Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Институт информационных технологий

Кафедра информационных систем и технологий

О. А. Шведова, М. В. Шило

МИКРОПРОЦЕССОРНЫЕ СИСТЕМЫ УПРАВЛЕНИЯ И СЕТЕВЫЕ ТЕХНОЛОГИИ

Рекомендовано УМО по образованию в области информатики и радиоэлектроники в качестве учебно-методического пособия для специальности 1-53 01 07 «Информационные технологии и управление в технических системах»

Минск БГУИР 2015

УДК 004.31-022.53(076)
ББК 32.973.202я73
Ш 34

Р е ц е н з е н т ы :

кафедра информационных технологий Белорусского государственного университета (протокол №6 от 05.03.2015);

начальник сектора СКБ-4 НПООО «ОКБ ТСП»,
кандидат технических наук, доцент А. Г. Стрижнев

Шведова, О. А

Ш34 Микропроцессорные системы управления и сетевые технологии : учеб.-метод. пособие / О. А. Шведова, М. В. Шило. – Минск : БГУИР, 2015. – 144 с. : ил.

ISBN 978-985-543-170-2

Содержит материалы к практическим занятиям и указания для выполнения лабораторных работ. Каждая лабораторная работа содержит краткие теоретические сведения, перечень вопросов, которые необходимо изучить в процессе подготовки к лабораторной работе, порядок выполнения, содержание отчета, контрольные вопросы. Приведена практическая информация о работе в среде программирования PC WORX *Express* и использовании комплекта ILC 130.

УДК 004.31-022.53(076)
ББК 32.973.202я73

ISBN 978-985-543-170-2

Содержание

Предисловие

Использование контроллеров существенно повышает уровень автоматизации процессов управления в технических системах. Функциональная гибкость, высокая надежность, малые габариты и стоимость микропроцессорных средств обусловили целесообразность их применения в различной аппаратуре, в том числе в системах локальной автоматики. В связи с этим существенно изменился процесс проектирования цифровых систем управления. Это вызвано как использованием более сложных функциональных компонентов, так и применением новых архитектурных решений, основанных на замене некоторых аппаратных средств программными модулями. Поэтому проектирование современных средств автоматизации требует от разработчика знания вычислительной техники и программирования на качественно новом уровне, с учетом специфики объектов управления в технических системах.

В то же время при проектировании цифровых систем управления на базе микроЭВМ разработчику приходится решать задачи, многие из которых возникают и при проектировании классической вычислительной техники. В учебно-методическом пособии рассматриваются архитектура, принципы функционирования и обработки информации в микропроцессорных системах управления. Приводятся базовые сведения о построении подсистемы памяти, организации ввода/вывода информации и системе команд контроллера Phoenix Contact ILC 130 ETH. Поскольку происходящее сегодня быстрое обновление технических средств делает нецелесообразным детальное описание конкретных устройств, в данном учебно-методическом пособии основной упор сделан на изучении базовых принципов построения микропроцессорных систем управления.

Материалы к практическим занятиям

Практическое занятие №1. Архитектура программного обеспечения и аппаратная структура

Цель: изучение учебного комплекта ILC 130 Starterkit, знакомство со средой программирования PC WORX.

1.1 Описание оборудования

Стартовый комплект ILC 130 – удобный способ ознакомления с устройствами управления Phoenix Contact, который обеспечивает возможность создания простой системы автоматизации.

С помощью интеллектуального компактного контроллера ILC 130 ETH можно самостоятельно и недорого автоматизировать малые и средние установки. Для ознакомления с программным обеспечением *PC WORX Express* используется удобная обучающая программа-пример, к которой пользователь может самостоятельно добавить недостающие модули расширения.

Стартовый комплект ILC 130 поставляется в подготовленном к работе состоянии. Все необходимые для работы компоненты закреплены на монтажной рейке и готовы к эксплуатации. В комплект входят компактный контроллер ILC 130 ETH, модуль аналогового ввода, блок питания, потенциометр и блок переключателей для выбора режимов работы. На компакт-диске с демонстрационной версией ПО Automation Software Suite, также содержится среда разработки PC WORX *Express*.
На рисунке 1.1 представлен внешний вид комплекта *ILC 130 Starterkit*, а его состав – на рисунке 1.2.

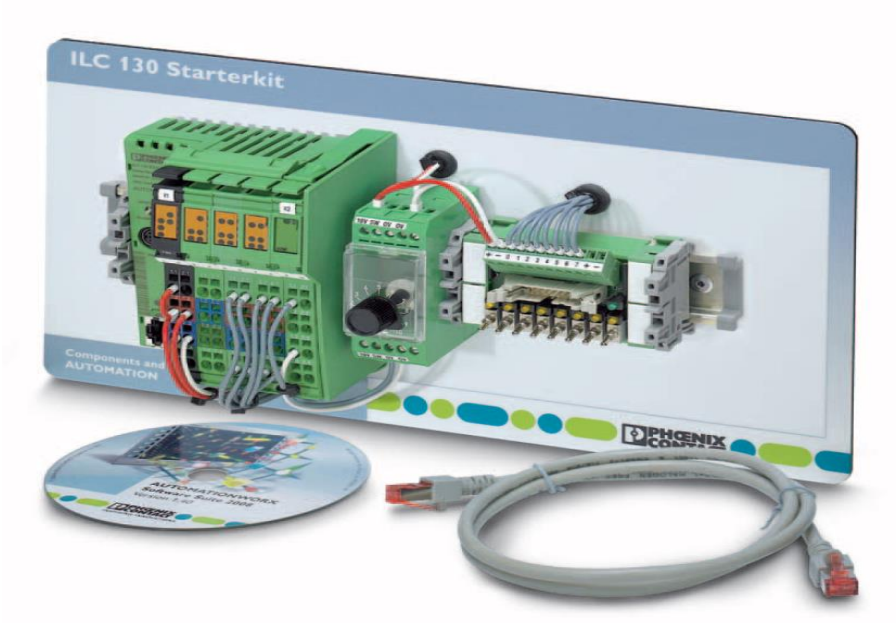


Рисунок 1.1 – Стартовый комплект ILC 130 Starterkit

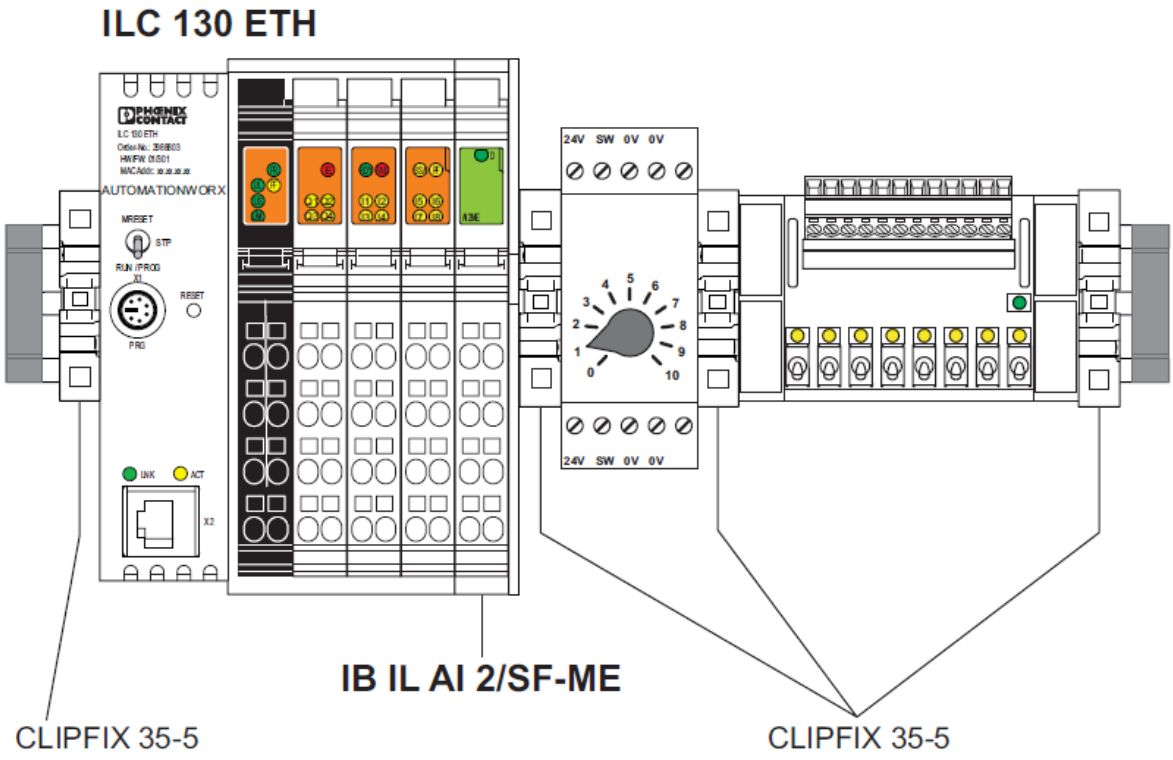


Рисунок 1.2 – Состав комплекта ILC 130 Starterkit

При сборке рабочей станции модули системы INTERBUS Inline защелкиваются в ряд подобно обычным шинным клеммам. При этом периферийные модули автоматически соединяются между собой

системной шиной, создавая общую структуру всех электрических связей рабочей станции.

Модернизация такой системы производится добавлением или изъятием определенных модулей. Сервисное обслуживание наглядно и позволяет поддерживать систему управления в рабочем состоянии. К модулям INTERBUS Inline необходимо подключить периферийные линии от датчиков и исполнительных устройств: все периферийные устройства подключаются по 2-, 3- или 4-проводной схемам с помощью разъемов COMBICON с пружинными клеммами. Разъемы можно снять с модулей, отключая периферию без оказания влияния на системную структуру INTERBUS. После этого можно быстро и аккуратно заменить модуль или произвести сервисные работы. В распределенных системах точки ввода и вывода данных иногда сильно рассеяны по объекту управления. При параллельном соединении всех точек ввода − вывода чрезмерно много средств затрачивается на кабели и их прокладку. Но есть простое и экономичное решение. Промышленный мультиплексор передает поток данных по 2-проводной шине. Пользователю не надо вдаваться в подробности теории локальных сетей, все возложено на мультиплексор.

Дополнительная справочная информация по рабочим характеристикам компактных устройств управления приведена в Приложении А.

Система собирается по следующей схеме:

1) на объекте устанавливаются датчики и исполнительные устройства;
2) выполняется их подключение к интерфейсным клеммам-модулям INTERBUS Inline;
3) клеммы-модули собирается на рейке в ряд вместе с мультиплексором;
4) получившаяся станция связывается с аналогичной станцией по 2-проводной линии;
5) включается питание (24 В).

1.2 Архитектура программного обеспечения. Рабочие пространства

Интерфейс PC WORX состоит из четырех свободно конфигурируемых рабочих пространств, отображающих подключаемые окна, которые могут быть при необходимости показаны либо скрыты. Каждым из четырех рабочих пространств учитывается доступ к основным функциям PC WORX: конфигурация шины, программирование и процесс редактирование данных, сравнение проектов.

Примечание – Даже при том, что рабочие пространства свободно конфигурируемы, их необходимо организовывать эффективно, не перегружая открытыми окнами и изменяя их согласно текущей задаче приложения.

Настройки рабочих пространств могут быть сброшены через *Extras → Options Menu → General*.

Строка меню обеспечивает быстрое переключение между различными областями. Так же выбор рабочего пространства и рабочих окон можно сделать через вкладку View в главном меню (рисунок 1.3).

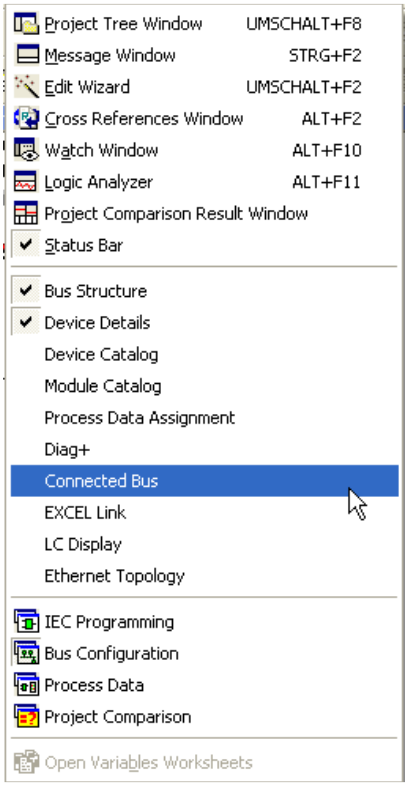


Рисунок 1.3 – Окно вкладки View

IEC programming (Программирование IEC)

Рабочее пространство *IEC programming* (функции, функциональные блоки и программы) используются для создания аппаратных средств PLC, объявления определенных пользователем типов данных, интегрирования библиотек и программно связанных функций (рисунок 1.4).

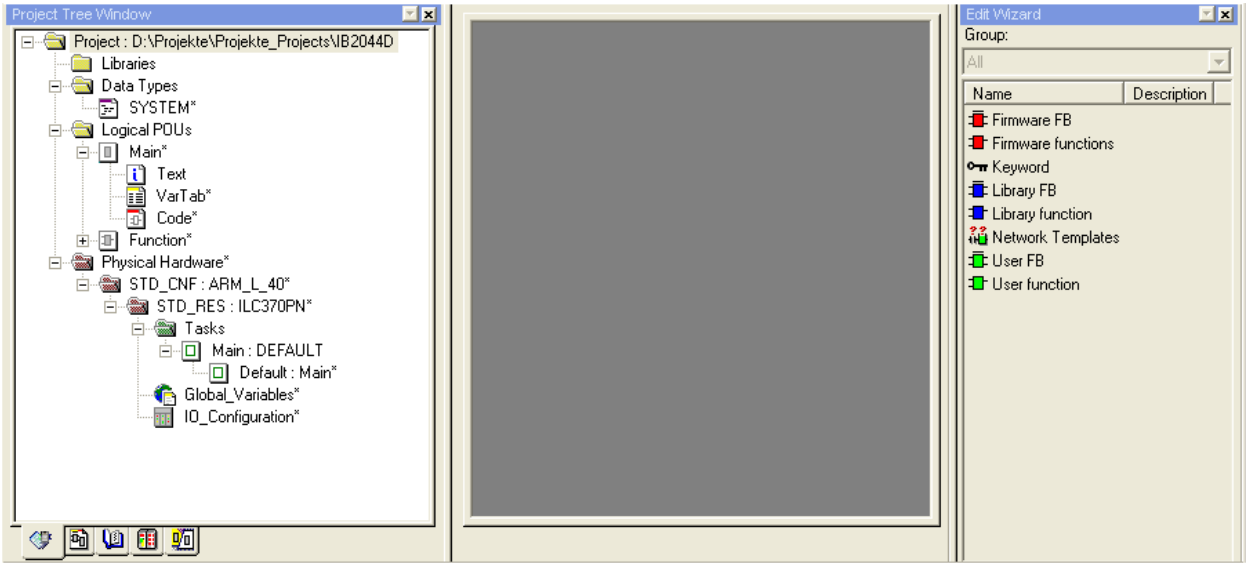


Рисунок 1.4 – Рабочее пространство программирования IEC

Рабочее пространство для программирования разделено на *Project tree* (дерево проекта) и *Edit wizard* (мастер редактирования).

Помимо полного отображения проекта, в окне *Project tree* даются определенные представления об организационных модулях программы (POU), библиотеках, аппаратных средствах и задачах, а также структурах функций вызова программ (запросах).

Edit wizard контекстно-зависим. Это означает, что в зависимости от работающего элемента, *Edit wizard* предоставляет помощь для создания определенных пользователем типов данных или создания программы.

Рабочее пространство, окрашенное серым цветом на скриншоте, используется для открытия окон с кодом программ (окна с кодом программ и таблицы переменных). Так же, как во многих других приложениях, эти окна могут быть расположены в различных стилях (см. *Window menu*).

Для того чтобы максимизировать рабочее пространство, имеет смысл скрывать окна *Project tree*, *Edit wizard* и окно сообщений, которое автоматически появляется во время процессов компиляции.

Сочетание клавиш быстрого доступа:

– *Shift+F8* – показывает/скрывает окно *Project tree*;
– *Ctrl+F2* – показывает/скрывает окно сообщения;
– *Shift+F2* – показывает/скрывает окно *Edit wizard*.

Примечание – Быстрый доступ может быть скорректирован через *Tools → Shortcuts menu*.

Bus configuration (Конфигурация шины)

Рабочее пространство *Bus configuration* используется для создания магистральных систем через INTERBUS и PROFINET, поддерживаемых контроллером, а также редактирования используемых устройств и общего управления их данными.

В проекте, прежде всего, осуществляется конфигурация аппаратных средств в рабочем пространстве *Bus configuration* (рисунок 1.5). Стандартно открыты следующие окна: *Bus configuration* (конфигурация шины), *Device details* (детали устройства) и *Device catalog* (каталог устройств). Окно, которое было открыто последним из представленных, выдвинуто на первый план, поскольку оно необходимо для редактирования.

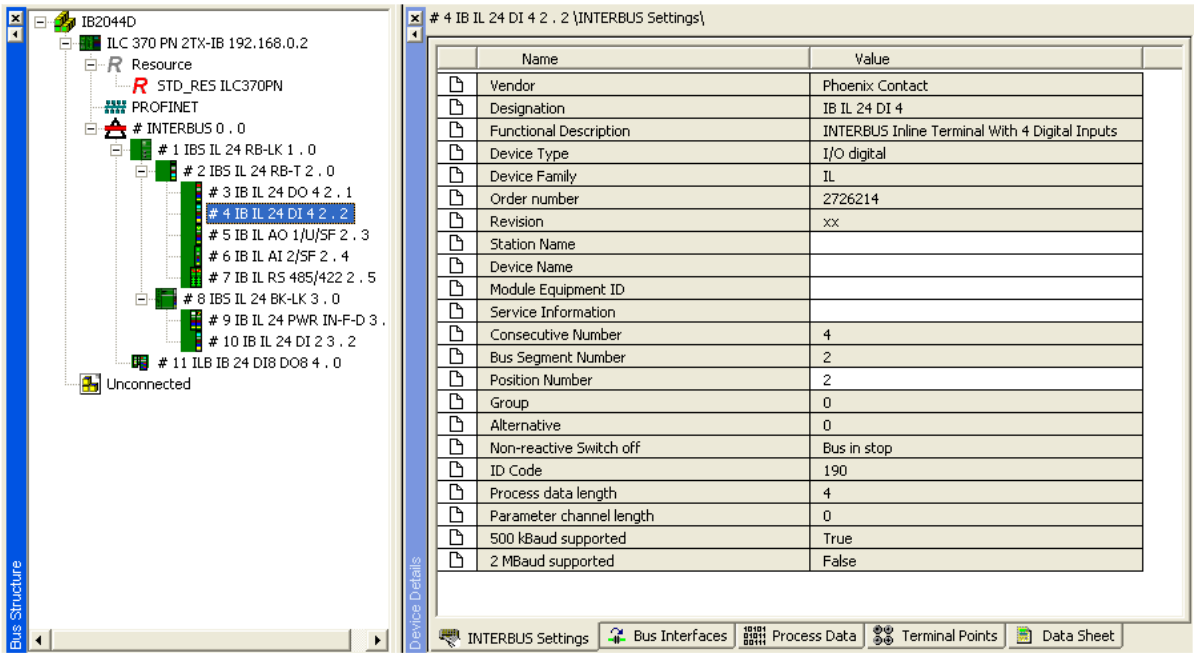


Рисунок 1.5 – Рабочее пространство конфигурации шины

Детали устройства непосредственно связаны с конфигурацией шины. Выбранный в *Bus configuration* элемент отображается со своими деталями в *Device details*. В зависимости от типа элемента (проект, каталог шин, устройство шины) число и типы вкладок в этом окне корректируются. Кратковременная задержка отображения информации объясняется тем, что информация считывается из устройства или системного файла. «Преобразованный» текстовый файл показывается в легко доступной для пользователя форме в *Device details*.

После конфигурации магистральной системы полная документация (идентификация оборудования, имя станции и т. д.) для аппаратных средств может быть введена в *Device details*.

Process Data Assignment (Данные процесса)

Рабочее пространство *Process Data Assignment* используется для соединения элементов данных процесса с глобальными переменными программ, которое стало доступным благодаря модулям шин, или создания глобальных переменных, базирующихся на ранее указанных объектах данных процесса, согласно стандарту синтаксиса.

Рабочее пространство *Process Data Assignment* (рисунок 1.6) разделено на четыре сектора. Левая сторона показывает PLC и его структуру. В зависимости от выбранного элемента соответствующие глобальные переменные выведены на экран в таблице ниже.

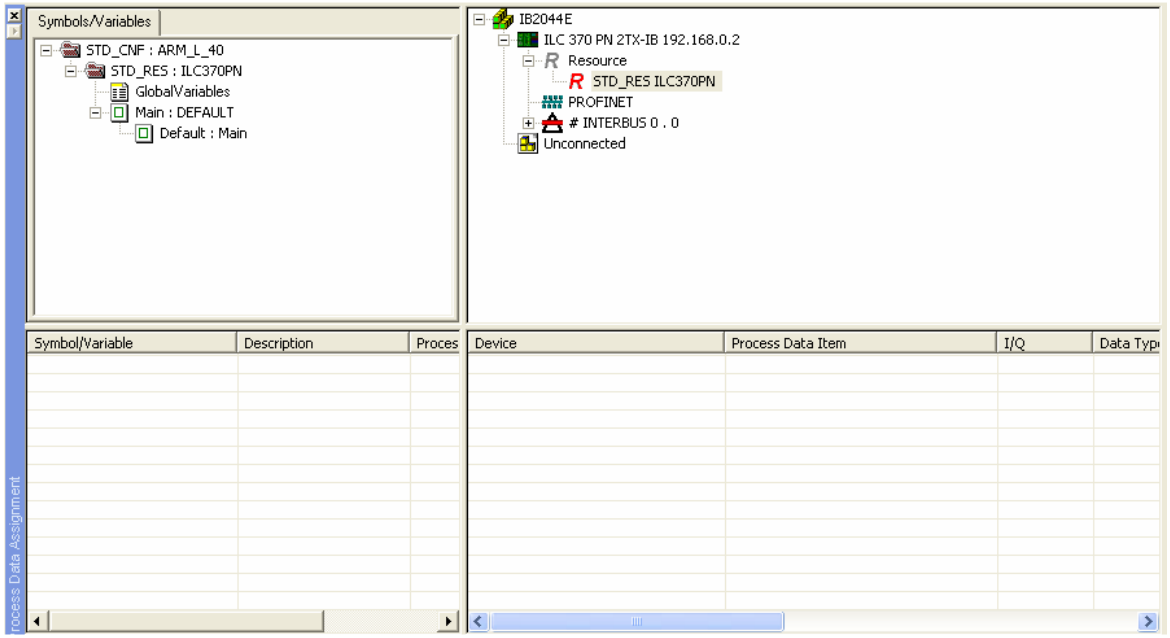


Рисунок 1.6 – Рабочее пространство данных процесса

Правая сторона показывает аппаратные средства магистральной системы. В зависимости от выбранного элемента здесь показаны соответствующие элементы данных процесса:

- для единичных устройств: только объекты отобранного устройства;
- для магистральных терминалов: все элементы данных процесса, доступные в соответствующем отделении;
- для систем управления: все объекты, зависящие от этой системы управления.

В этом окне программные переменные сопоставлены с элементами объекта данных процесса в соединенных магистральных системах.

Project Comparison (Сравнение проектов)

Рабочее пространство *Project Comparison* (рисунок 1.7) используется для наблюдения различий через сравнение двух проектов в режиме offline.

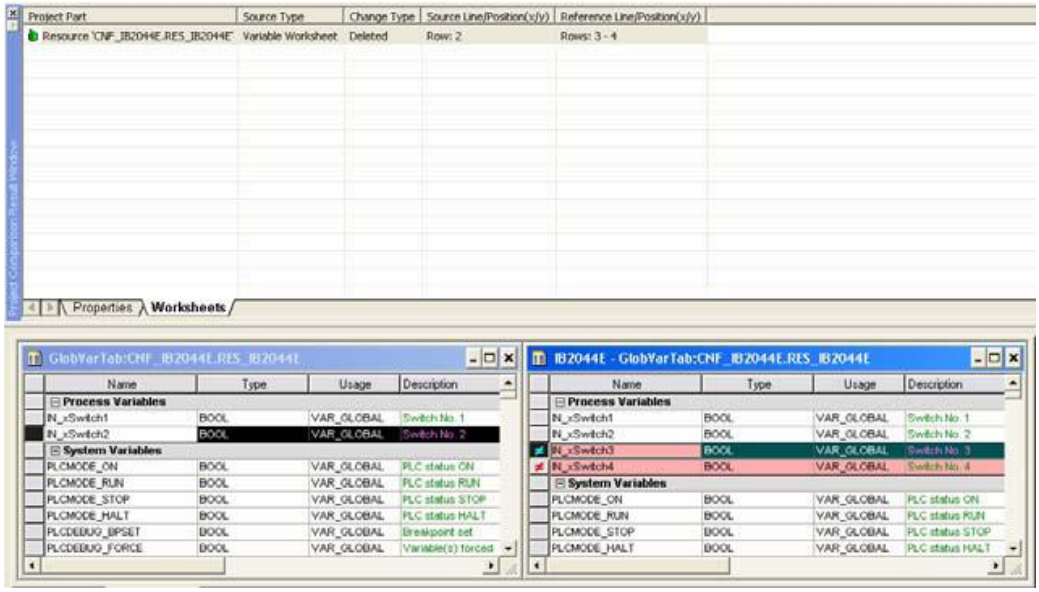


Рисунок 1.7 – Рабочее пространство сравнения проектов

1.3 Рабочие окна

Пользователь обычно использует окно *Connected Bus* (Шина соединения) только для ввода данных в систему соединения INTERBUS (рисунок 1.8). Канал связи этого окна взят от центральных параметров настройки и может быть вызван через *Selected Control System list* (Список выбора системы управления).

