонентов организации программ (*POU*).
15. Структура ПЛК.
16. *SFC* – диаграммы. Описание, состав.
17. Язык *ST*. Описание. Область применения.
18. В чем состоит необходимость визуализации проекта?
19. Серверы данных (*DDE* и *OPC*). Назначение.
20. Опишите средства, реализующие выполнение программ для ПЛК.
21. Операторы и циклы на языке *ST*.

БИБЛОГРАФИЧЕСКИЙ СПИСОК

1. **Петров, И. В.** Программируемые контроллеры. Стандартные языки и приемы прикладного проектирования; ред. В. П. Дьяконова / И. В. Петров - М.: СОЛОН-Пресс, 2004. – 256 с.

2. CODESYS V2.3 - CODESYS V3. [Электронной ресурс] – URL: https://www.codesys.com.

3. CODESYS v.2 — CODESYS 2.3. [Электронной ресурс] – URL: http://www.owen.ru/catalog/codesys_v2/opisanie.

4. Первые шаги с CoDeSys; Русская редакция ПК «Пролог». – 2004. [Электронной ресурс] – URL:http://www.kipshop.ru/CoDeSys/steps/first_steps_with_codesys_ru.pdf

5. Руководство пользователя по программированию ПЛК в CoDeSys 2.3; Русская редакция ПК «Пролог». 2006. [Электронной ресурс] – URL: http://www.kipshop.ru/CoDeSys/steps/codesys_v23_ru.pdf

6. Визуализация CoDeSys. Дополнение к руководству пользователя по программированию ПЛК в CoDeSys 2.3; Русская редакция ПК «Пролог». 2006. [Электронной ресурс] – URL: http://www.kipshop.ru/CoDeSys/steps/codesys_visu_v23_ru.pdf

7. Правила устройства электроустановок. Все действующие разделы шестого и седьмого изданий с изменениями и дополнениями по состоянию на 1 июля 2010 г. – М.: КНОРУС, 2010. – 488 с.

8. **Андреев, В. А**. Релейная защита и автоматика систем электроснабжения: Учебник для вузов / В. А. Андреев. – М.: Высш. шк., 2006. – 639 с.

9. **Ершов, А.М.** Автоматизация систем электроснабжения: учеб. пособие для студентов специальности 100400. – Челябинск: Изд-во ЮУрГУ, 2009. - 100 с.

10. **Котов, В.Е.** Сети Петри. - М.: Наука (Главная редакция физико-математической литературы), 1984. – 160 с.

ЛОСКУТОВ АЛЕКСЕЙ БОРИСОВИЧ
ЛОСКУТОВ АНТОН АЛЕКСЕЕВИЧ
ЗЫРИН ДМИТРИЙ ВЛАДИМИРОВИЧ
ШУМСКИЙ НИКИТА ВАСИЛЬЕВИЧ

ПРОГРАММИРОВАНИЕ ПЛК В CODESYS

Редактор **Т. В. Третьякова**
Компьютерный набор и верстка авторов

Подписано в печать 25.09.2018. Формат 60x84 1/16.
Бумага офсетная. Печать офсетная. Усл. печ. л 6,5.
Тираж 100 экз. Заказ ___.

Нижегородский государственный технический университет
им. Р. Е. Алексеева.
Типография НГТУ.
Адрес университета и полиграфического предприятия:
603950, г. Нижний Новгород, ул. Минина, 24.