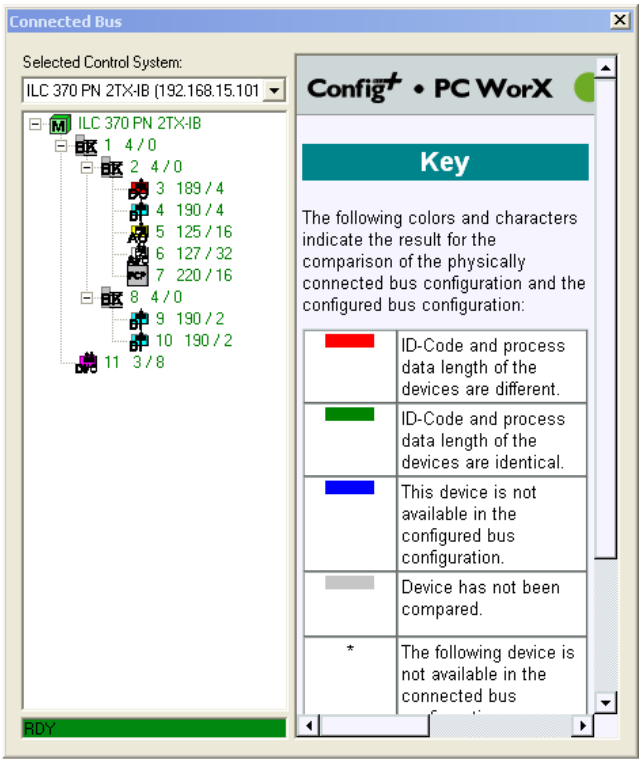


Рисунок 1.8 – Соединительная шина данных

Обозначения:

1) *Красный цвет* – идентификационный код и длина данных о процессе устройств отличаются;
2) *Зеленый цвет* – идентификационный код и длина данных о процессе устройств идентичны;
3) *Синий цвет* – устройство не доступно в формируемой конфигурации шины;
4) *Серый цвет* – устройство не было сравнено;
5) * – следующее устройство не доступно в текущей конфигурации шины;
6) * – следующий уровень шин не доступен в текущей конфигурации шины;
7) X/Y – идентификационный код и длина данных процесса.

Окно *Device Catalog* (каталог устройств) требуется для конфигурации магистральных систем в режиме offline и должно быть показано только во время этой операционной фазы (рисунок 1.9). Оно также необходимо для исправлений устройств, неправильно подобранных во время конфигурации в режиме online.

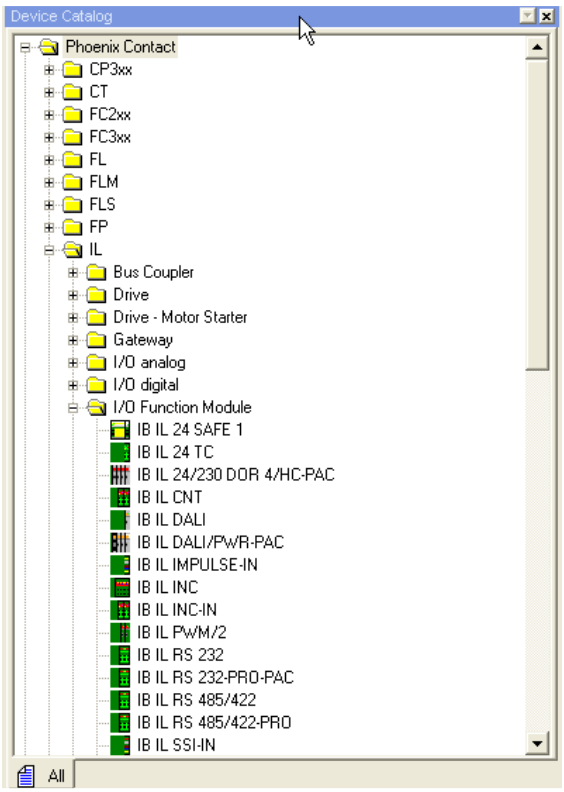


Рисунок 1.9 – Каталог устройств

Все файлы описания устройств перечислены на вкладке *Device Catalog* в закладке *All*. Через контекстное меню пользователь может настроить структуру дерева проекта.

1.4 Архитектура аппаратных средств

Операционная система *ProConOS* (*Programmable Controller Operating System*) требуется для обеспечения системных услуг PLCtypical на специальных или стандартных аппаратных платформах. Они включают загрузку и обработку внешне создаваемых программ PLC, а также обеспечение функций отладки для программирования, установку и обслуживание управляемых PLC машин и систем.

ProConOS используется в качестве операционной системы для большинства систем управления Phoenix Contact. Ее общая структура архитектуры аппаратных средств отображена на рисунке 1.10.

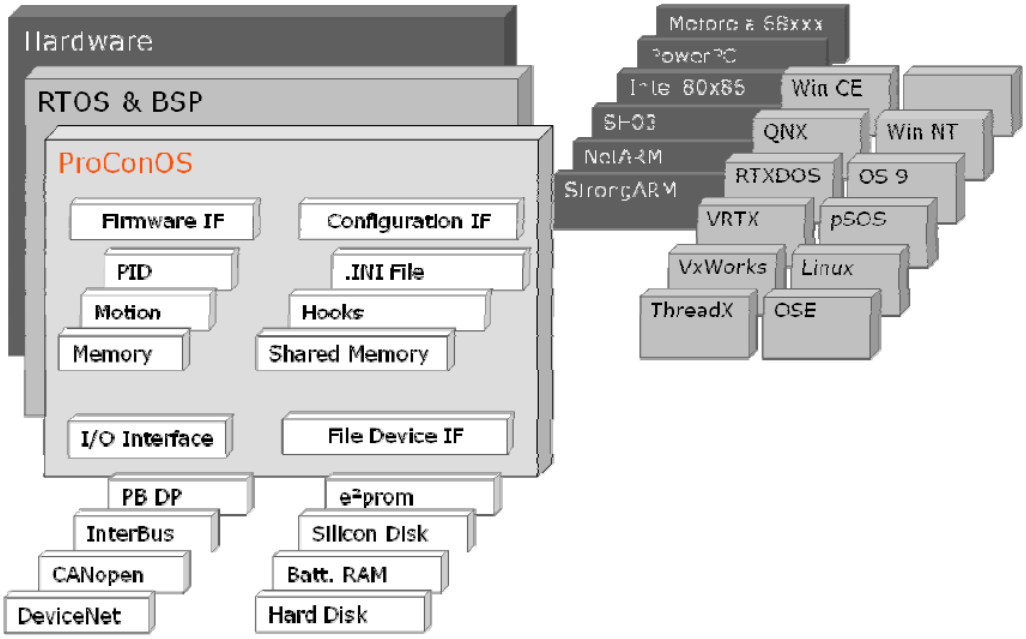


Рисунок 1.10 – Общая структура архитектуры аппаратных средств

Следовательно, *преимуществом* является то, что все системы управления могут быть параметризованы и запрограммированы в одной и той же среде разработки PC WORX.

ProConOS основана на стандартных многозадачных операционных системах, которыми управляют посредством приоритетов задачи. В ProConOS есть три различных уровня приоритетов:

1) приоритетный уровень для пользовательских задач;
2) приоритетный уровень для задач ProConOS;
3) приоритетный уровень для задач администратора системы.

Эта классификация гарантирует, что процессор всегда распределяет вычислительное время, полностью доступное PLC, в пользу важных для приложения задач. Если синхронизация позволяет, то другие системные задачи получают необходимое для решения время. Это ProConOS внутренняя предпосылка для временного детерминизма приложения. Использование многозадачности в архитектуре системы ProConOS (рисунок 1.11) таким образом стремится к измеримому ответу времени, оптимизации производительности, т. е. уменьшению времени отклика и определенной реакции на ошибки периода выполнения.

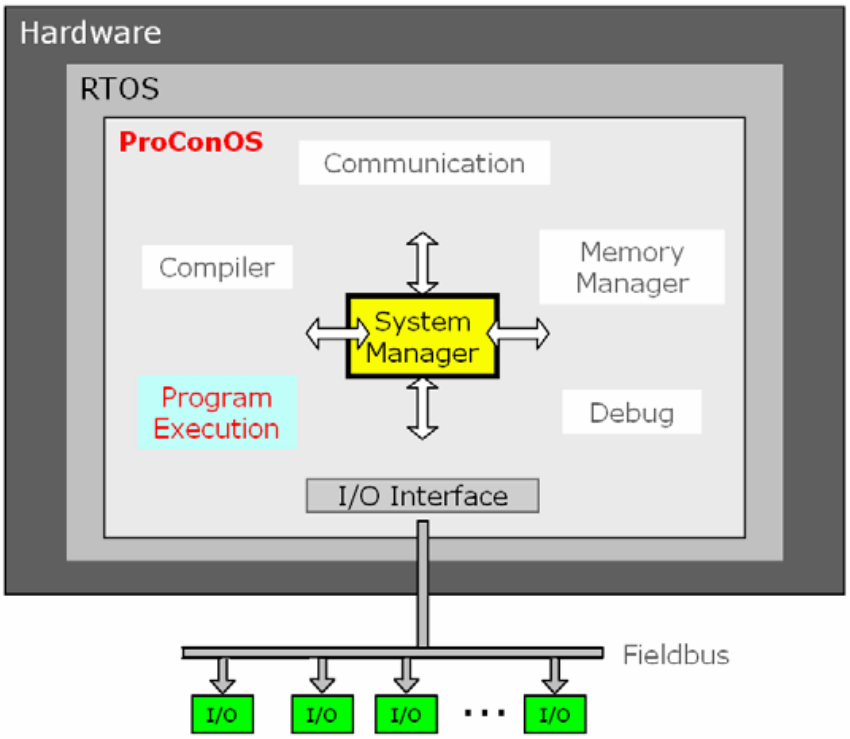


Рисунок 1.11 – Многозадачность

Пользовательские задачи включают циклические задачи, задачи события и задачи по умолчанию. В пределах установленного временного интервала циклические задачи выполняются циклически согласно приоритету, определенному пользователем. Кроме того, реакция на не циклически происходящее событие, например, аппаратное прерывание, может воздействовать на конечный результат решения задачи.

Задача по умолчанию – это задача с самым низким приоритетом, она выполняется, когда остальные задачи не активны и является циклической.

Во время адаптации ProConOS на различные платформы управления Phoenix Contact использовались различные интерфейсы, чтобы реализовать специфичные для управления опции.

1.5 Система управления. Последовательный интерфейс каналов связи

Для сетевой коммуникации представлена передача IP-адресов к системам управления, например, использование служб BootP.

Канал связи для соединения устройства программирования и системы управления выбирается на вкладке *Communication* (соединение),

когда выбран элемент *Control System* (система управления). В зависимости от типа системы управления доступны различные носители.

Каждая система управления обеспечивает последовательный интерфейс (рисунок 1.12), для которого могут использоваться стандартный соединительный кабель IBS или кабель с соединителем Mini-DIN.

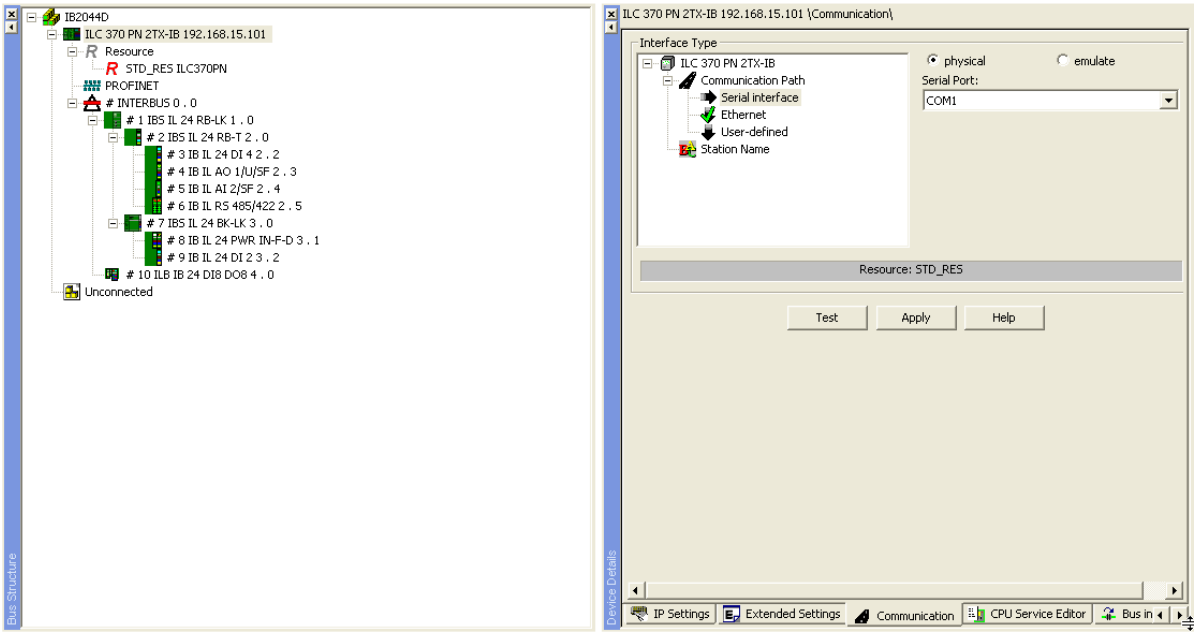


Рисунок 1.12 – Последовательный интерфейс каналов связи

В случае если устройство программирования не обеспечивает соединение через последовательный интерфейс и должен использоваться адаптер USB, необходимо выбрать эмулированный интерфейс.

Нажатие кнопки *Test* активирует установление соединения, которое выведено на экран в строке состояния.

Через кнопку *Apply* можно определить канал связи для текущей системы управления в момент работы проекта. Используя эти настройки, другие окна могут все время получать доступ к системе управления.

Примечание – При использовании ранее сконфигурированных систем управления, сообщение *Connection names* (имена подключений) не может высветиться на экране, потому что имя подключения для соединения сохранено в системе управления. Это предотвращает случайный доступ к системе управления.

Настройки сети системы управления

В дополнение к последовательному интерфейсу большинство систем управления обеспечивает возможность организации связи через TCP/IP. Для этого типа передачи IP-адрес должен быть сохранен в системе управления. Кроме того этот адрес должен быть передан в PC WORX.

Система управления с часами реального времени, IP-адресом устанавливаются на вкладке *External Settings* после того, как был выбран элемент *Control System* (рисунок 1.13). После выбора функции *Read* во фрейме *Network Settings* отобразится сетевая конфигурация системы управления (значение по умолчанию после поставки – использование BootPserver). Для этого необходимо выбрать готовый к работе канал связи, обычно это последовательный интерфейс.

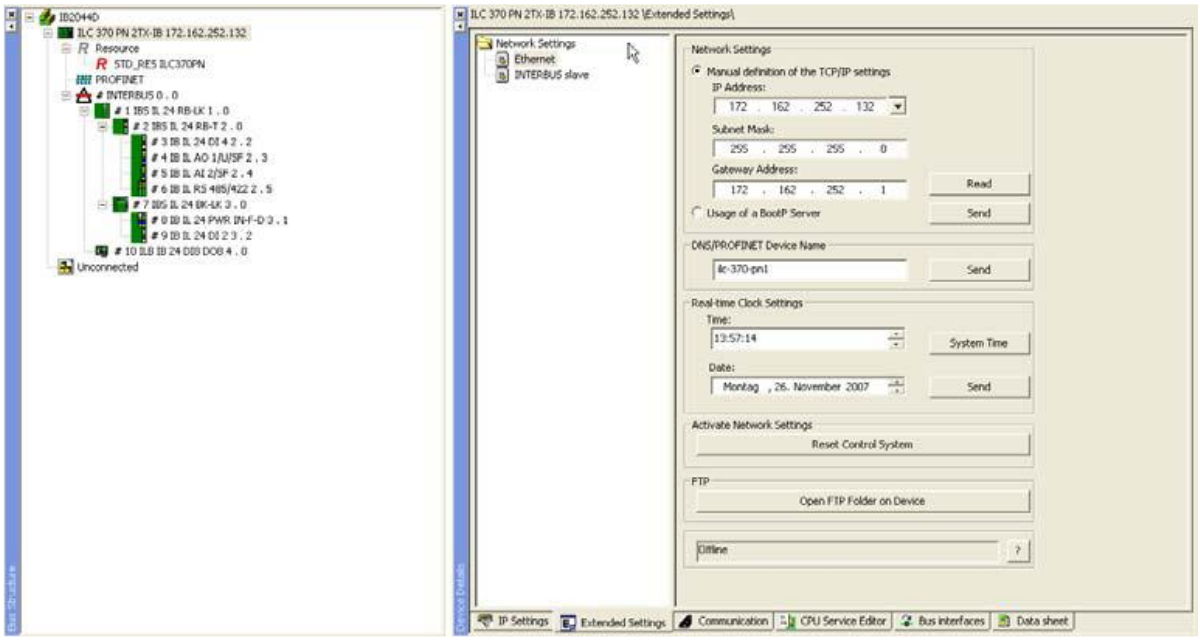


Рисунок 1.13 – Настройки сети системы управления

Связь устанавливается согласно требованиям проекта. В случае необходимости в корректировке часов реального времени системы управления, это можно сделать, вызывая и отправляя *System Time*. Проверка допустимости внесенных изменений может быть реализована через сбрасывание системы управления.

Поскольку у систем управления есть сервер FTP с успешно сконфигурированной сетевой связью, то программное обеспечение для доступа FTP может быть запущено через кнопку *Open FTP Folder on*

Device. Таким образом, контент области FTP карты CompactFlash выведен на экран.

Присвоение IP-адреса через BootP

Если после запуска сервера BootP система управления должна получить свой IP-адрес, то сервер опции Usage of BootP должен быть выбран в расширенных настройках (рисунок 1.14). С передачей нового статического IP-адреса эта конфигурация должна быть отправлена и затем проверена сбросом контроллера.

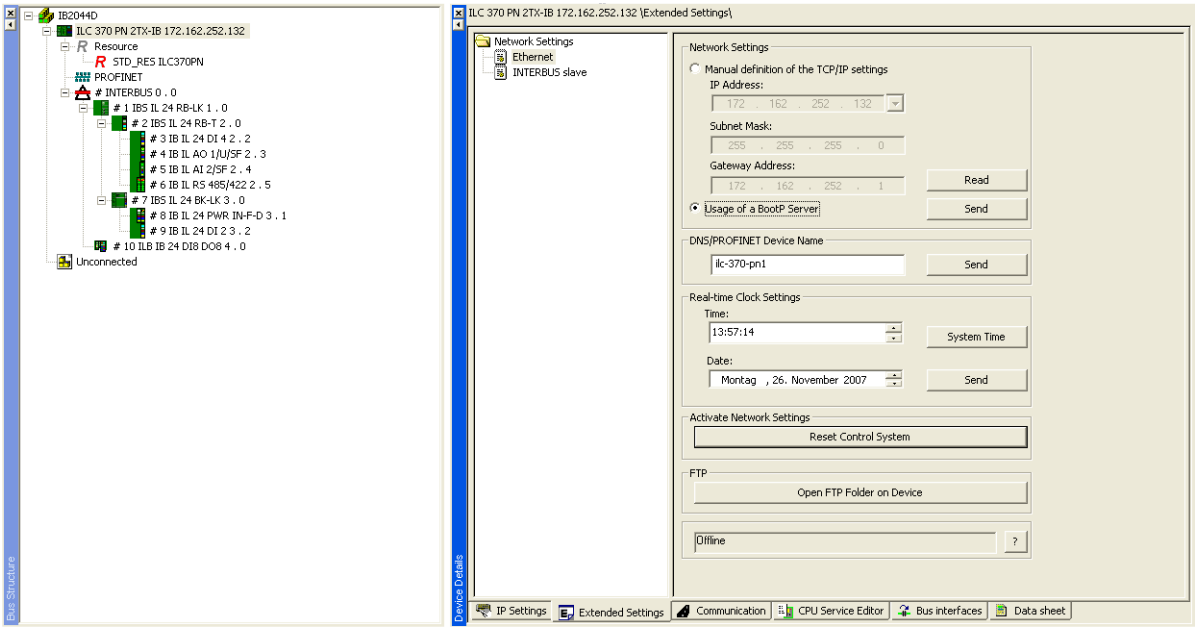


Рисунок 1.14 – Присвоение IP-адреса через BootP

Примечание – Когда используется опция BootP и система управления заменена, не достаточно только изменить карту CF, необходимо новый MAC-адрес передать серверу BootP (рисунок 1.15).

Чтобы использовать опцию BootP для системы управления посредством интегрированного в PC WORX сервера BootP, необходимо активировать сервер BootP через меню *Tools → BootP/SNMP/TFTP-Configuration*. Для присвоения MAC-адреса системе управления, он должен быть введен в *Extended Settings*.

Примечание – Вы можете найти MAC-адрес на объединенной шестнадцатеричной этикетке (этикетке штрих-кода) на передней или левой стороне корпуса.

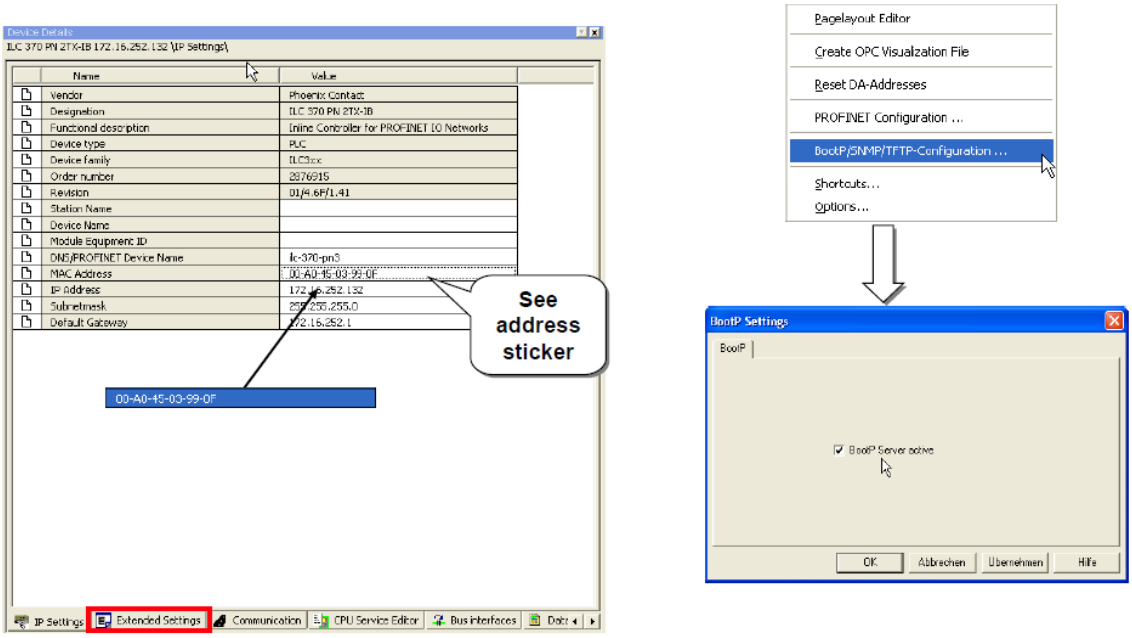


Рисунок 1.15 – Окна ввода MAC-адреса, активирующего сервер BootP

Сетевая связь. Выбор сетевого соединения

Если контроллер был сконфигурирован с допустимым IP-адресом, этот адрес необходимо сделать доступным для PC WORX (рисунок 1.16).

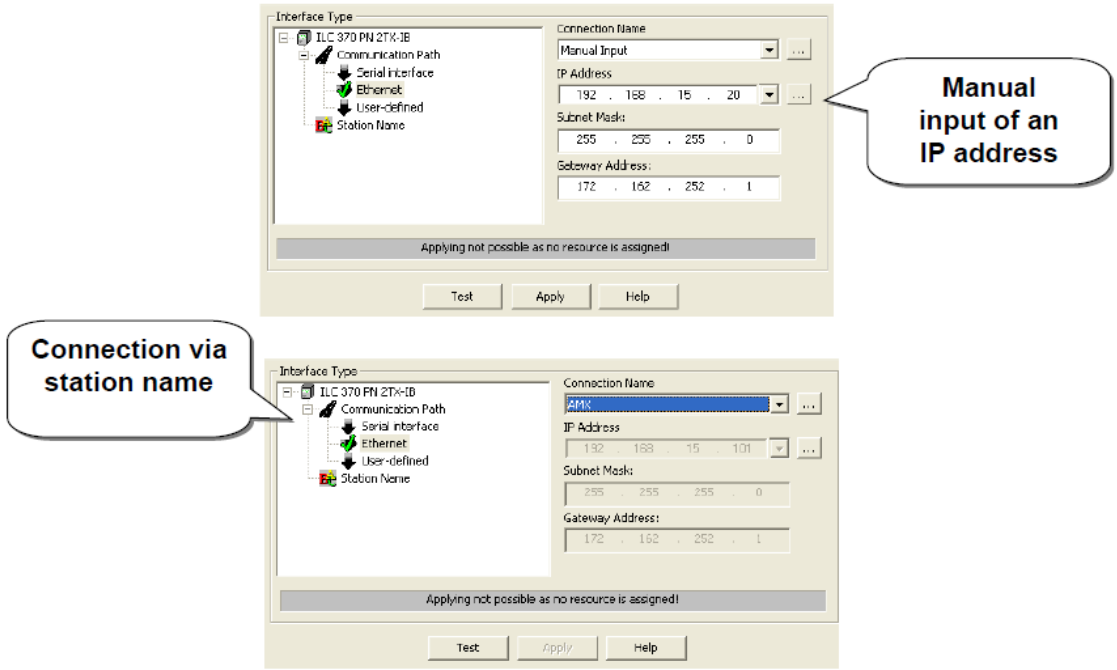


Рисунок 1.16 – Выбор сетевого соединения

Есть две возможности выбрать канал связи TCP/IP (Ethernet / Local host):

1) Выберите элемент *Manual Input* и введите требуемый IP-адрес во включенные поля.
2) Выберите станцию из списка имен подключений. Имена станций соединены с IP-адресами через файл адреса или запись в реестре.

Через кнопку **...** можно загрузить файл адреса или изменить записи имени станции. Самым легким способом создать файл адреса является изменение записи в реестре и затем экспортировка списка. Используя инструмент редактирования текста, создаваемый файл **.dat* может быть отредактирован как разделенный на табулятор текстовый файл. Записи в файле адреса или в реестре будут выведены на экран в поле *Connection Name*, если они были активированы.

Контрольные вопросы

1. Сколько рабочих пространств в PC WORX? Каким образом можно сбросить настройки рабочих пространств?
2. Каковы основные составляющие и функции рабочего пространства *IEC programming*?
3. Для чего используется рабочее пространство *Bus configuration*?
4. На какие четыре сектора разделено рабочее пространство *Process Data Assignment*?
5. Какие цвета используются в окне *Connected Bus* и для чего?
6. Какие три уровня приоритетов есть в ProConOS?
7. Объясните назначение кнопок на вкладке Communication элемента *Control System*.
8. Каким образом настраивается сеть системы управления через TCP/IP?

Задания

Задание 1. Изучить состав комплекта ILC 130 Starterkit.

Изучите компоненты комплекта ILC 130 Starterkit, обратите внимание на разъемы и индикаторы контроллера ILC 130 ETH.

Задание 2. Присвоить контроллеру IP-адрес через BootP.

Задание 3. Изучить рабочие пространства среды разработки PC WORX. Создать проект в PC WORX.

Практическое занятие №2. Конфигурирование оборудования

Цель: изучение методов конфигурации оборудования.

2.1 Конфигурация IBS

В случае если готовая к работе система соединена с системой управления, может использоваться конфигурация в режиме offline, которая описана для адаптации и конфигурации перед запуском аппаратных средств.

Для конфигурации в режиме online также требуется рабочее соединение между PC и системой управления.

Требования

Для обоих методов конфигурации требуются файлы описания устройства для устройств, используемых в проекте. По результатам установки PC WORX обеспечен широкий диапазон на PC. Дальнейшие файлы описания устройства (формат XML) могут быть зарегистрированы позже после копирования соответствующих каталогов через окно *Device Catalog*.

Конфигурация в режиме online

Для вывода на экран конфигурации в режиме online используются вкладки меню *View → Connected Bus*. Чтобы сохранить используемое в настоящий момент рабочее пространство (обычно рабочее пространство *Bus Configuration*) в доступном режиме, оно должно быть переключено в режиме offline и скрыто снова после ввода данных.

Для чтения в подключенной системе INTERBUS необходимо выбрать систему управления в *Selected Control System list*, продолжающую работать в настоящий момент. Тогда PC WORX выведет на экран фрейм конфигурации INTERBUS, сохраненный в системе управления или, в случае если система из *Ready state* используется, создаст новый фрейм конфигурации.

Если система не готова работать или находится в неисправном состоянии, то это окно не может вывести на экран фрейм конфигурации. Для диагностики может использоваться окно *Diag+*.

Если магистральная система выведена на экран (рисунок 2.1), она может быть скопирована с контекстного меню в окно *Bus Configuration* через функцию *Import to Project → With Device Description.*

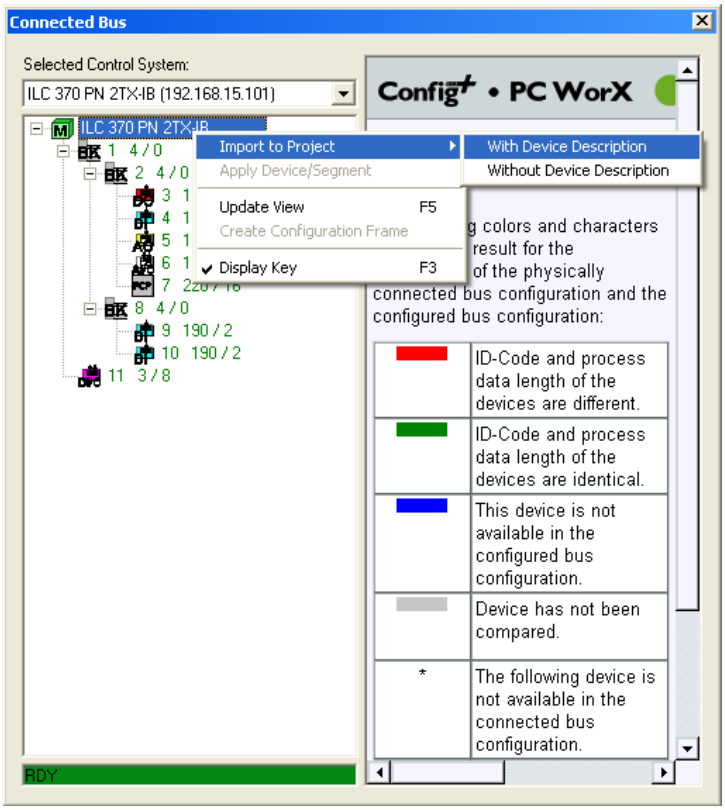


Рисунок 2.1 – Ввод данных для соединения магистральной системы

Внимание! Пожалуйста, учитывайте, что когда Вы выбираете эту функцию, то уже сконфигурированная магистральная система сбрасывается. Если необходимо применить только отдельные устройства от магистральной системы, (например, как дополнение, если конфигурация не была завершена), то должна использоваться функция *Insert Device/Segment.*

В зависимости от работающего в настоящий момент устройства, список файлов, зарегистрированных в PC WORX для описания устройства, выведется на экран в *Select Device dialog*. Кроме того, выбор также определен посредством комбинации идентификационного кода и длины данных процесса текущего устройства.

В случае если определяемые пользователем каталоги устройства были созданы в PC WORX прежде, чем выбрано устройство, необходимо выбрать каталог, к которому функция должна получить доступ во время ее поиска.

Примечание – Если бы выбор был ошибочным, то процесс выбора не должен быть прерван, а должен продолжиться с учетом ошибки. В большинстве случаев последующее исправление через каталог устройства занимает меньше времени, чем повторный ввод данных в систему.

Конфигурация в режиме offline

Для конфигурации в режиме offline должно быть активировано окно *Device Catalog* из меню представлений (рисунок 2.2). Желательно расположить это окно между *Bus configuration* и *Device Details*.

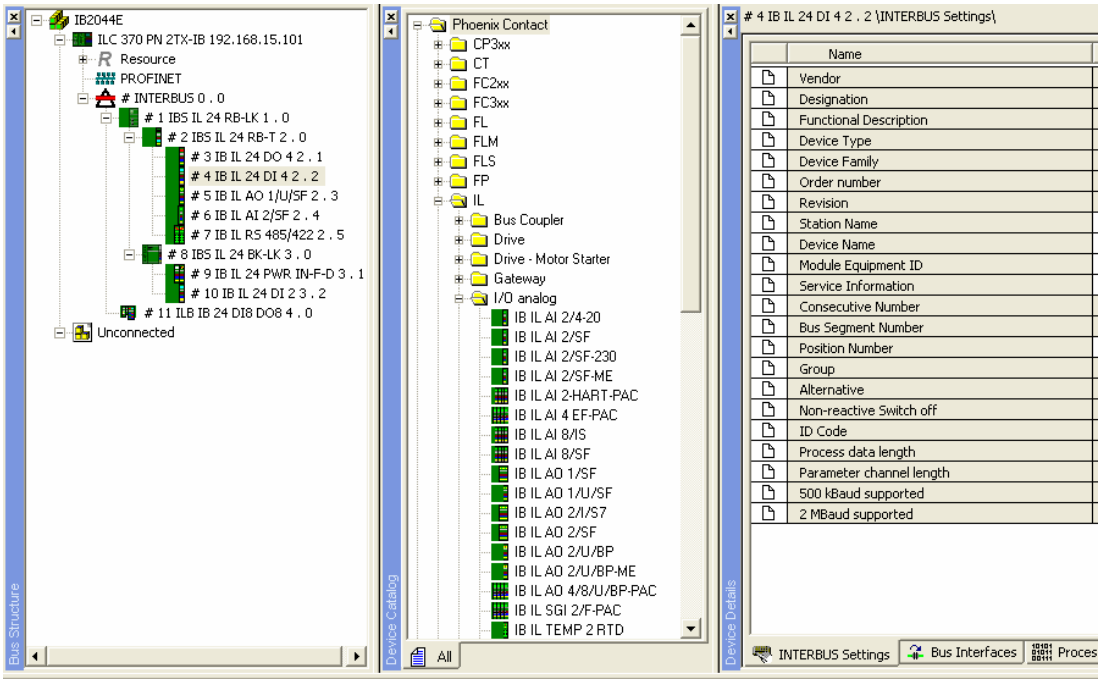


Рисунок 2.2 – Вставка устройства из Device Catalog

Таким образом, окно *Device Details* частично вытиснится из видимой области, а окна *Connected Bus* и *Device Catalog* будут отображены на экране, пока это будет необходимо. Функция каталога относительно конфигурации шины ограничена тремя действиями:

1. Вставка устройства на тот же самый уровень (удаленное устройство шины позади удаленного устройства шины или устройство локальной шины позади устройства локальной шины).

Это может быть сделано через контекстное меню (*Device Catalog* → *Copy device* → *Bus configuration* → *Insert to same level*) или с использованием мыши. Для этого устройство должно быть выбрано в каталоге (нажата левая кнопка мыши) и перемещено к устройству в конфигурации шины, после которого надо вставить новое устройство. Указатель мыши выводит на экран ниже символ при вставке на тот же самый уровень  .

2. Вставка устройства в более низкий уровень (ответвление) (удаленное устройство шины в удаленной шине, устройство

ответвительной или локальной шины позади модуля терминала локальной шины).

Это также может быть сделано через контекстное меню (*Device Catalog → Copy device → Bus configuration → Insert to lower level*) или с использованием мыши. Выберите требуемое устройство в каталоге и потяните его с помощью нажатой кнопки мыши в *Bus configuration*. В отличие от вставки к тому же самому уровню, на устройстве, после которого надо вставить новое, мышь должна быть перемещена в правую сторону, пока символ ниже не выведен на экран. Символ при вставке в более низкий уровень будет иметь вид  .

Примечание – При перемещении мыши с нажатой кнопкой сдвига, Вы можете переключиться между этими двумя возможностями – вставка устройства на тот же самый уровень и вставка устройства на более низкий уровень.

3. Замена устройства (возможна, если устройство предлагает аналогичные интерфейсы). Прежде всего, заменяющее устройство должно быть скопировано в *Device Catalog* через контекстное меню, тогда функция замены контекстного меню может быть реализована.

Используя мышь и клавиатуру, требуемое устройство необходимо выбрать в *Device Catalog* и, удерживая кнопку мыши в нажатом состоянии, переместить к устройству, которое будет заменено в конфигурации шины.

Для того чтобы использовать замену вместо вставки, должна быть нажата та же самая кнопка управления так, чтобы ниже указателя мыши отобразился символ замены  .

После установки программного обеспечения каталог имеет древовидную структуру следующего вида: Поставщик – Семейство устройства – Тип устройства. Эта сортировка оказалась практичной для всего каталога, но может быть скорректирована через функцию *Modify Catalog* в контекстном меню. Обхождение без любой структуры могло бы иметь смысл для специально определяемых пользователем каталогов с узким диапазоном устройств.

Создание пользовательского каталога устройств имеет смысл, если должно быть реализовано множество проектов с использованием ограниченного количества устройств. Новый каталог может быть создан через контекстное меню окна *Device Catalog* (каталог устройств). Этот каталог первый пустой и неструктурированный.

Новый каталог может быть заполнен, используя контекстное меню для копирования устройства из основного каталога и вставки в этот новый каталог (рисунок 2.3). Альтернативно это может быть сделано через перетаскивание и удаление, как показано выше.

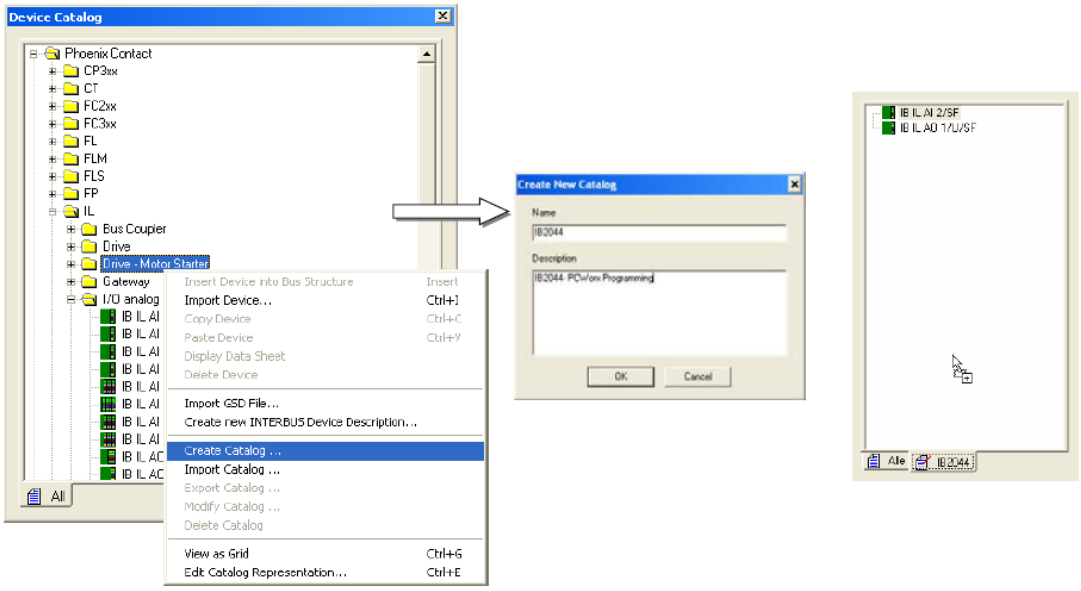


Рисунок 2.3 – Создание пользовательского каталога

Определяемый пользователем каталог включает более быстрый доступ во время конфигурации в режиме offline и online. Для последнего режима только ограниченное количество устройств будет выведено на экран при вставке.

Примечание – Каждый определяемый пользователем каталог устройств может содержать только устройства, которые находятся в основном каталоге. Нет никакой дифференцируемой регистрации описаний устройства.

2.2 Конфигурация PN

Конфигурация в режиме online описывает конфигурацию для случая, когда установлена система PROFINET. Для адаптации и конфигурации перед запуском аппаратных средств описана конфигурация в режиме offline.

Требования. Для обоих методов конфигурации требуются файлы описания устройства для устройств, используемых в проекте.

Определенный диапазон уже обеспечен на PC по результатам установки PC WORX. Дальнейшие файлы описания устройства (формат XML) могут быть зарегистрированы позже, после копирования соответствующих каталогов через окно *Device Catalog*.

Диапазон адресов сети PROFINET в проекте PC WORX создается через элемент проекта в окне *Bus Configuration* (рисунок 2.4). Для каждого добавленного устройства PROFINET IP-адрес автоматически импортируется из этого диапазона адресов в конфигурации шины.

Внимание! Для того чтобы ввести диапазон IP-адреса, начальный адрес должен быть не больше, чем конечный адрес.

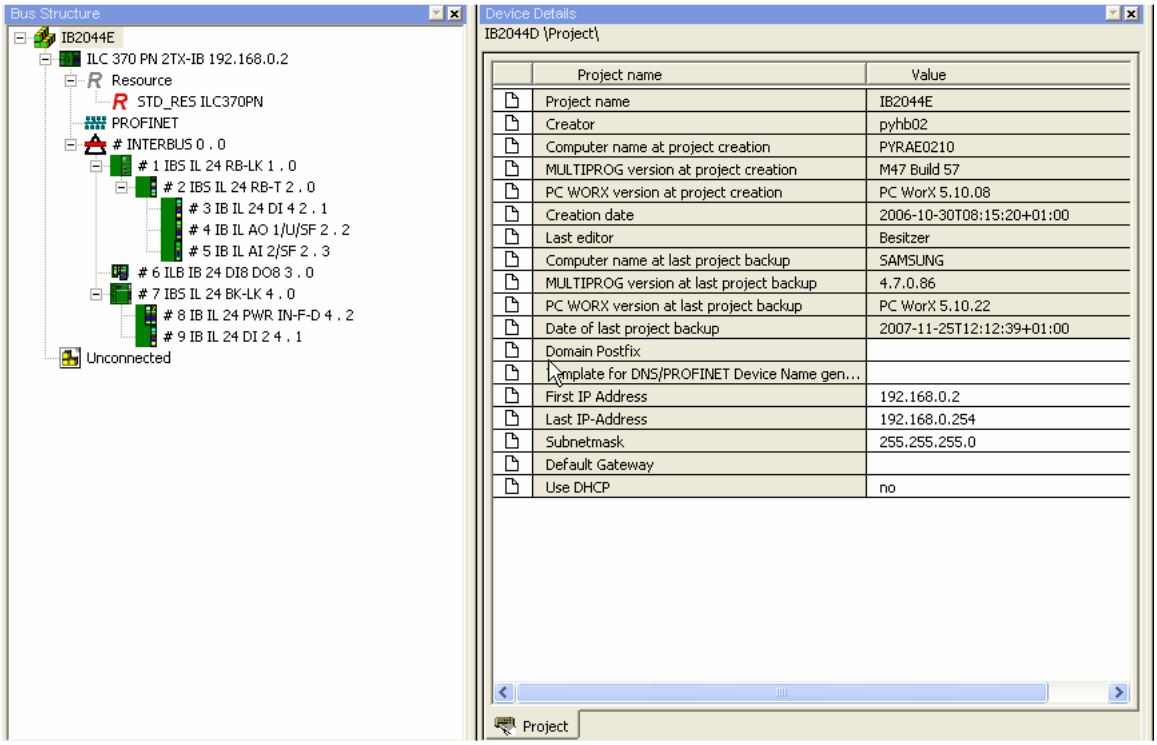


Рисунок 2.4 – Установка адресного пространства

Через контекстное меню проекта можно центрально адаптировать IP-адреса в конфигурации шины к адресному пространству набора.

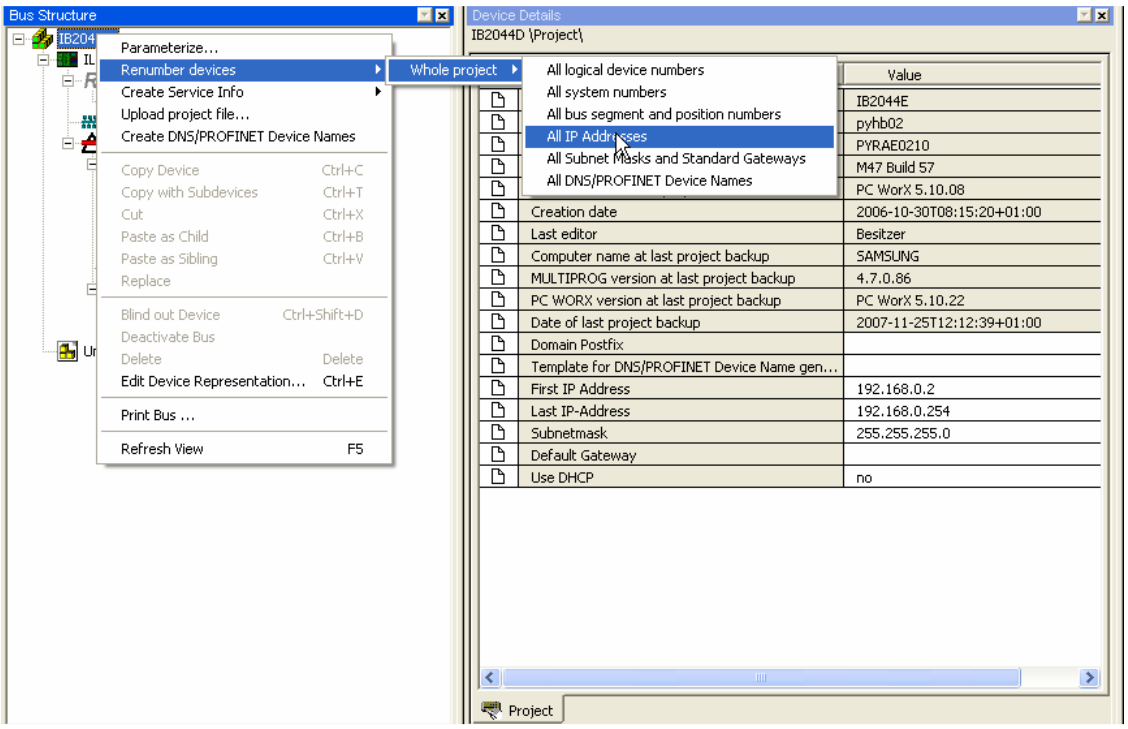


Рисунок 2.5 – Импорт адресов

Внимание! Если сетевая маска или стандартный шлюз изменены, то должны быть осуществлены следующие команды для устройства *Renumber devices* → *Whole project* → *All Subnetmasks* и *Standard Gateways* (рисунок 2.5).

Конфигурация в режиме offline/online

В конфигурации в режиме offline из *Device catalog* устройства PROFINET могут быть добавлены к сетевой конфигурации. Это может быть сделано, используя перетаскивание и удаление или копируя и вставляя через контекстном меню.

Сеть PROFINET также может быть сконфигурирована в режиме online, используя PC WORX, если устройства, которые будут сконфигурированы, установлены и их файлы описания устройства зарегистрированы в PC WORX (рисунок 2.6).

Примечание – Если никакие устройства PROFINET не найдены, хотя система PROFINET соединена, то возможно была произведена неправильная установка сетевой платы в конфигурации PROFINET.

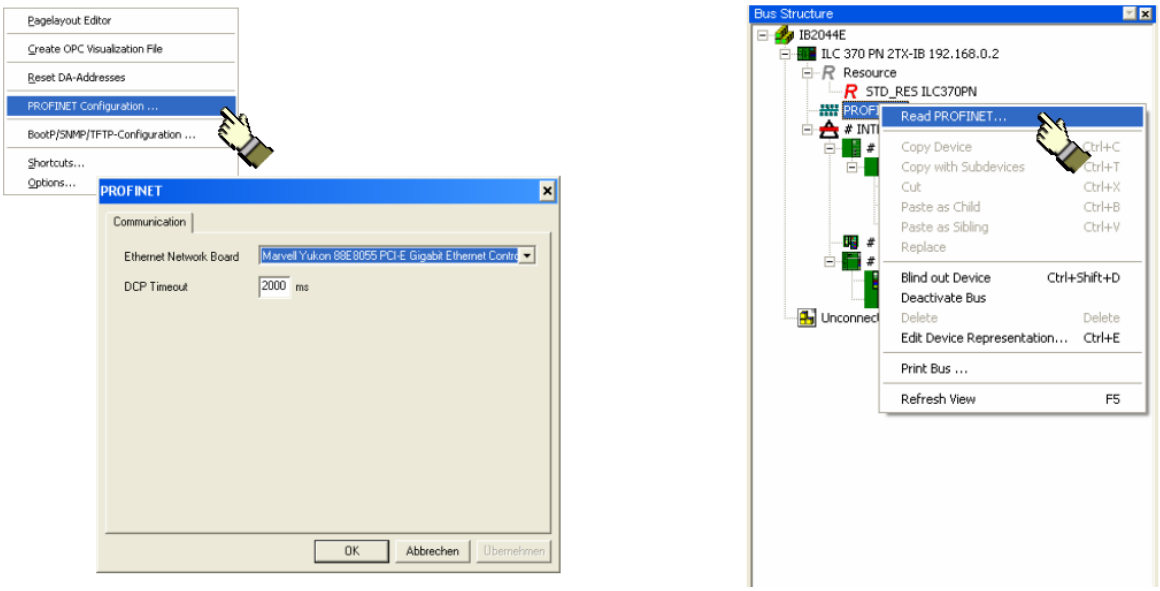


Рисунок 2.6 – Конфигурация в режиме online

Вверху списка отображаются все устройства ввода/вывода PROFINET, доступные в сети (рисунок 2.7). Если некоторые соединенные устройства не представлены в списке, то это могло произойти из-за опций фильтра ниже окна.

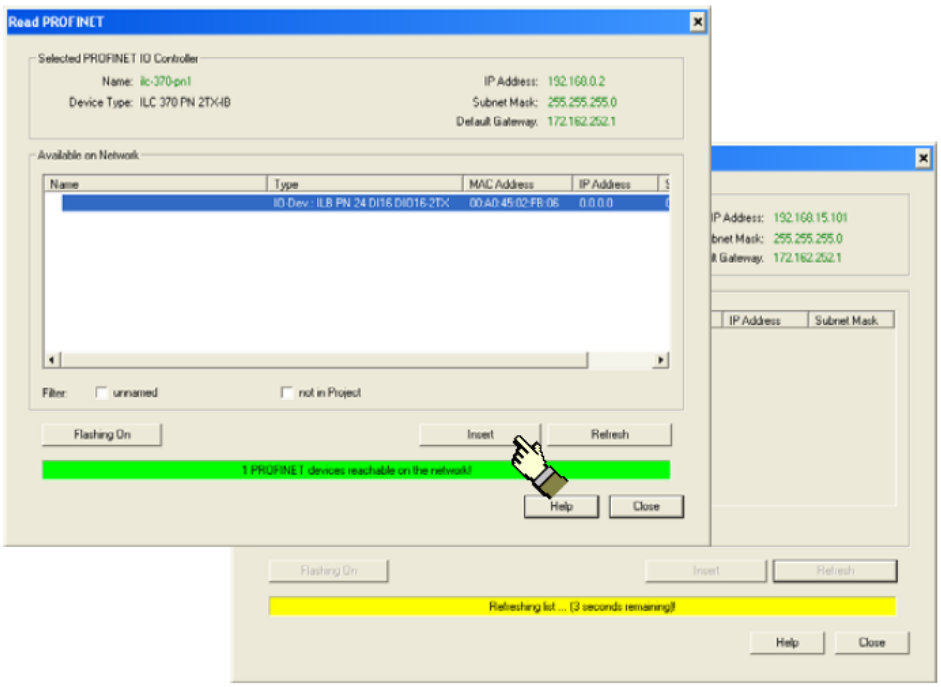


Рисунок 2.7 – Окно отображения допустимых устройств

Вставка устройств в проект

Выбранное устройство ввода – вывода вставлено в конфигурацию шины PC WORX через кнопку *Insert* (рисунок 2.8).

В другом диалоговом окне имя устройства PROFINET может непосредственно быть присвоено устройству ввода – вывода. Во-первых, имя устройства вводится только в конфигурации шины. Фактическое присвоение имени происходит только при активации поля *Name Device control* для непосредственной записи имени устройства к устройству ввода – вывода.

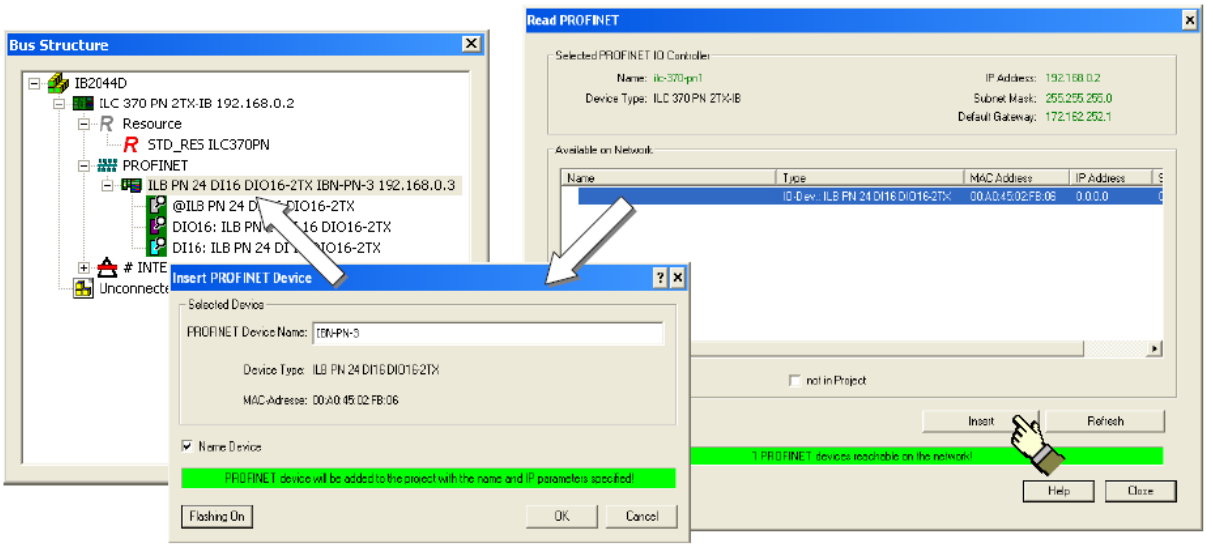


Рисунок 2.8 – Вставка устройств в проект

Для каждого устройства ввода – вывода PROFINET временное обновление для вводов и выводов может быть определено через вкладку *PROFINET Settings* (рисунок 2.9).

Время обновления для отдельных устройств ввода – вывода так же как для вводов и выводов для каждого устройства могут отличаться.

Имена отдельных устройств ввода – вывода в проекте PC WORX могут быть присвоены через вкладку *PROFINET Settings*.

Доступный набор символов для того, чтобы присвоить имена устройств будет следующий:

1) Буквы: a-z;
2) Цифры: 0-9;
3) Знак дефиса: «-».

Примечание – Присвоение имени устройства отсылает исключительно к присвоению в проекте PC WORX. Фактическое присвоение имени устройства должно быть реализовано отдельно.

Внимание! Имя устройства имеет самый высокий приоритет в PROFINET. Оно служит для идентификации устройств ввода – вывода в системе PROFINET, а также должно быть однозначным для использования во всей сети.

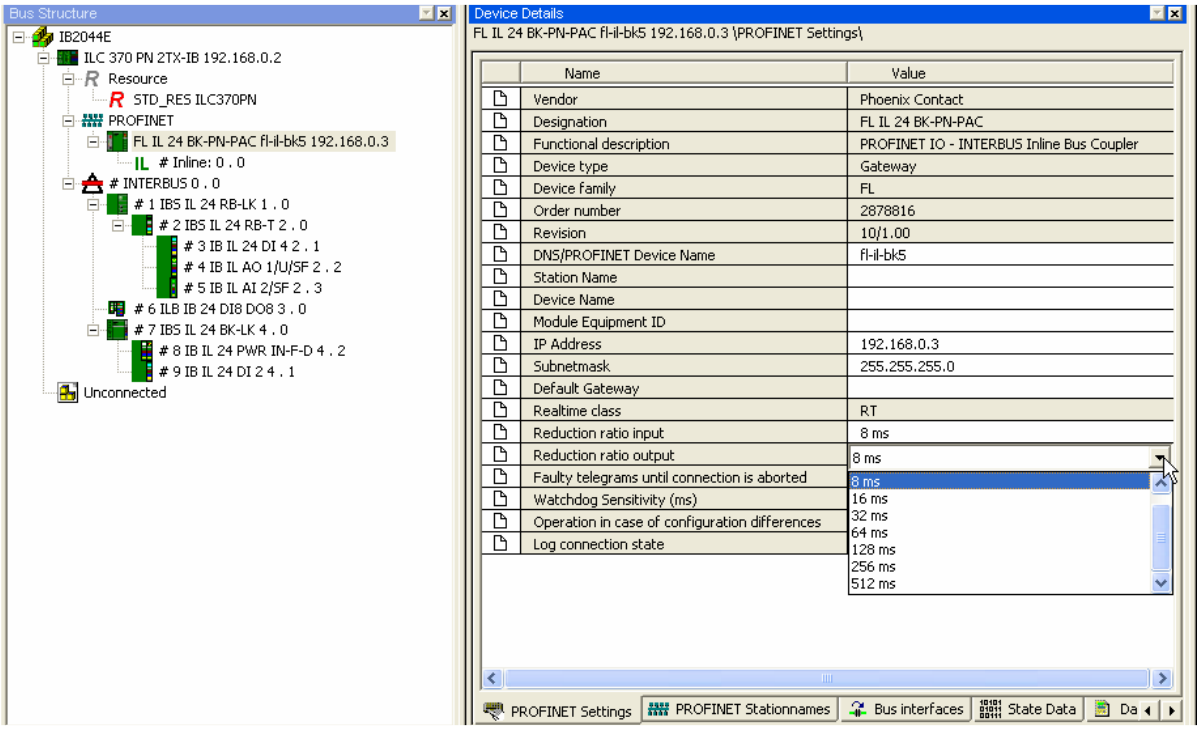


Рисунок 2.9 – Настройки устройства

Устройства PROFINET в сети

Через вкладку *PROFINET Stationnames* все устройства ввода – вывода PROFINET, доступные в сети, считываются и отображаются в начале списка (рисунок 2.10). Используя это диалоговое окно, можно записать имена устройств PROFINET и соответствующие IP-адреса для каждого устройства ввода – вывода. Фактическое присвоение имени устройства реализуется нажатием кнопки *Assign Name*.

Примечание – Вкладка *PROFINET Stationnames* доступна только, если устройство ввода – вывода PROFINET было выбрано в конфигурации шины.

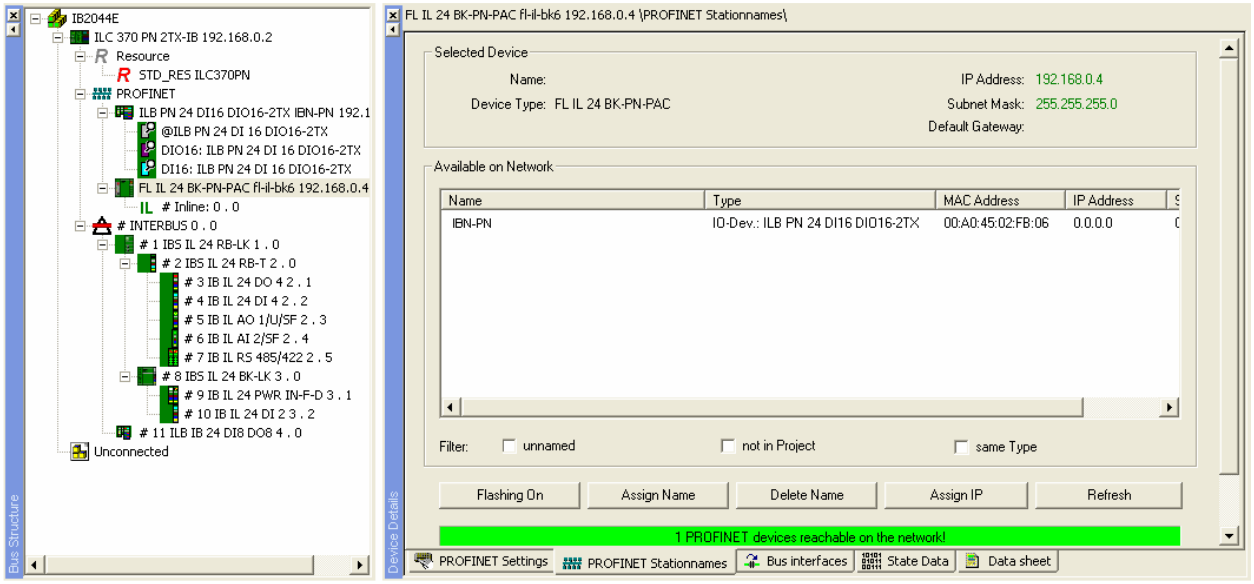


Рисунок 2.10 – Устройства PROFINET в сети

При присвоении имени, имя выбранного в конфигурации шины устройства ввода – вывода присваивается устройству ввода – вывода, выбранному в окне *Device Details*. Пожалуйста, удостоверьтесь, что выбранные устройства имеют тот же самый тип. Имя устройства постоянно сохраняется на соответствующее устройство ввода – вывода.

Примечание – Описанный здесь процесс обычно упоминается как присвоение имени устройству.

В отличие от энергозависимого IP-адреса, имя устройства остается закрепленным за устройством ввода – вывода PROFINET после того, как напряжение сброшено.

Контрольные вопросы

1. Опишите конфигурацию системы управления через *Device Catalog*.
2. Какими действиями относительно конфигурации шины ограничена функция *Device Catalog*?
3. Какой набор символов используется для присвоения имени устройствам?

Задания

Задание 1. Скопировать конфигурацию шины в режиме online

Через диалоговое окно *Connected Bus* (Шина соединения), используйте систему управления, чтобы просмотреть конфигурацию шины соединения. Скопируйте эту конфигурацию (устройства шины на базисной платформе установки), выбирая соответствующие устройства из диапазона доступных устройств.

Задание 2. Расширить конфигурацию шины в режиме offline

После соединения устройств, непосредственно связанных с системой управления, используйте *Device catalog* (Каталог устройств) к оставшимся устройствам магистральной шины, чтобы расширить конфигурацию в программном обеспечении.

Лабораторный практикум

Лабораторная работа №1. Работа с переменными, типы данных, входные и выходные сигналы

Цель: создание переменных для обработки данных (входных и выходных сигналов) системной шины, сконфигурированной в PC WORX. Изучение механизма связи глобальных переменных с входными и выходными сигналами системной шины.

1.1 Работа с переменными

Для создания переменных необходимо выбрать элемент, соответствующий модулю ЦПУ ПЛК в левом верхнем квадранте области. До того как пользователь переименует этот элемент при создании проекта, в стандартном шаблоне проекта элемент имел обозначение *STD_RES (Standard Resource)* – модуль ЦПУ ПЛК.

Создание завершается выбором функции *Create Variable (Создать переменную)* из контекстного меню выбранной строки (рисунок 1.1). Убедитесь, что данные процесса (*Process Data Items)* не пересекаются между собой, т. к. это позднее может привести к ошибкам при компиляции проекта. Созданные переменные автоматически помещаются в группу переменных *Auto*, которая создается автоматически программой PC WORX.

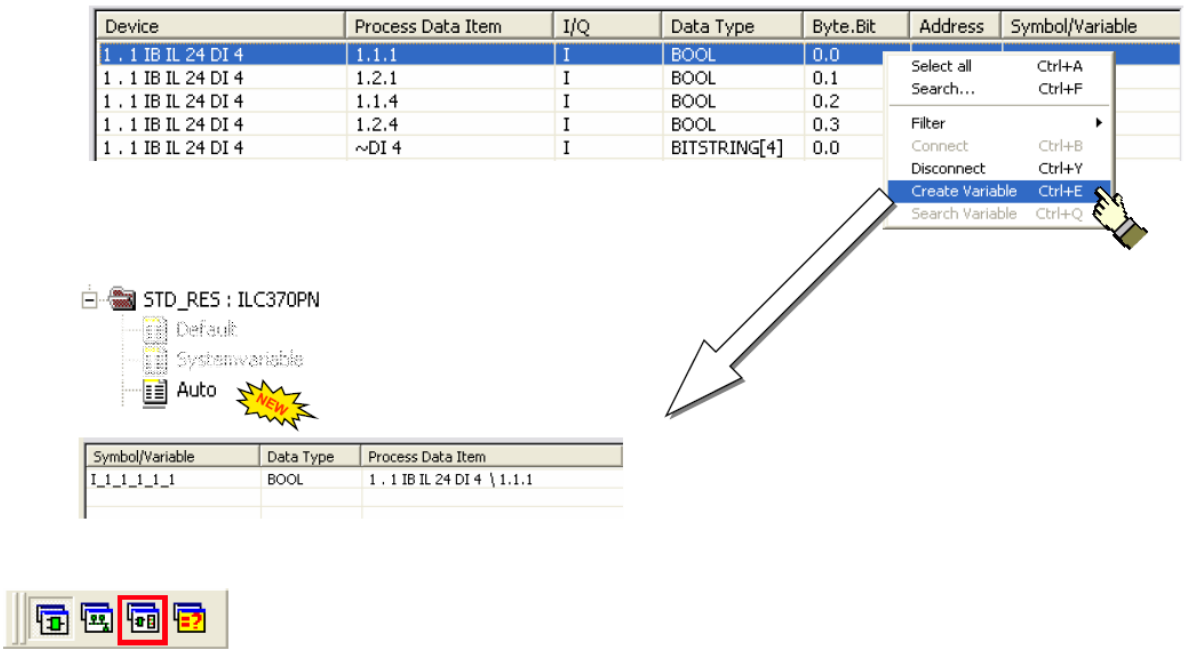


Рисунок 1.1 – Создание переменных

Название автоматически созданных переменных дается по образцу:

<data direction>__<position>_<connection point>

Для того чтобы соединить данные процесса (*Process Data*) с уже существующими глобальными переменными, в верхнем левом квадранте выберите группу переменных, в которой сохранена переменная, которую необходимо связать (рисунок 1.2). Из верхнего правого квадранта выберите модуль и затем данные о процессе, с которыми переменная должна быть связана.

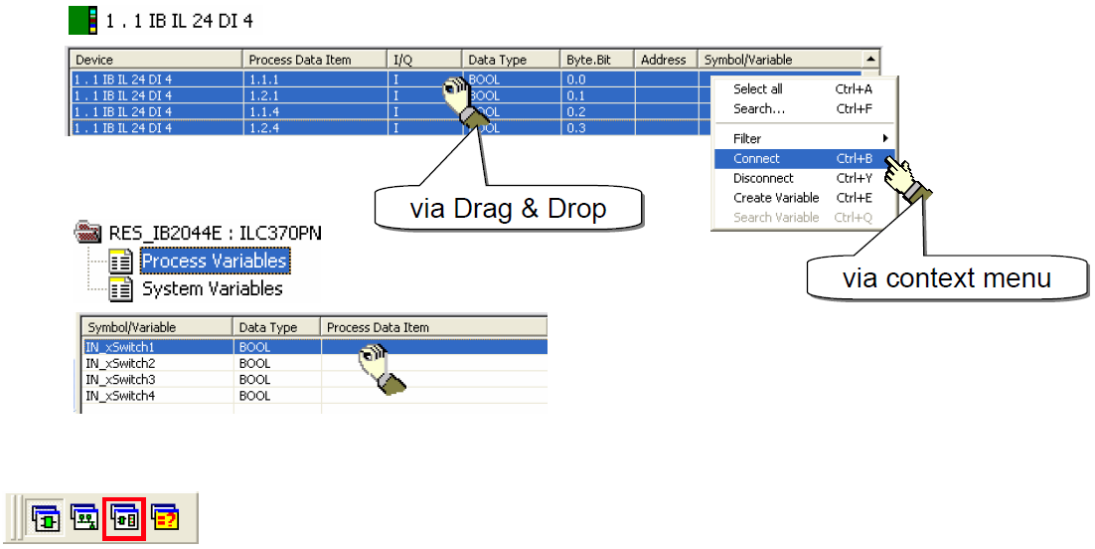


Рисунок 1.2 – Соединение данных процесса с глобальными переменными

Функция *Connect* (соединить) будет доступна в контекстном меню переменной (см. рисунок 1.2).

Как альтернатива связь может быть установлена, используя *Drag & Drop*.

Независимо от того, каким способом происходит соединение данных о процессе с переменными, результатом будет то же самое. Только названия для созданных переменных необходимо привести к виду, соответствующему эксплуатационным требованиям (рисунок 1.3).

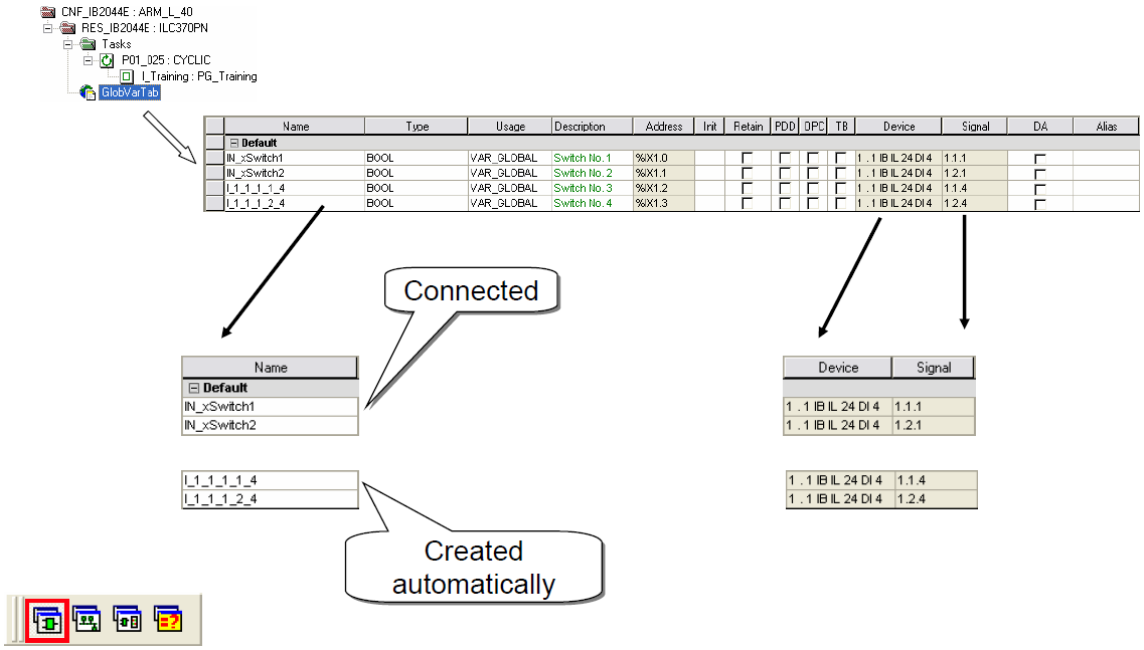


Рисунок 1.3 – Название переменных

1.2 Конфигурация ПЛК

Модель программного обеспечения представлена на рисунок 1.4.

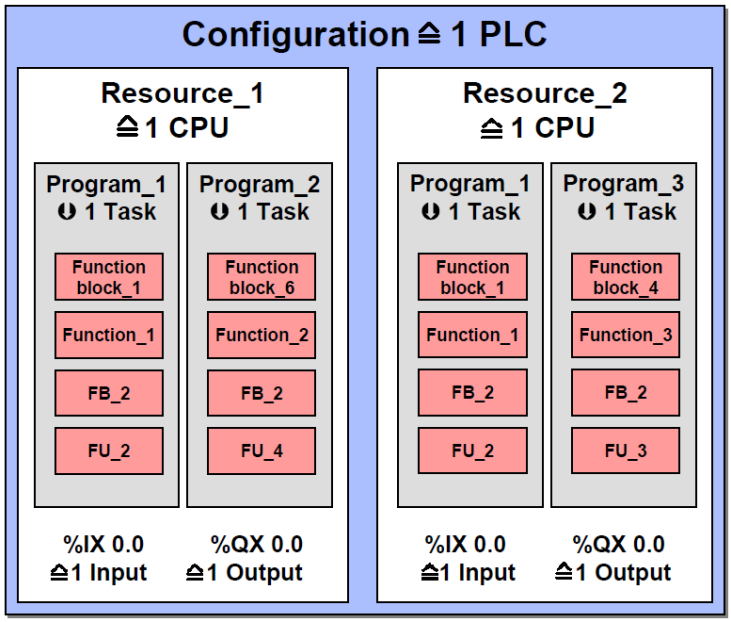


Рисунок 1.4 – Модель программного обеспечения в соответствии со стандартом IEC 61131

Configuration **(**Конфигурация) описывает полную систему PLC (физика и логика).

Resource (Ресурс) предлагает поддержку всех функций, которые необходимы для выполнения программ; интерфейс между одной программой и физическими каналами ввода – вывода ПЛК.

Tasks (Задачи) отображает время для контроля выполнения различных прикладных программ.

Program (PG) (Программа) – программный блок (POU), назначается на определенную задачу.

Function block (FB) (Функциональный блок) – Программный блок (подпрограмма) (POU), содержит статические данные.

Function (FU) (Функция) – подпрограмма (POU), не содержит исчерпывающих данных.

В левой части рабочей области отображается описание проекта в соответствии с IEC 61131 (рисунок 1.5).

В верхней части дерева проекта содержится описание логической части проекта: библиотек, типов данных и логических POU.

В нижней части описывается физическая структура: тип центрального процессора(ов), задача структуры, доступ к данным процесса.

В правой части рабочей области отображаются составляющие отдельных элементов.

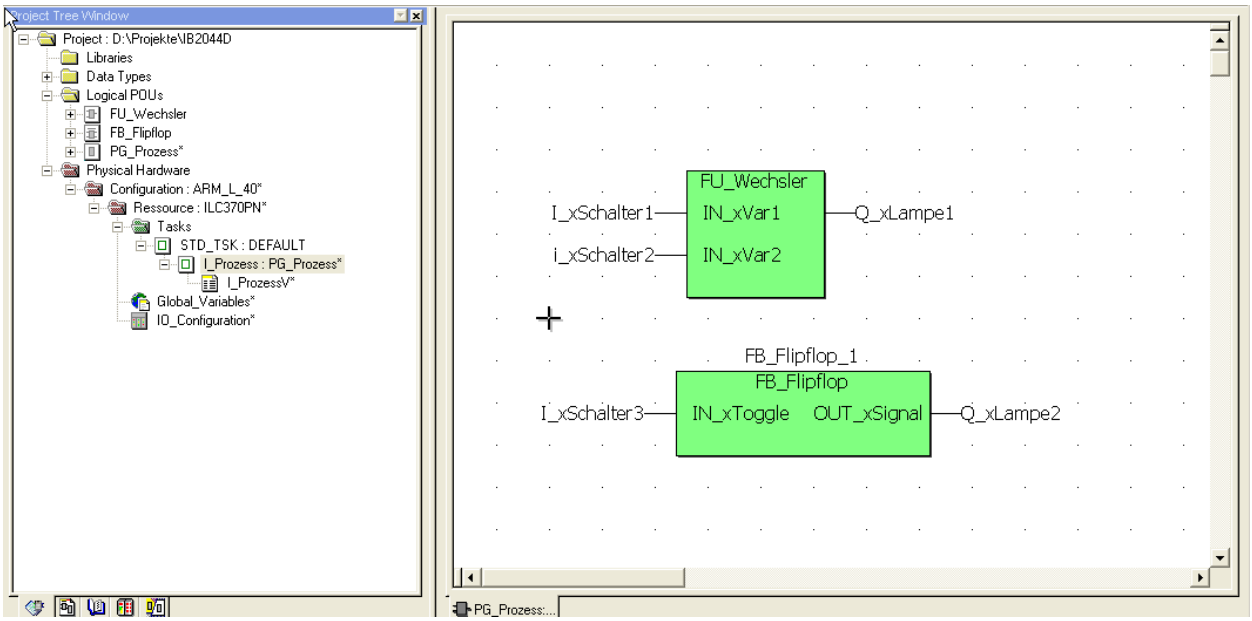


Рисунок 1.5 – Рабочая область

В PC WORX структура аппаратной составляющей ПЛК состоит главным образом из стойки ПЛК, процессора ПЛК и управления процессорной мощностью (рисунок 1.6).

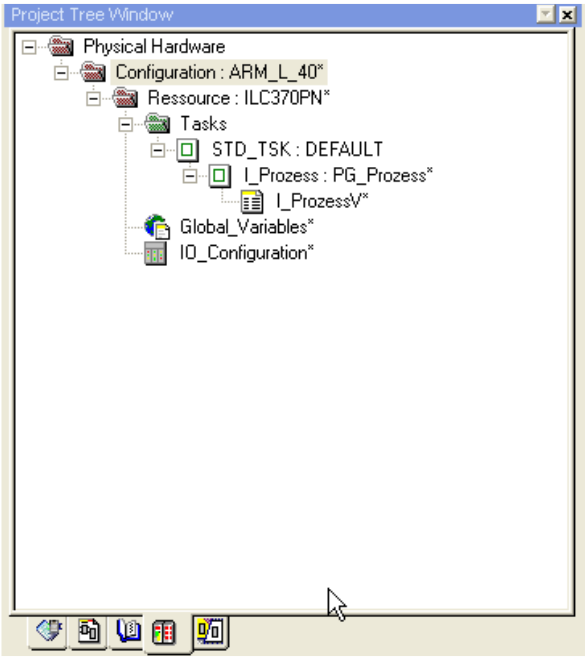


Рисунок 1.6 – Дерево аппаратной конфигурации

1.3 Исполнительные классы

Согласно IEC 61131, высокоуровневый элемент структуры аппаратных средств ПЛК называют *Configuration* (Конфигурация). Для классических систем управления это соответствует стойке PLC.

После установки, в PC WORX доступны четыре различных исполнительных класса (рисунок 1.7). Эти классы отличаются относительно используемых процессоров. Кроме того для двух верхних исполнительных классов доступны различные версии системы исполнения.

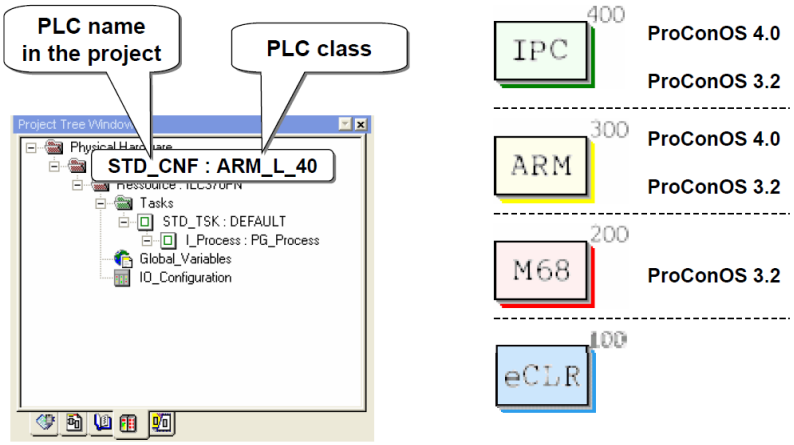


Рисунок 1.7 – Исполнительные классы

Класс 400 рассчитывает на процессоры INTEL, в классе 300 используются процессоры ARM и для систем управления класса 200 используются процессоры Motorola. Последние системы управления класса 100 используют eCLR.

Различие между двумя версиями 3.2 и 4.0 операционной системы – степень редактирования вариантов во время работы ПЛК.

Из-за возможности работать с несколькими системами управления в дополнение к исполнительному классу системы управления, должно быть назначено имя. Шаблоны проекта в PC WORX стандартно используют имя *STD_CNF* (стандартная конфигурация).

Внимание! Термин «Конфигурация», используемый здесь, не связан с конфигурацией INTERBUS.

Процессоры, которые будут использоваться, и, следовательно, соответствующая система управления зависят от исполнительного класса. Рисунок 1.8 показывает, какие системы управления с какими исполнительными классами связаны. Некоторые системы управления доступны с двумя версиями 3.2 и 4.0 ProConOS.

Как и с исполнительным классом, в дополнение к типу процессора, должно быть назначено имя для элемента ресурса (*Resource*). Шаблоны проекта в PC WORX стандартно используют имя *STD_RES* (стандартный ресурс).

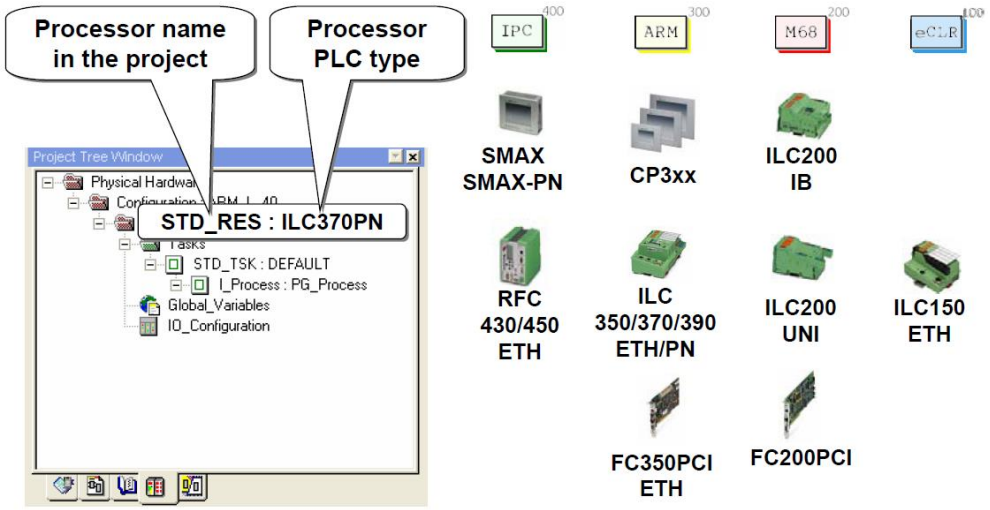


Рисунок 1.8 – Связь систем управления с исполнительными классами

1.4 Задачи в PC WORX

Благодаря управлению задачами в PC WORX производительность процессора может точно использоваться и настраиваться на приложение.

Задачи разделены на два класса: циклические и управляемые событием (результатом) задачи (рисунок 1.9).

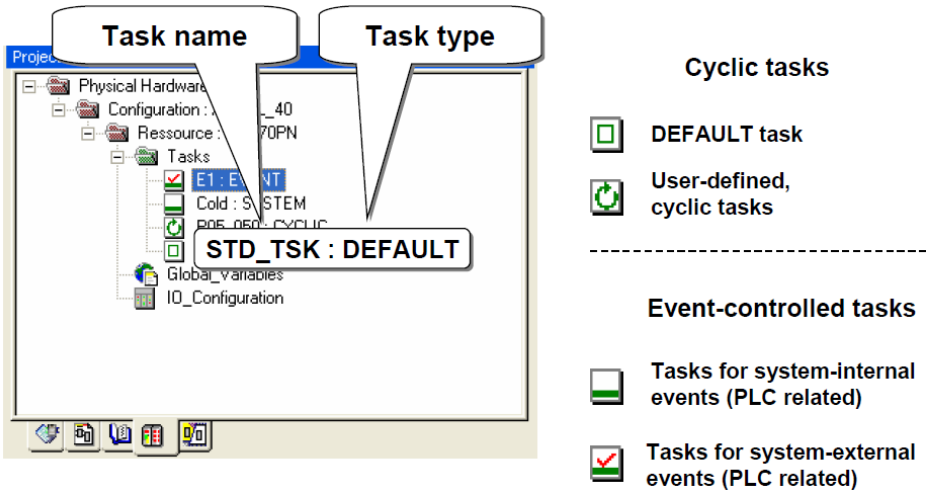


Рисунок 1.9 – Задачи в PC WORX

Обработка данных в ПЛК обычно представляет собой циклическое выполнение. Задача *DEFAULT* (по умолчанию) является циклической, поэтому может использоваться для большинства приложений. Она работает с самым низким приоритетом и минимальное время цикла предлагает tmin=2·system tick (два системных интервала). Для *IPC* и *ARM*

процессоров системные интервалы установлены в 1 миллисекунду, для процессоров M68 – 5 миллисекунд. Задача *DEFAULT* (по умолчанию) может быть создана только один раз, и из-за ее низкого приоритета она может быть прервана любой задачей.

Определенные пользователем задачи *CYCLIC* (циклические) предлагают возможность свободно выбрать приоритет и время цикла в рамках технических условий. Приоритет может быть установлен между значениями «0» (самый высокий) и «31» (самый низкий).

Ко второму классу задач относятся задачи *SYSTEM* (системные) и *EVENT* (событие). Системные задачи отвечают за внутренние события ПЛК (например, холодный старт, теплый старт, деление на нуль и т. д.). Они включают в себя регулярные и нерегулярные состояния системы управления.

В зависимости от используемой системы управления определяются внешние события, которые могут быть определены как задачи *EVENT* (событие) (например, дискретные входы, состояние которых изменяется как случайная величина). Одно из событий – окончание цикла, когда подключен INTERBUS. События для задач *EVENT* и *SYSTEM* могут быть установлены в диалоговом окне PC WORX.

Можно определить удобную структуру аппаратных средств, назначая имена, которые соответствуют особенностям. У задач предварительной обработки данных всегда должно быть имя PDP.

Выбор времени задачи *DEFAULT* (по умолчанию) соответствует времени большинства классических систем управления (рисунок 1.10). Каждый следующий цикл начинается непосредственно после выполнения начатого цикла. Нет никаких фиксированных интервалов.

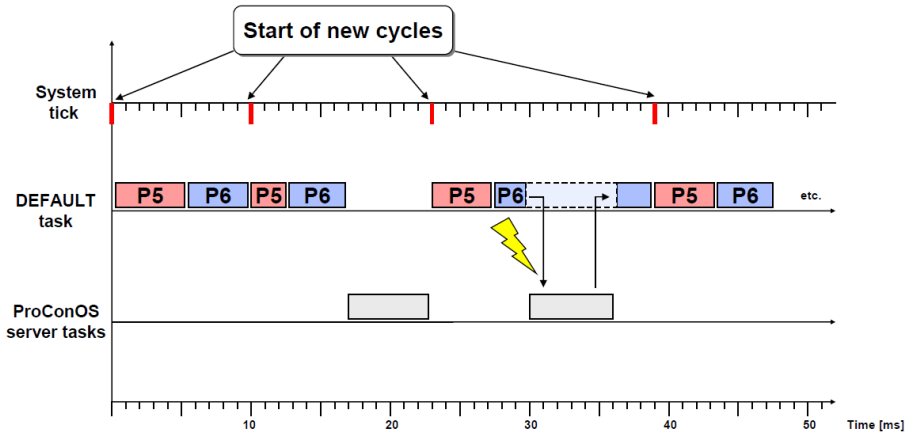


Рисунок 1.10 – Задачи DEFAULT

Из-за программирования время цикла может значительно изменяться. Прерывание задачи по умолчанию задачей сервера (отладка или коммуникация) также приводит к изменениям времени цикла. После каждого цикла (код P5 и код P6) немедленно начинается следующий.

Детерминированные интервалы времени – обработка цикла в пределах фиксированных интервалов, независимых от эффективного времени цикла.

Задачи в реальном времени характеризуются временным детерминизмом: блок (P1), назначенный задаче, выполнен однократно в фиксированных интервалах (рисунок 1.11). Таким образом, может быть вычислено время отклика. Задачи сервера становятся активными, только если задачаи реального времени уже выполнены.

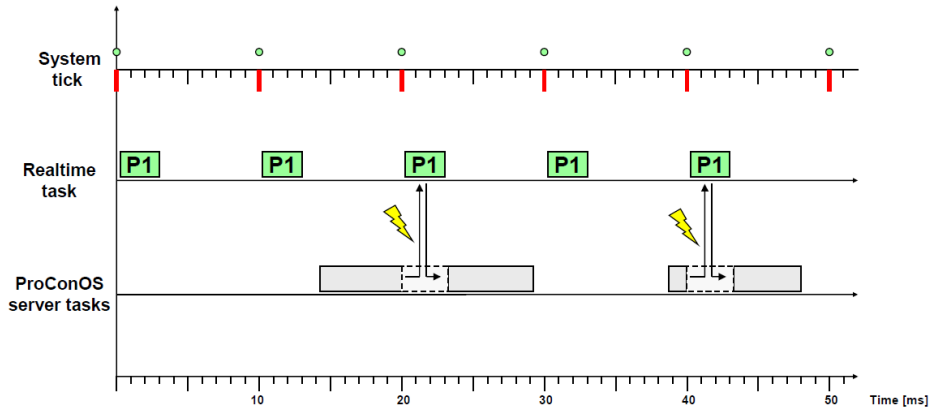


Рисунок 1.11 – Задачи в реальном времени

Контроль времени гарантирует, что, даже если обработка назначенного блока не может быть закончена в пределах интервала (например, блок содержит слишком много команд процессора), вычислительная операция все еще будет осуществляться. Аналогично, если выключатели системы управления находятся в состоянии STOP или выполняется определенная пользователем операция (рисунок 1.12).

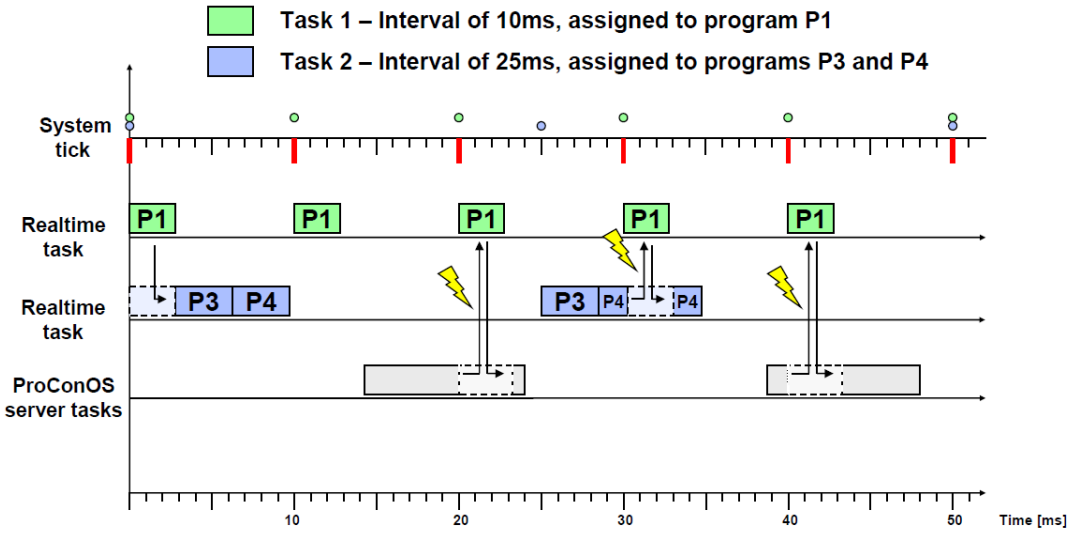


Рисунок 1.12 – Несколько задач реального времени

1.5 Программные блоки

Согласно IEC 61131 определены три различных типа программных блоков: программа, функциональный блок и функция (рисунок 1.13). Вообще, программирование можно определить как неструктурированное выстраивание в линию команд. Однако использование этих трех типов программных блоков вносит более высокую степень ясности в проекте, и, что важнее, предлагает преимущества относительно эффективного программирования на стадии редактирования.

Особенности и области применения отдельных типов программных блоков описывают обмен информации между формируемыми программными блоками.

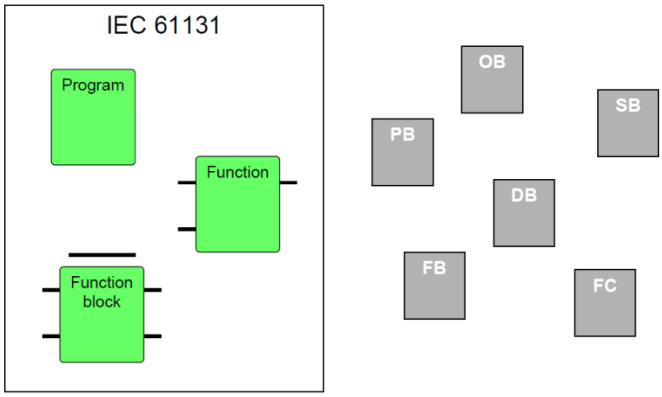


Рисунок 1.13 – Программные блоки в соответствии со стандартом IEC 61131

На рисунках 1.14 и 1.15 показана иерархия трех типов программных блоков и их базовые свойства.

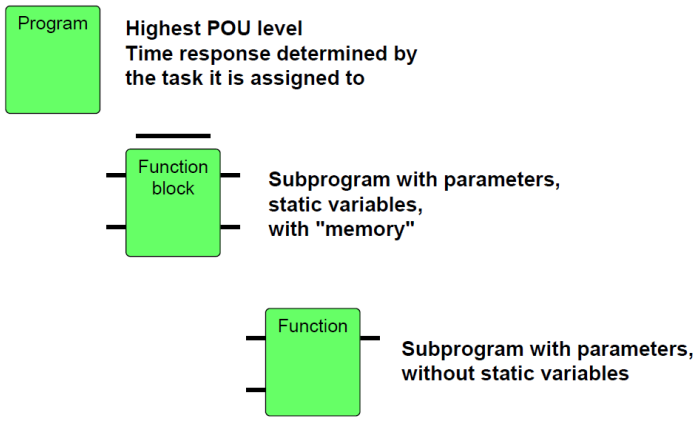


Рисунок 1.14 – Иерархия трех типов программных блоков

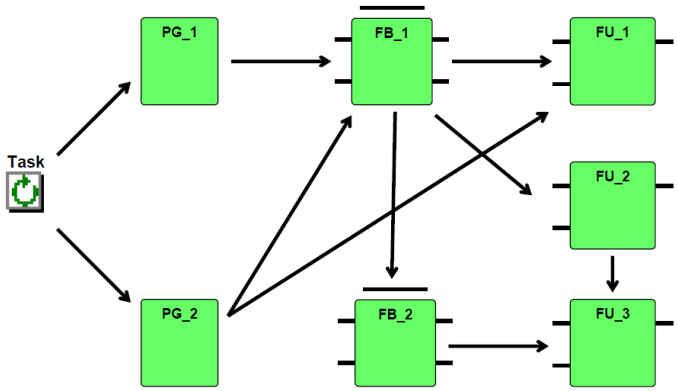


Рисунок 1.15 – Пример организованного вызова структуры

Программа – программный блок высшего уровня (рисунок 1.16). В проекте требуется, по крайней мере, один программный блок этого типа. Программы так же, как и другие программные блоки, сохранены в папке *Logical POUs* и не влияют на проект автоматизации. Только, когда программа используется в задаче, сохраненная программа выполнена на ПЛК.

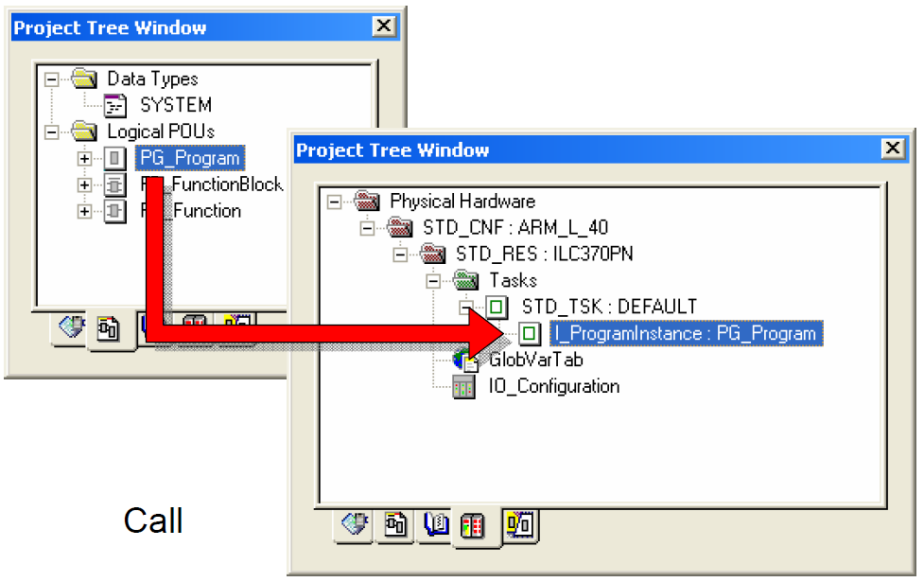


Рисунок 1.16 – Программа как тип программного блока

Экземпляр программы – область памяти центрального процессора, в котором выполняется программа в зависимости от задачи. Реализация программы – это запуск программы ЦПУ на выполнение. PC WORX поддерживает многократную реализацию программы.

Одна из главных задач программ – связывать сигналы аппаратных средств в программирование. Глобальные переменные – единственный способ установить эту связь. Они также используются, чтобы организовать обмен глобальной информацией с системой управления (рисунок 1.17). Для программ глобальные переменные – единственная возможность обменяться информацией с другими программными элементами.

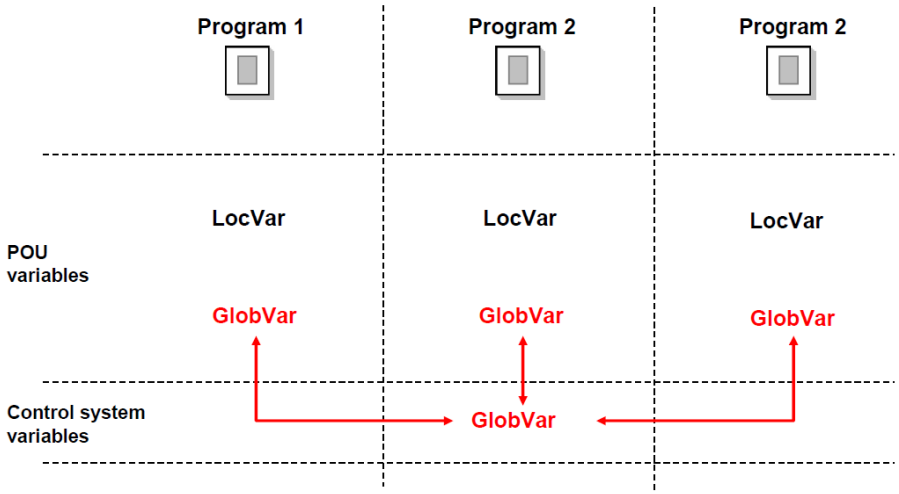


Рисунок 1.17 – Обмен данными между программами

Флаги объявляются как локальные переменные, пока их значения не касаются внутреннего проекта.

Как параметризованные элементы, блоки функции можно вызывать в программах и других функциональных блоках. Создание функциональных блоков (и функций) в основном основывается на двух принципах. Во-первых, как только код программы был создан, он может использоваться многократно. Во-вторых, параметризированные программные блоки также обеспечивают преимущество инкапсуляции. Это означает, что сложная программа создана, например, на текстовом языке. Подпрограмма, включая связь только нескольких входных и выходных значений, затем может быть преобразована в графический язык, который обеспечивает более ясную структуру.

Как показано на рисунке 1.18, функциональные блоки можно вызывать в других функциональных блоках и в программах. Рекурсивный вызов не возможен.

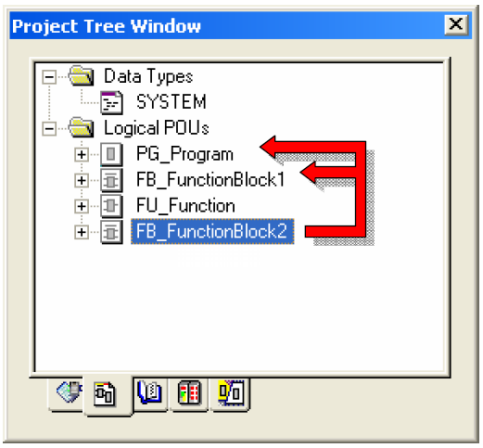


Рисунок 1.18 – Функциональный блок как тип программного блока

Поскольку функциональные блоки могут хранить величины более чем в нескольких циклах программы для каждого вызова функционального блока, в системе управления должна быть выделена отдельная память. Эту память называют экземпляром функционального блока и ее можно сравнить со сложной переменной. Экземпляр функционального блока занесен в список переменных программного блока (POU), в котором вызывают блок функции.

Относительно обмена данными функциональные блоки обеспечивают самую высокую гибкость (рисунок 1.19). Согласно классическому подходу, обмен данными выполняется только через

параметры входа и выхода (переменные интерфейса). Для определенных приложений, однако, мог бы иметь смысл обмен информации с функциональным блоком через глобальные переменные. Неудобство для программиста при использовании этого метода заключается в том, что нет никакой ясности относительно обработанных данных во время вызова.

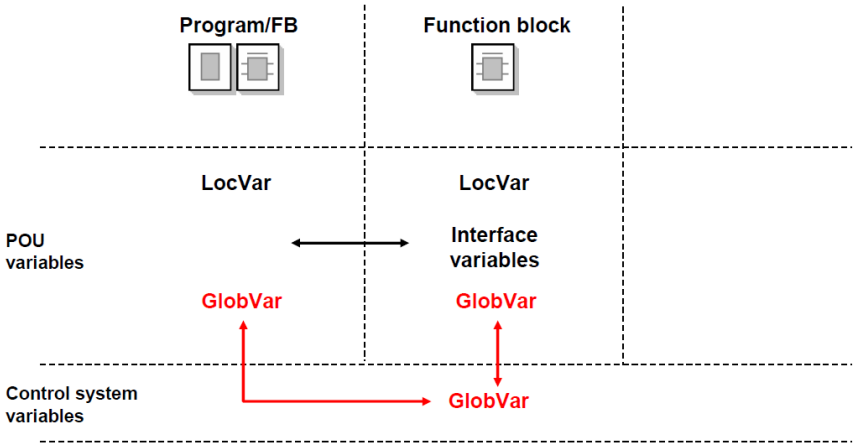


Рисунок 1.19 – Обмен данными между функциональными блоками

Функции – второй тип параметризированных программных блоков. Как показано на рисунке 1.20, они могут использоваться в других функциях, функциональных блоках и программах. Точно так же как с функциональными блоками, рекурсивный вызов не возможен.

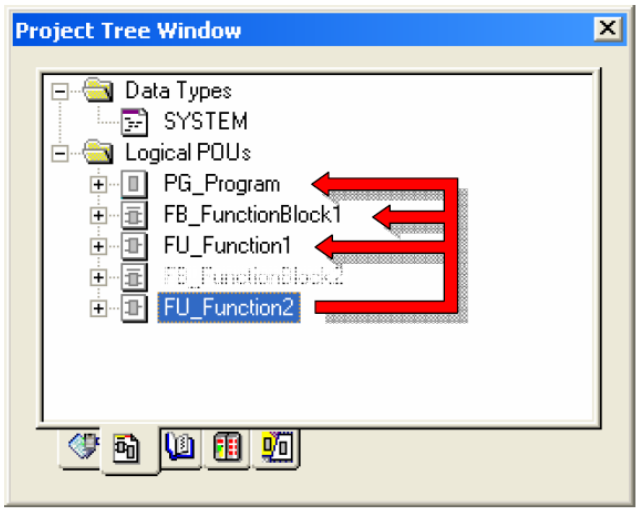


Рисунок 1.20 – Функции как тип программного блока

В отличие от функциональных блоков, функции ограничены в том, что они не могут сохранить данные более чем на несколько циклов ПЛК. Функции также не подходят для случая, если блок должен возвращать

больше одного выходного параметра, поскольку функции возвращают единственный результат.

Преимущество функций, по сравнению с блоками функции, заключается в том, что функция не требует выделения дополнительной памяти при многократном ее использовании. Для многократного использования это означает, что та же самая область памяти используется каждый раз для выполнения функции.

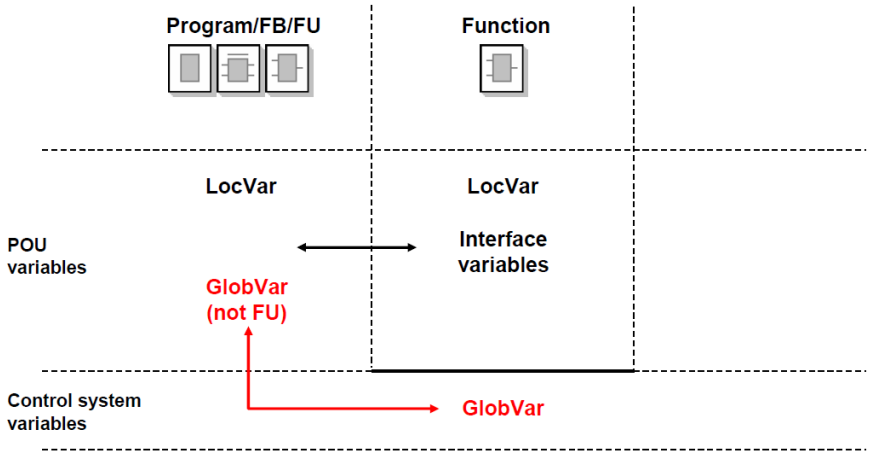


Рисунок 1.21 – Обмен данными между функциями

Что касается обмена данными, функции ясно определены. Они должны иметь по крайней мере один входной параметр и возвратить точно один параметр. Внутреннее возвращаемое значение функции должно быть сохранено в параметр, который уже определен при создании функции в дереве проекта (название параметра = имя функции). Функции не могут получить доступ к глобальным переменным.

Каждый программный блок (независимо от типа) может иметь одно или несколько листов комментариев, в котором программист может сохранить описание реализованных функций или информацию о версии. Только одна таблица локальных переменных доступна для сохранения переменных, необходимых для функции программного блока (рисунок 1.22).

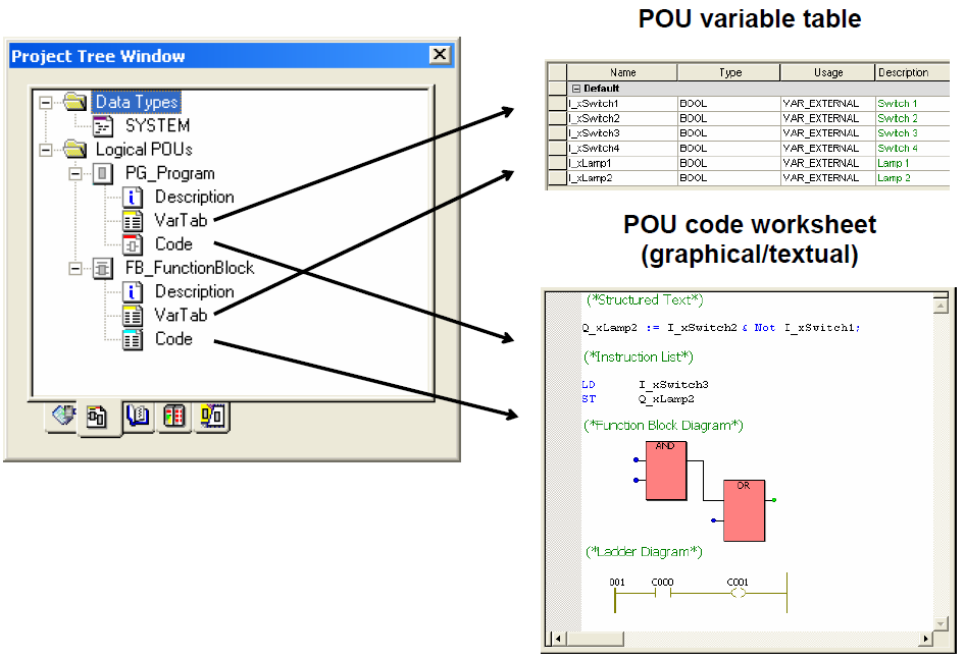


Рисунок 1.22 – Компоненты программных блоков

Для написания программы у программиста в распоряжении один или более графических или текстовых рабочих листов.

Один важный параметр переменной – *тип данных*. После установки PC WORX доступны типов данных, определенные IEC 61131, а после объявления пользователем – пользовательские типы данных (рисунок 1.23).

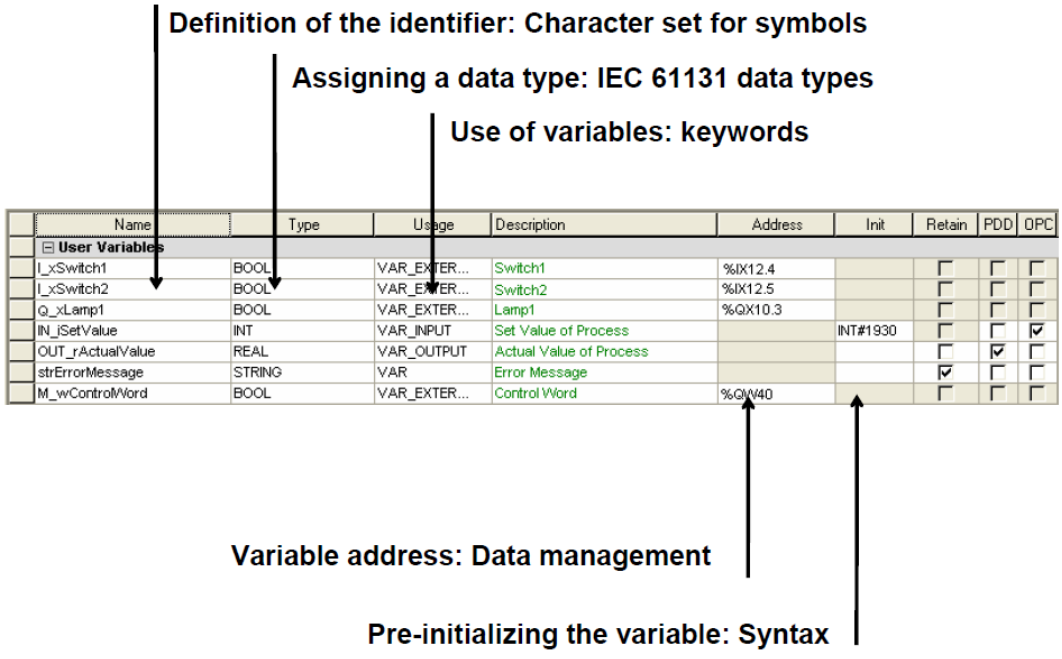


Рисунок 1.23 – Таблица переменных

Использование переменной указывает на права, которые она имеет относительно ПЛК, и какие функции она может выполнять для обмена данными программного блока с другими блоками и ПЛК.

Описание переменной является дополнительным и может содержать любые символы.

Физический адрес системы управления определяет точку обмена переменной с периферийными устройствами. Для этого IEC 61131 определяет фиксированный синтаксис:

% <Принадлежность данных> <Префикс> <Выходной байт [.Позиция бита]>

%<Data direction><Capacity prefix><ByteOffset[.BitPosition]>

Пример: %QX4.0 (выходной бит 4.0),

Переменной можно задавать значение по умолчанию (стандарт 0) (поскольку синтаксис ссылается на постоянное обновление).

1.6 Типы данных

Основная часть типов данных, определенных IEC 61131, доступна в PC WORX. Эти типы данных разделены на группы (рисунок 1.24). Справочная информация по диапазонам используемых для них значений приведена см. в Приложении Б.

Определенные функции и функциональные блоки (с перегруженными параметрами входа и выхода) не обязательно должны быть заданы переменной определенного типа данных, но обязательно типом данных группы (например, функциональные блоки булевой логики И, ИЛИ, исключающее ИЛИ, НЕ (*AND*, *OR*, *XOR*, *NOT*) описываются как *ANY_BIT*).

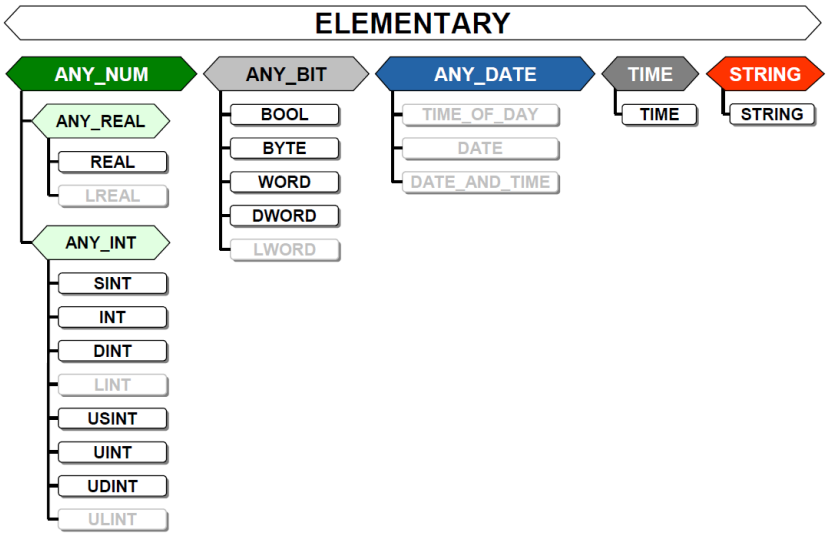


Рисунок 1.24 – Иерархия типов данных

1.7 Константы

Формально правильный синтаксис объявления константы (официальное обозначение констант согласно IEC 61131 является буквенным) следующие (рисунок 1.25):

- тип данных;
- основание (пропуск значения для 10, 2 для двоичной системы, 8 для восьмеричной, 16 для шестнадцатеричным образом закодированной константы);
- постоянная величина;
- единица измерения (только для типа данных TIME (время)), например, с, мс.

Синтаксис констант
<Data type>#<Basis>#<Constant value><Unit>
<Тип данных>#<Основание>#<Постоянная величина> <Единица>

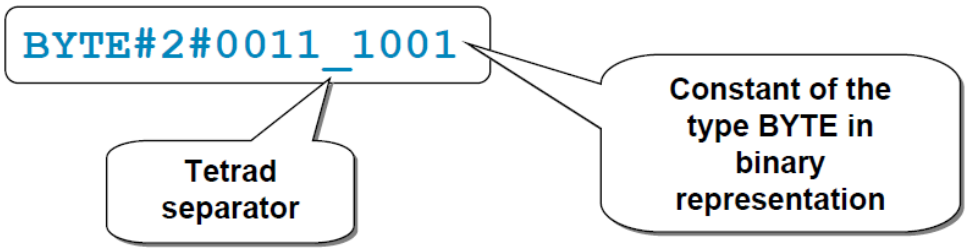


Рисунок 1.25 – Обозначение константы

Как показано на рисунке 1.26 символы «#», а также нижнее подчеркивание помогают улучшить восприятие. Они никак не влияют на значение постоянной величины.

Этот пример показывает возможные форматы записи для констант. Для некоторых типов данных позволено использовать сокращенный синтаксис, который часто применяется при программировании. Сокращенная форма, например, как для типа данных целого числа, не возможна для других типов целых чисел.

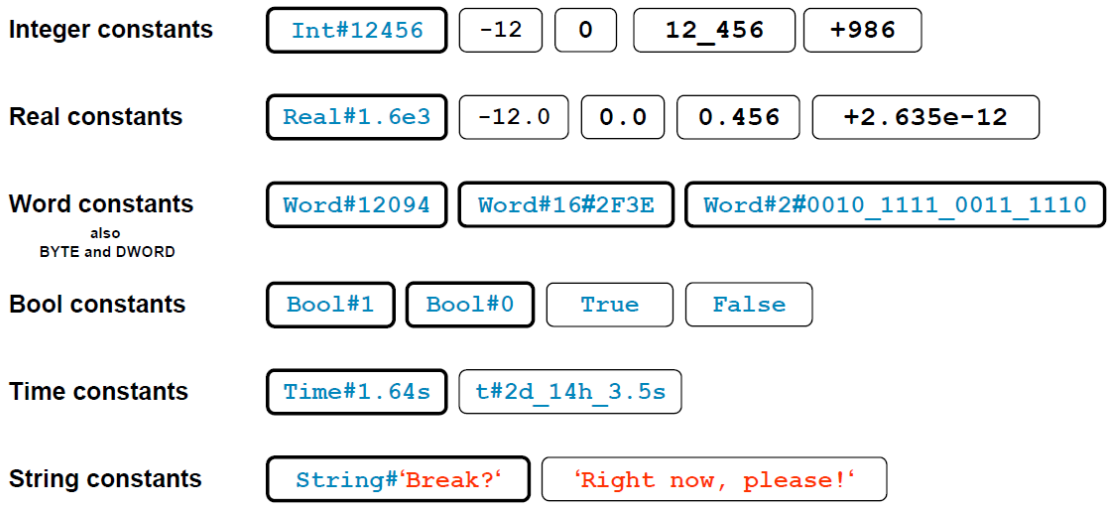


Рисунок 1.26 – Пример записи констант

1.8 Использование переменных

Инкапсуляция данных в IEC 61131 организована по примеру языков высокого уровня.

Используя этот метод, две переменные с одинаковым именем используются в двух различных программных блоках, но обозначают различные параметры (рисунок 1.27).

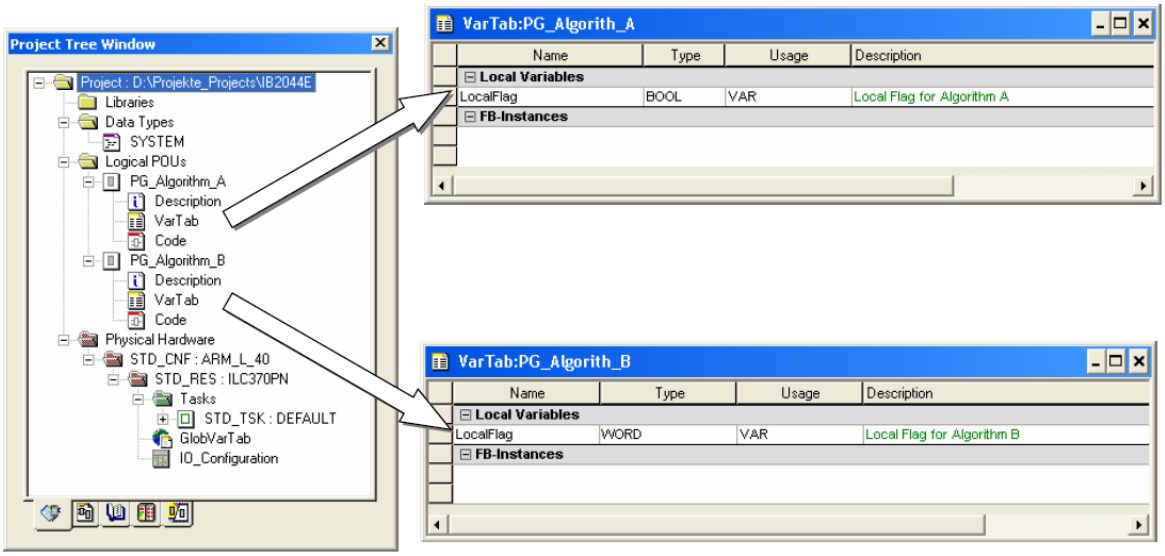


Рисунок 1.27 – Использование локальных переменных

Глобальные переменные хранятся в таблице глобальных переменных. Локальное использование глобальных переменных в любом случае ссылается на эту таблицу (рисунок 1.28).

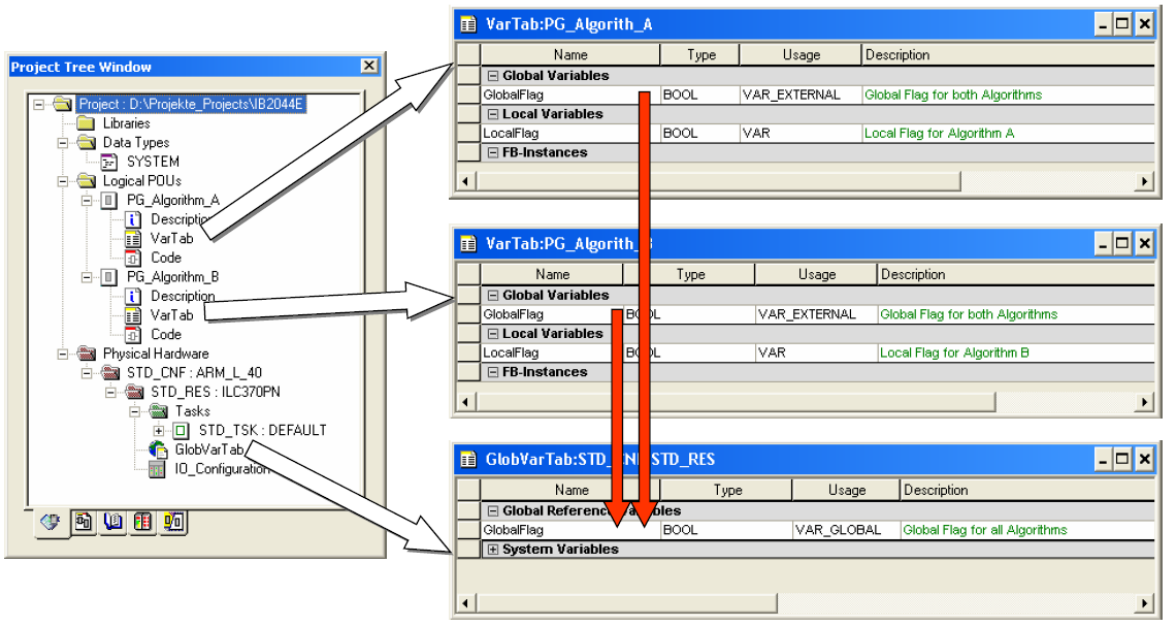


Рисунок 1.28 – Использование глобальных переменных

Глобальная переменная обозначается *VAR_GLOBAL*, а переменная, используемая при программировании в программном блоке (POU), обозначается *VAR_EXTERNAL*.

Таким образом, замена созданной переменной в процессе программирования имеет смысл, только если изменения внесены в таблицу глобальных переменных (PC WORX поддерживает синхронизацию переменной, введенной в программный блок после изменений). Таблица переменных связана с ресурсами и поэтому является частью структуры аппаратных средств.

В таблице локальных переменных находятся локальные и глобальные переменные, которые должны быть связаны друг с другом в программе в программном блоке. В таблице локальных переменных находятся локальные переменные, принадлежащие определенному программному блоку, и локально используемые глобальные переменные.

В контексте глобальных переменных, переменные *VAR_EXTERNAL_PG* и *VAR_GLOBAL_PG* представляют собой особый случай. У этих переменных есть все свойства глобальных переменных. Однако для каждого экземпляра программы, который создан для

программы с такими переменными, выделяется отдельная область памяти. Таким образом, достигается связь с аппаратными средствами (рисунок 1.29).

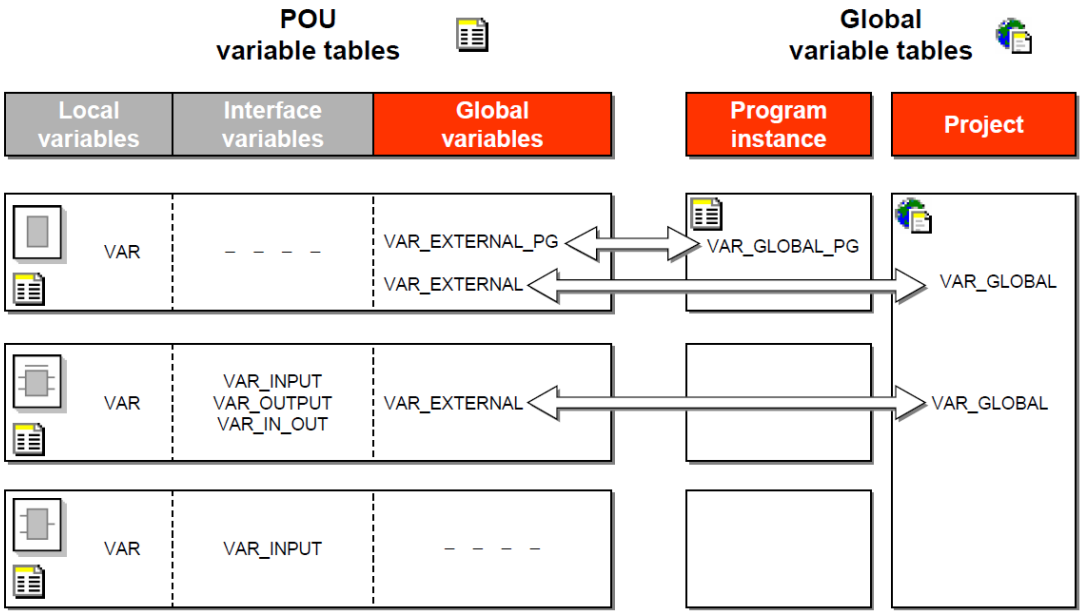


Рисунок 1.29 – Связь переменных программы и глобальных переменных

На рисунке 1.30 показаны возможности использования переменных в функциях и функциональных блоках. Использование *VAR_IN_OUT* особенно интересно при передаче очень больших объемов данных (переменные область и структуры), т. к. в примере нет необходимости копировать данные.

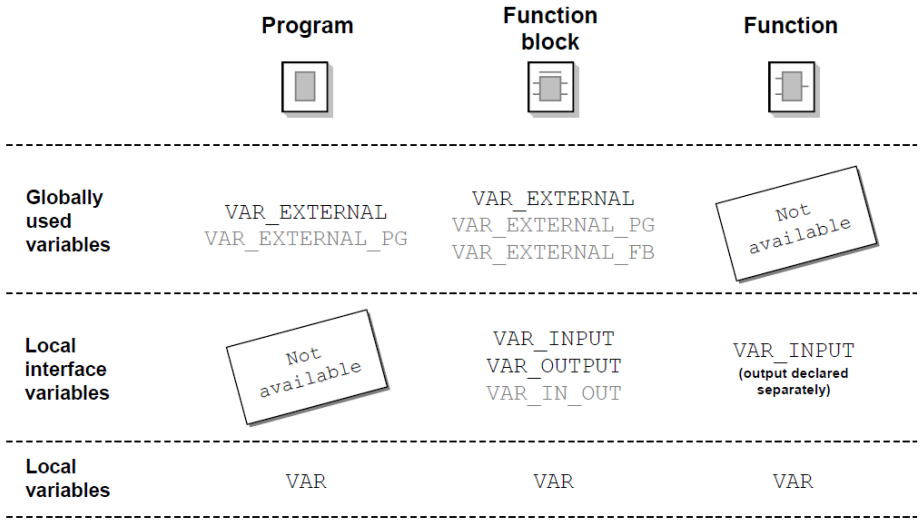


Рисунок 1.30 – Использование переменных в функциях и функциональных блоках

Примеры использования переменных показаны на рисунке 1.31.

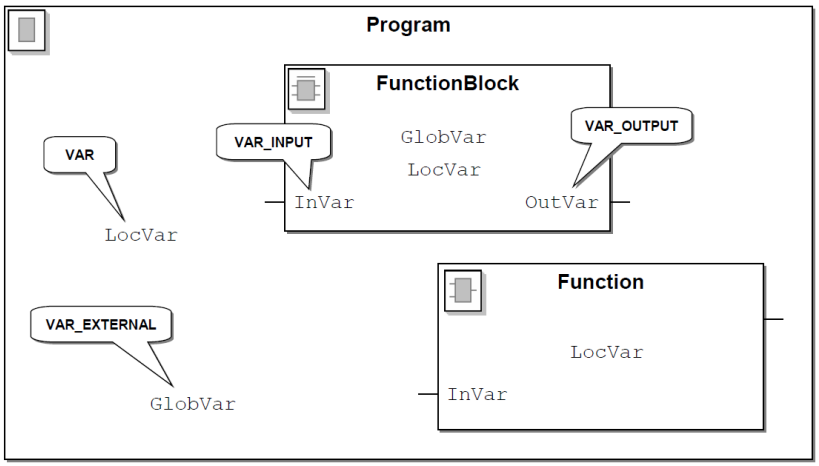


Рисунок 1.31 – Примеры использования переменных

Для достижения более ясной и прозрачной структуры в таблице переменных, рекомендуется разделять переменные на группы. Они никак не влияют на программирование, а просто служат средством для лучшей организации и большей эффективности при работе с переменными.

Группа *Default* (по умолчанию) задается стандартно при создании рабочей области. В зависимости от типа программных блоков можно применить группы, показанные на рисунке 1.32. В более сложных проектах, можно использовать группы переменных по выполняемым функциям.

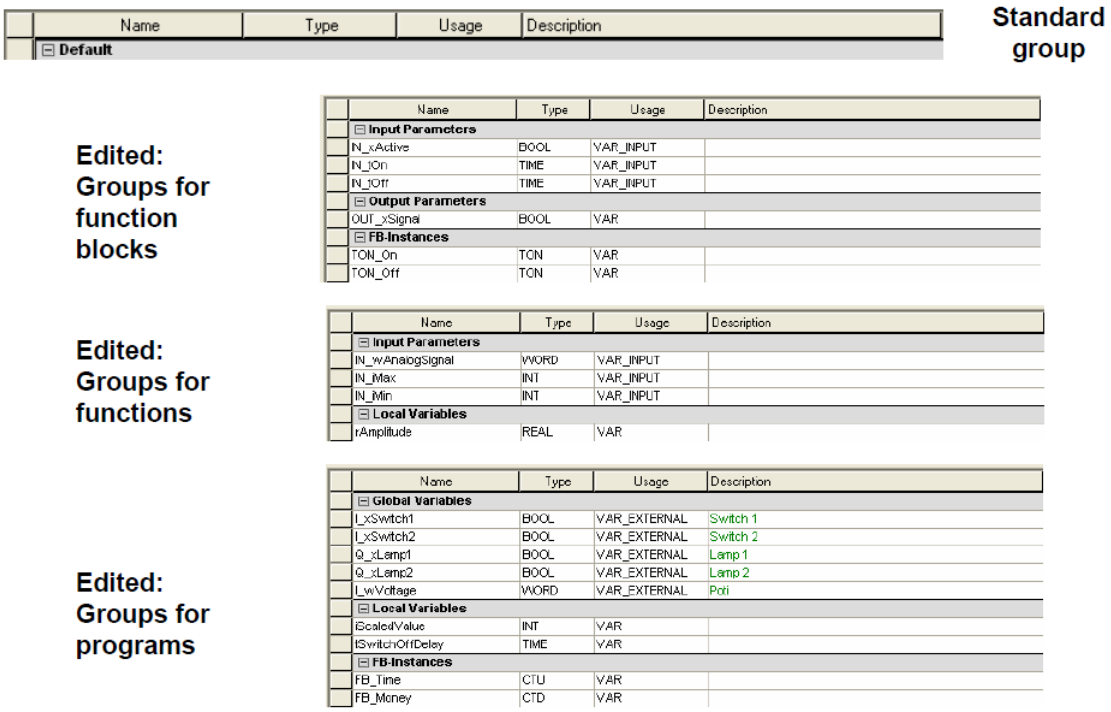


Рисунок 1.32 – Группы переменных

Этот тип группировки подходит не только для таблицы локальных переменных, но также и для таблицы глобальных переменных.

Группа *System Variables*, которая содержит системные переменные, может также быть переименована.

Диалоговое окно переменных позволяет открыть доступ к переменным в соответствующих таблицах локальных и глобальных переменных (рисунок 1.33). Диалоговое окно также позволяет рассмотреть и заменить существующие переменные и объявить новые.

Если глобальные переменные должны быть рассмотрены и объявлены, необходимо указать группу глобальных переменных в дополнение к группе локальных переменных.

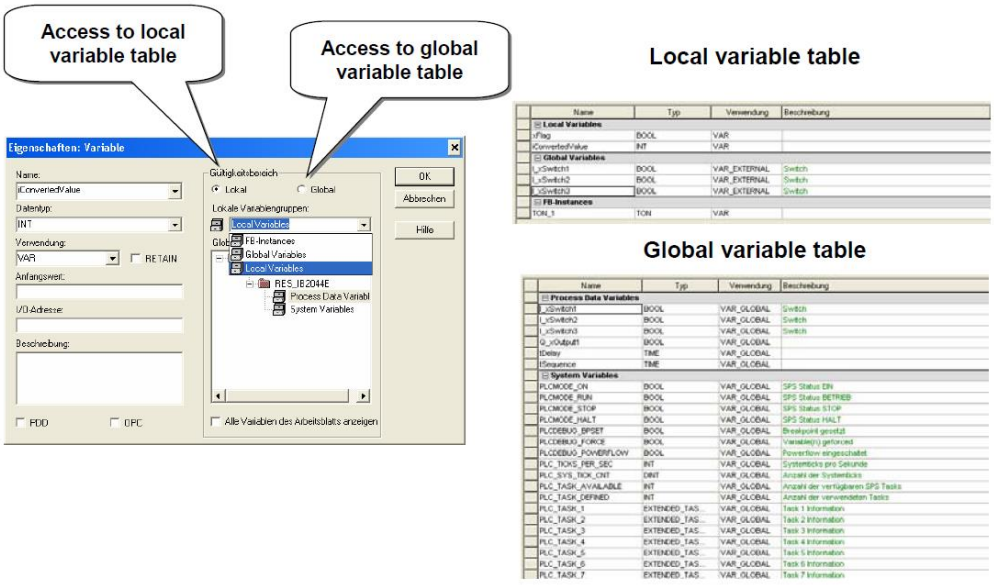


Рисунок 1.33 – Диалоговое окно переменных

Контрольные вопросы

1. Как происходит создание переменных?
2. Как происходит связь систем управления с исполнительными классами?
3. Какие задачи могут использоваться в PC WORX?
4. Для каких целей используется задача DEFAULT?
5. Назовите основные типы программных блоков. В чем их особенности?
6. Какой тип программных блоков обеспечивает самую высокую гибкость обмена данными?
7. Назовите основные типы данных.
8. Какие существуют основные форматы записи для констант?
9. Как реализована инкапсуляция данных в PC WORX?
10. Для чего необходимы группы переменных?
11. В чем разница между глобальными и локальными переменными?

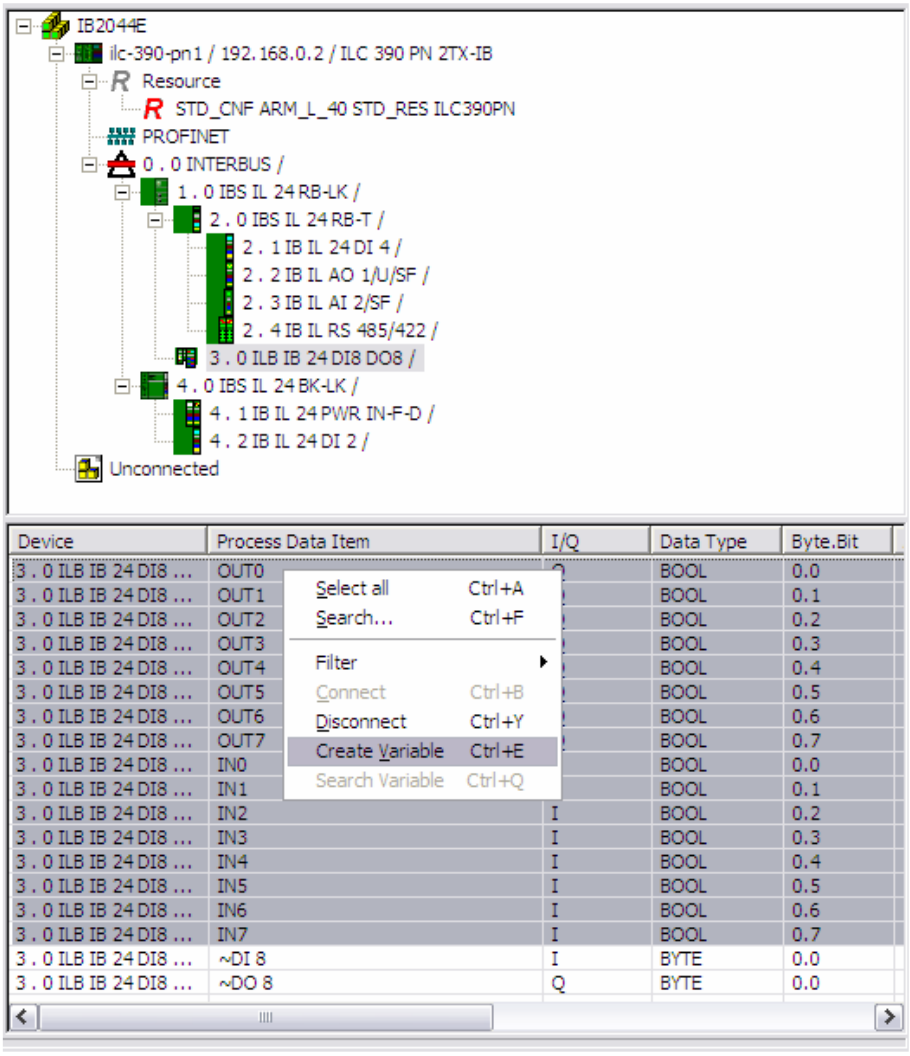
Задания

Задание 1. Создать переменные процесса

На основании обработки элементов данных, которые вам доступны, настройте устройства, создайте глобальные переменные в окне *Process*

Data Assignment. Все автоматически созданные переменные должны быть сохранены на носителе.

Какая связь используются между автоматически сгенерированным именем переменной и пунктом обработки данных?



Задание 2. Проверить соответствие состояний входов/выходов управления и связанных с ними значений переменных процесса

После создания переменных для элементов данных процесса, вы должны скомпилировать проект. Это можно сделать нажатием на указанную кнопку или функциональную кнопку *F9*.

После успешной компиляции необходимо передать проект в основную память системы управления.

Проведите изменения в *Programming View* (если они еще не сделаны) и открыть таблицу глобальных переменных.

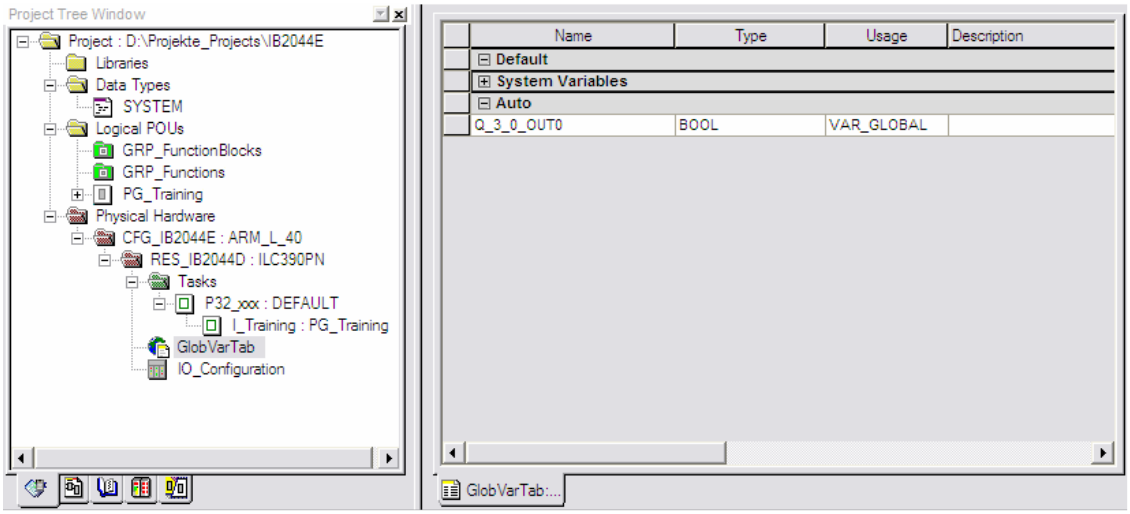
Созданные вами переменные автоматически добавлены в *Auto Group*. Далее сверните все другие группы переменных (по умолчанию, системных переменных) и активируйте режим отладки (диагностические и индикатор состояния) системы управления. Это можно сделать нажатием на указанную кнопку или функциональную кнопку *F10*.

Проверьте состояние ваших входных переменных и используйте ключевые кнопки слева от имен переменных, чтобы открыть диалоговое окно отладки для отдельных переменных. Проверьте правильность работы ваших выходов, управления ими.

Задание 3. Изменить имена переменных

В таблице глобальных переменных корректные имена для них генерируются автоматически, таким образом, чтобы они были аппаратно связаны.

Создайте новую группу переменных *Process Variables* и переместите переменные в эту группу, не теряя связи с данными процесса.



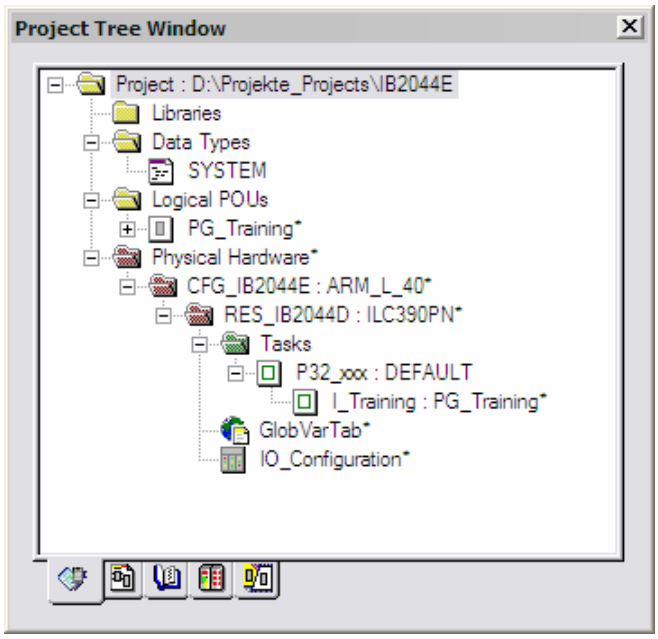
Совет: в таблице переменных содержимое может быть просто перезаписано после выбора ячейки.

При выборе ячейки с помощью клавиш *Pos1* и *End* переместите курсор в положение 1 или конец текста в ячейке.

Совет: серый ключ кнопки в переменной таблице слева от имени переменной может быть использован для обозначения рабочей строки. Если строка отмечена, он может быть перемещен с использованием маркера красной линии цели (без потери связей данных процесса).

Задание 4. Настроить проект

Если используется шаблон проекта, у элементов в дереве проекта всегда отображаются те же стандартные имена. Измените их с помощью диалогового окна свойств элемента.



Совет: элемент свойств диалога дерева проекта можно переименовать с помощью комбинации клавиш *Alt + Enter* или через контекстное меню.

Лабораторная работа №2. Языки программирования PC WORX. Язык FBD (функциональных блок-схем), язык IL (списка инструкций)

Цель: изучение обзорной информации о языках программирования PC WORX, освоение организационных модулей программы и использование их в проекте, изучение функций и функциональных блоков языка функциональных блоковых диаграмм и списка инструкций.

2.1 Языки программирования PC WORX

Для программирования в PC WORX предлагается пять языков, описанных в стандарте IEC 61131(рисунок 2.1).

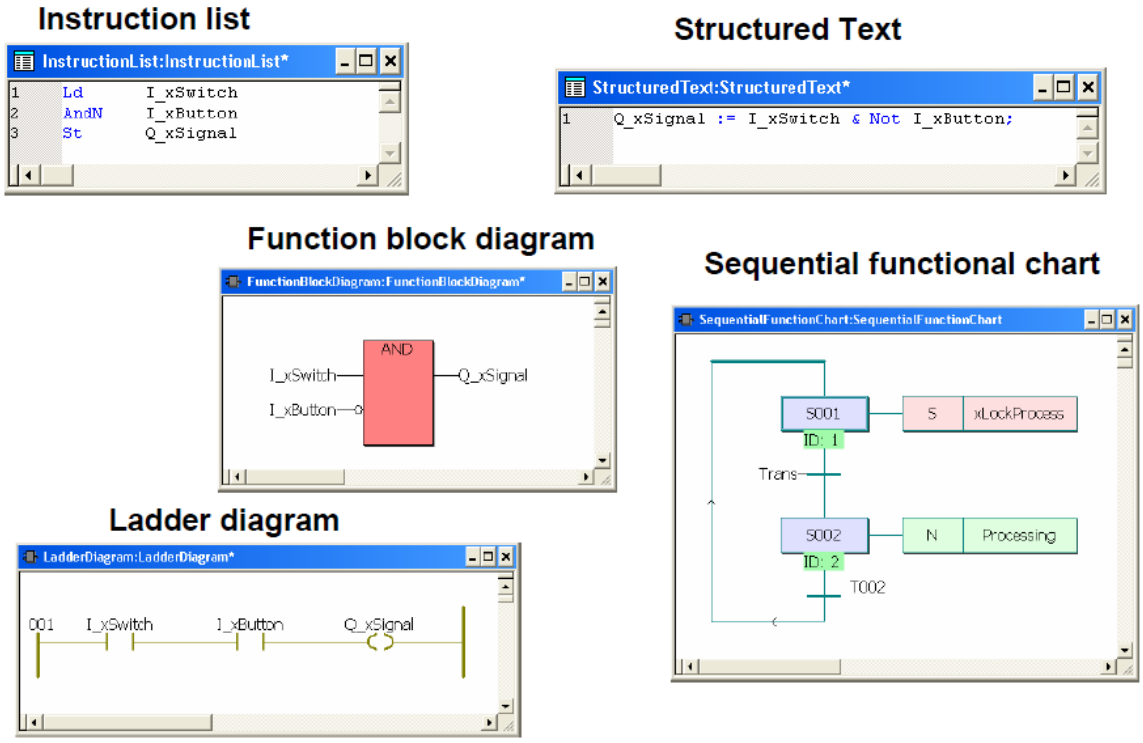
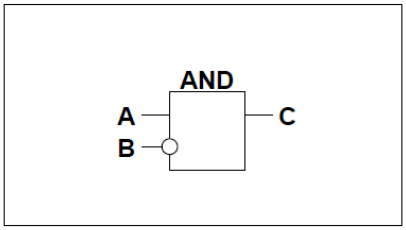


Рисунок 2.1 – Языки программирования IEC 61131-3

Рассмотрим каждый из них более подробно:

1. *Язык функциональных блок-схем (FBD)*:
   a) программные элементы в форме функциональных блоков;
   b) функциональные блоки могут быть «соединены проводом» так же, как в принципиальной электрической схеме;

c) язык, используемый во множестве приложений, ответственных за поток информации между компонентами системы управления.



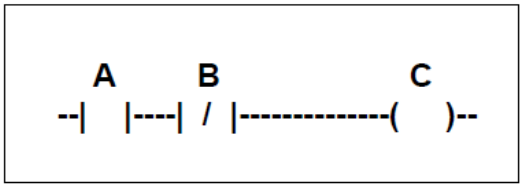
Функциональные блок-схемы в PC WORX редактируются в графических рабочих листах, программист может разрабатывать каждую блок - схему индивидуально.

2. *Список инструкций (IL)*:
   a) ассемблерная модель программирования, использует один аккумулятор;
   b) в одной строке прописывается одна команда, например, сохранение значения в аккумулятор.

```
LD    A
ANDN  B
ST    C
```

Поскольку это один из двух текстовых языков программирования согласно IEC 61131, то список инструкций в PC WORX программируется в свободном редакторе так же, как и графические языки, что приводит к высокой степени свободы для программиста, особенно когда дело доходит до структуры.

3. *Язык релейных диаграмм (LD)*:
   a) стандартизированный, ограниченный набор символов для программирования релейных систем управления;
   b) язык, основанный на североамериканском стиле программирования, подобном стандарту US для рисования принципиальных электрических схем.



Работа с релейными схемами в PC WORX строится так же, как при работе с функциональными блок-схемами – в свободном графическом

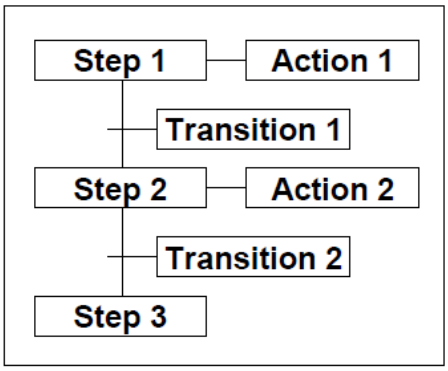
редакторе. Редактор – один и тот же инструмент, который учитывает высокую степень комбинации обоих языков программирования.

4. *Язык структурированного текста (ST)*:
   a) высокоуровневый язык, структурированный через подпрограммы;
   b) синтаксис, подобный Паскалю;
   c) сложные и вложенные инструкции.

```
C := A AND NOT B;
```

В дополнение к стандартным программным элементам структурированный текст предлагает преимущества высокоуровневого программирования посредством сложных инструкций.

5. *Язык последовательных функциональных схем (SFC)*:
   a) графический язык программирования для того, чтобы описать поведение последовательности управляющих программ;
   b) язык, используемый для того, чтобы структурировать проблемы управления;
   c) ясно расположенный язык программирования, учитывает быструю диагностику;
   d) элементы Bacic: шаги с блоками действия и переходами;
   e) поддерживает альтернативные и параллельные последовательности.



Поскольку предлагается возможность структурировать процессы уже через структуру языка, то у языка последовательных функциональных схем есть специальная позиция среди языков программирования, описанных в IEC 61131.

2.2 Программирование в PC WORX. Организационные модули программы (POU)

Если вставленная автоматически через шаблон проекта основная программа не должна использоваться, то первый шаг для начала программирования – вставка POU (Организационные модули программы) типа программы. Выбирается элемент Logical POUs или существующий POU, затем вставляется через контекстное меню или строку меню (рисунок 2.2).

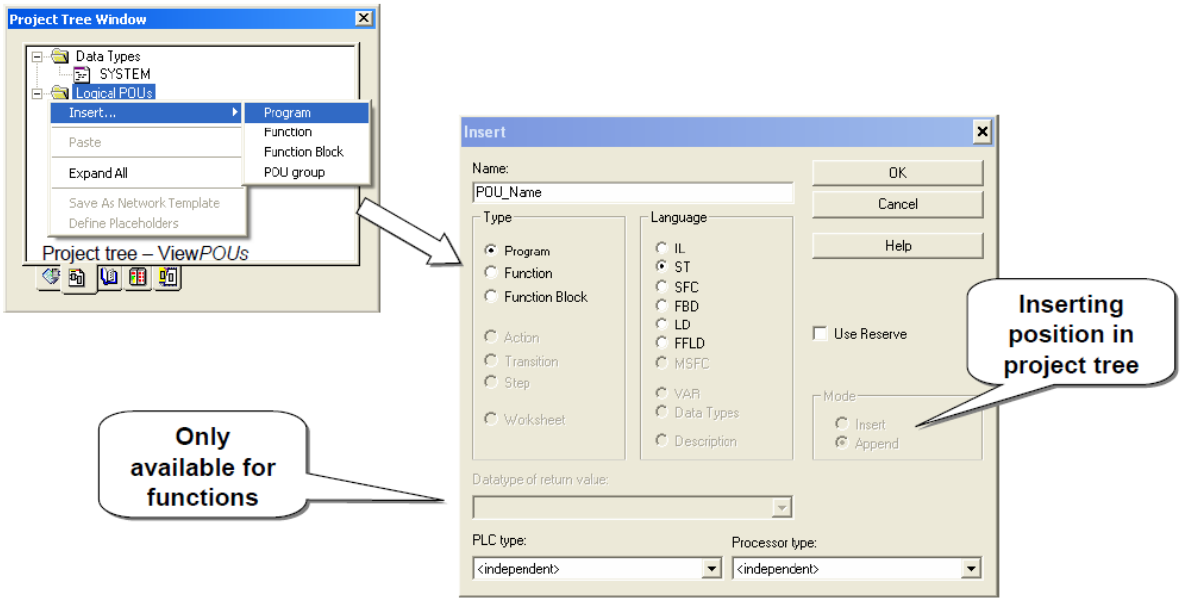


Рисунок 2.2 – Этапы вставки POUs

В зависимости от элемента, который будет вставлен (тип фрейма), становятся доступными несколько опций. Например, не все языки программирования могут быть выбраны для каждого типа POU (например, функции не поддерживают язык последовательных функциональных схем).

Опция *Datatype* возвращаемого значения является активной только для функций. Как уже описано ранее, у функций есть только один выходной параметр. Тип данных этого параметра определен через выбор функции, его имя – через имя POU.

Выбор <*independent*> для типа PLC и типа процессора должен применяться, если используются специфичные функции и

функциональные блоки для PLC или процессора. При использовании этой опции можно управлять POU на PLC или процессоре установленного типа.

Примечание – Позиция POU в дереве проекта не влияет на порядок выполнения. Вызов программы в форме экземпляра программы через задачу или вызов функций и функциональных блоков в рабочих листах кода определяет последовательность программы.

За исключением языка, свойства POU могут быть настроены в диалоговом окне свойств POU (рисунок 2.3).

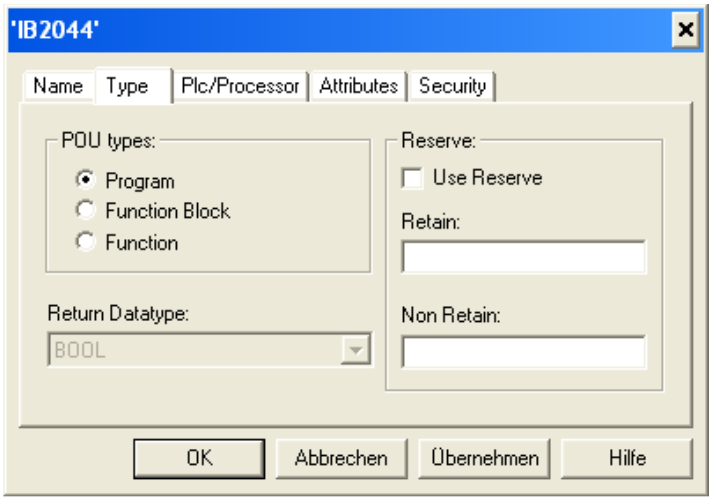


Рисунок 2.3 – Окно свойств POU

Изменение типа POUs возможно, но в зависимости от частоты использования модулей, что может приводить к затратам по адаптации для проекта. Резерв обращается к памяти, зарезервированной для POU во время загрузки для последующих изменений. Как стандарт, ресурс (аппаратная структура) пути установлен так, чтобы память была доступна для каждого POU и не требуется отдельный резерв.

Атрибут «только для чтения» используется для многопользовательской функции, но может также использоваться в качестве защиты от ненамеренных изменений.

Настройки безопасности для POU учитывают защиту экспертизы для программирования. Защита активирована только после введения пароля через меню *File → Enter password*.

Чтобы сохранить созданные самостоятельно организационные модули программы ясно расположенными в проекте, рекомендуется использовать группы POU, как только определенный размер проекта был достигнут (рисунок 2.4).

Группы POU не влияют на распределение работы по программированию. Они служат только средством для улучшения организации и могут быть созданы в столь же сложном порядке, как нравится программисту.

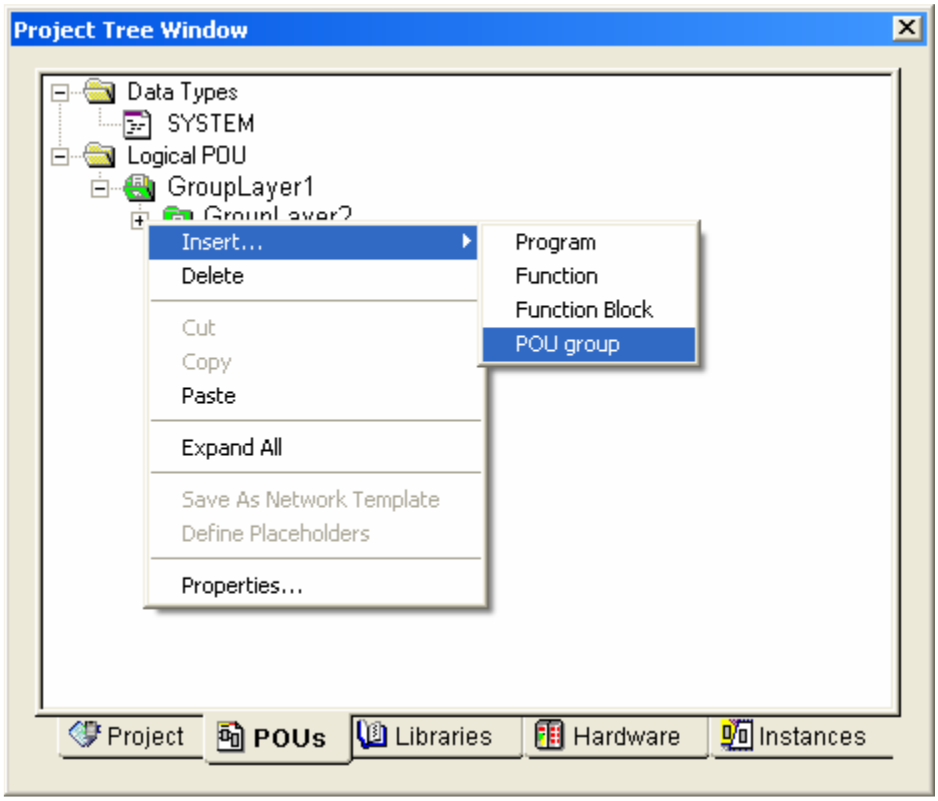


Рисунок 2.4 – Группы POU

Группы POU всегда перечисляются в начале списка логической папки POU и появляются в алфавитном порядке на одном уровне.

Стандартные функции

После установки PC WORX доступно множество функций. Они могут быть просмотрены через *Edit wizard* (Мастер редактирования), где представлено большое разнообразие типов данных.

Детали относительно соединения и функций блоков могут быть получены из отдельной справки HTML для каждого блока в PC WORX.

Рисунок 2.5 – Стандартные функции

Стандартные функциональные блоки

IEC 61131 определяет функциональные блоки, которые могут быть разделены на четыре группы: оценка переходов (скачков), флипфлопы, счетчики, таймеры (рисунок 2.6). Что касается функций, т. е. отклонения от блоков классических систем программирования PLC. Одним из

примеров этого является тот факт, что триггер RS является доминирующим при сбросе, а в триггере SR, доминирующем при установке.

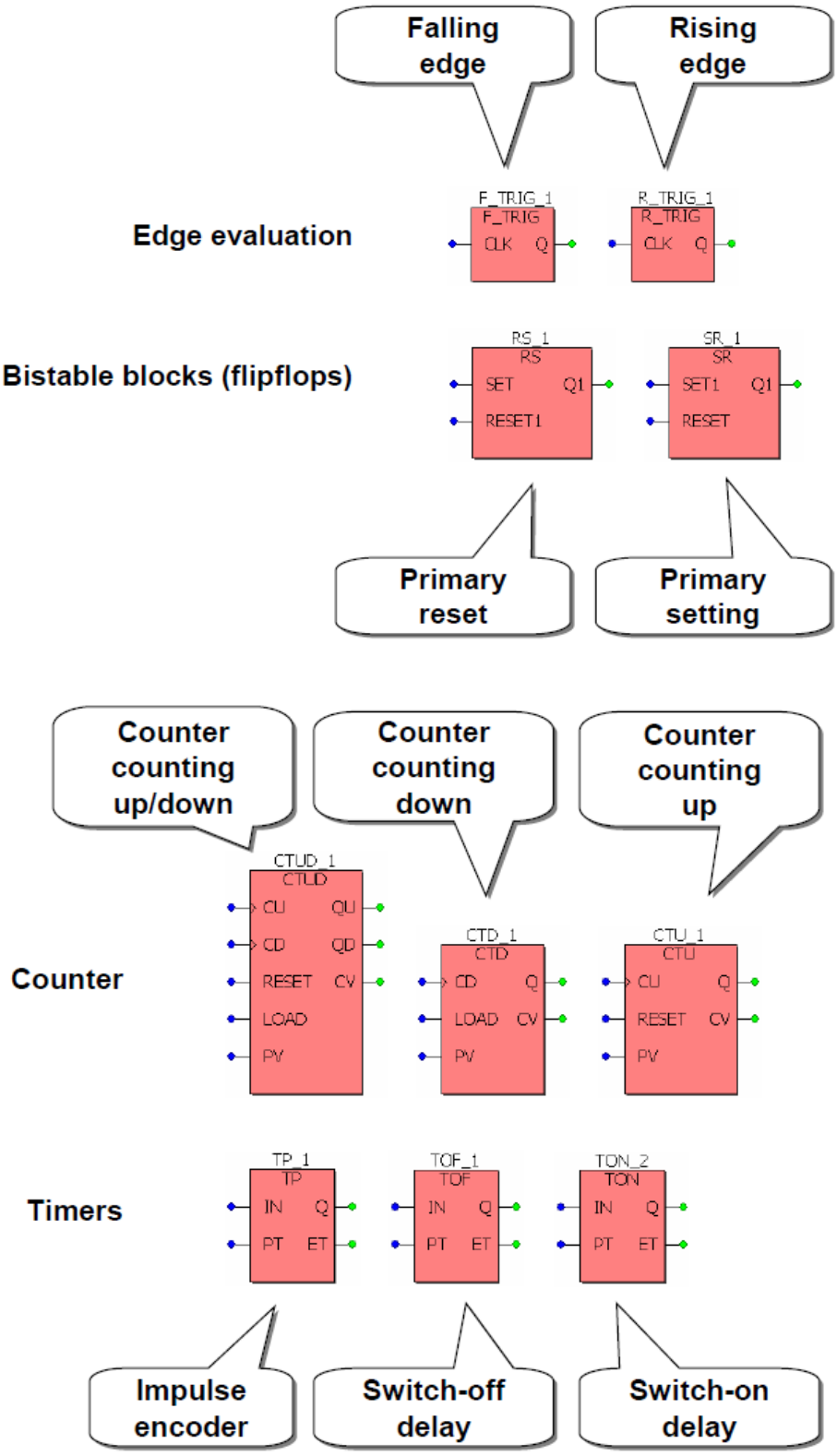


Рисунок 2.6 – Стандартные функциональные блоки

Дополнительная информация о проводном соединении, функциях функциональных блоков и функциях вообще может быть получена через справку HTML PC WORX.

2.3 Язык FBD (функциональных блок-схем)

Рисунок 2.7 дает краткий обзор элементов языка, доступных в функциональных блок-схемах. В основном это – блоки и переменные.

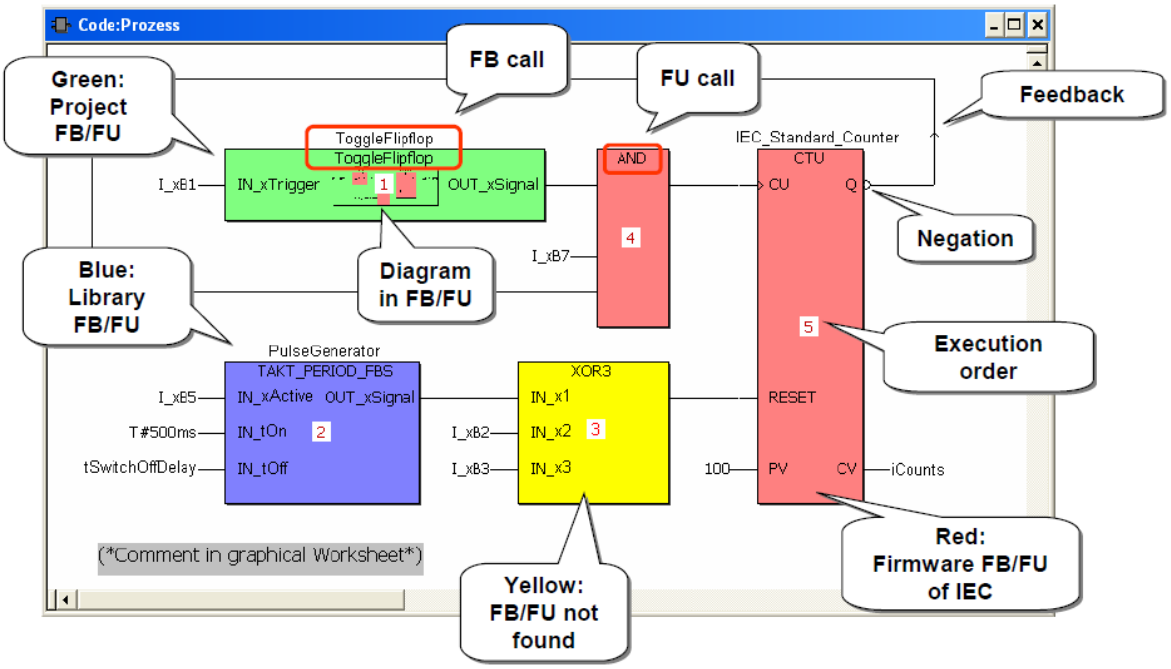


Рисунок 2.7 – Краткий обзор элементов языка, функциональных блок-схем

Вставка переменной в FBD выполняется через переменное диалоговое окно, которое предлагает доступ к локальным и глобальным переменным.

В свободном редакторе FBD сначала должна быть установлена метка вставки или же точка подключения функции или функционального блока должна быть отмечена прежде, чем диалоговое окно можно показать.

Диалоговое окно может быть активировано через контекстное меню, как показано на рисунке 2.8, или через кнопку функции *F5* (настройка по умолчанию).

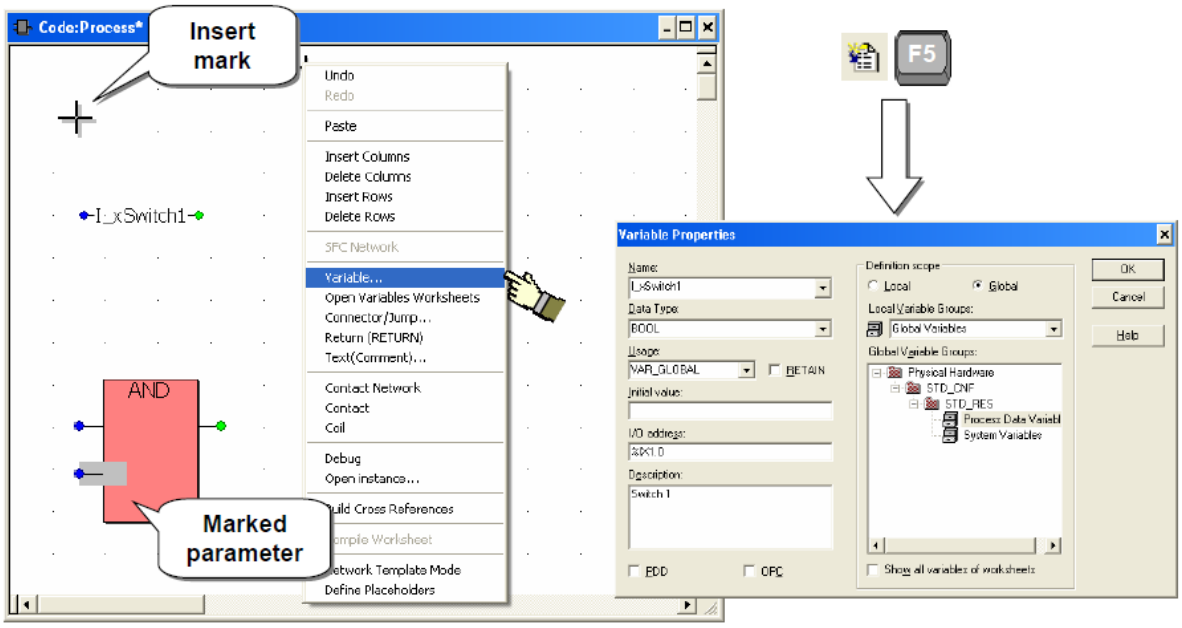


Рисунок 2.8 – Вставка переменных в языке функциональных блок-схем

Чтобы вставить функции, прежде всего, должна быть установлена метка вставки (рисунок 2.9). Если имя функции известно, то можно переместиться в списке, вводя первую букву. Сама вставка выполняется через двойной щелчок по требуемой функции.

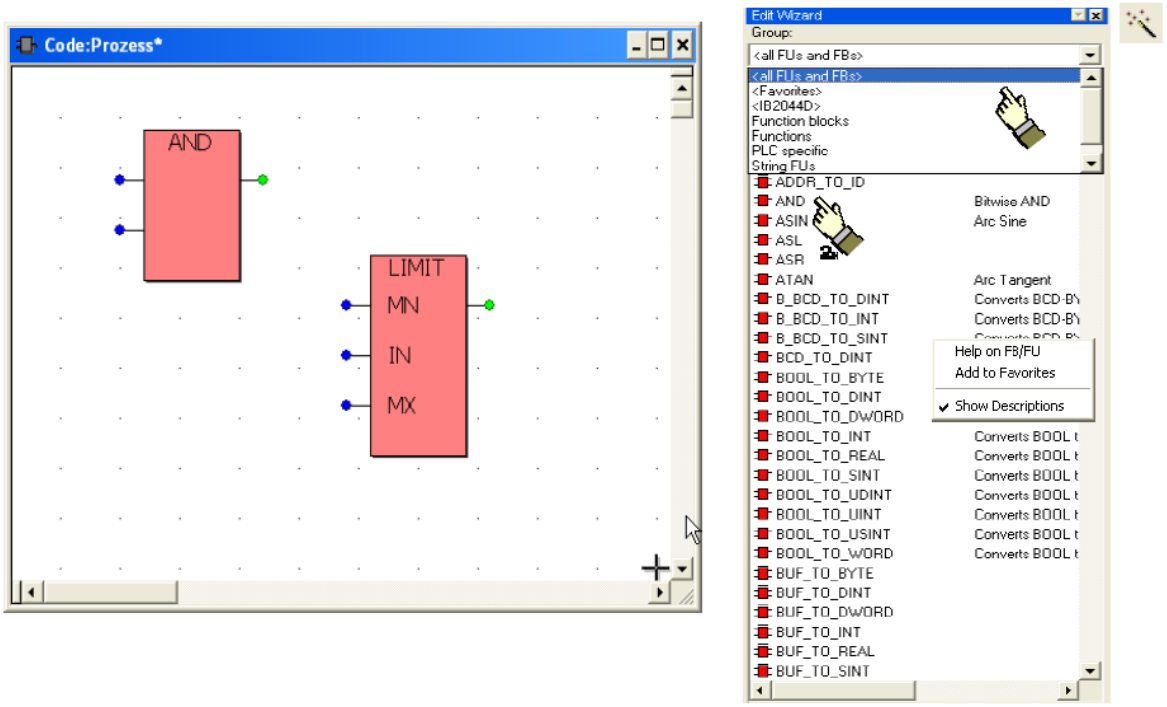


Рисунок 2.9 – Вставка функций в языке функциональных блок-схем

Примечания

1) Для каждого стандартного блока доступна справка HTML, которую можно вызвать через контекстное меню;
2) Через кнопку или стандартное сочетание клавиш *Shift+F2*, может быть показан/скрыт мастер редактирования.

Вставка функциональных блоков сделана таким же образом, как вставка функций. Однако, прежде чем вставить функциональный блок, его экземпляр должен быть объявлен. Как имя экземпляра, PC WORX предлагает имя функционального блока плюс инкремент, соединенный подчеркиванием. Имя может быть задано пользователем индивидуально, но желательно, чтобы оно отражало функцию, которую выполняет функциональный блок.

Для каждого стандартного блока доступна справка HTML, которую можно вызвать через контекстное меню.

На рисунке 2.10 показан пример диалогового окна вставки проекта, в котором была уже создана группа локальной переменной для экземпляров FB. Иначе, была бы доступна только группа по умолчанию.

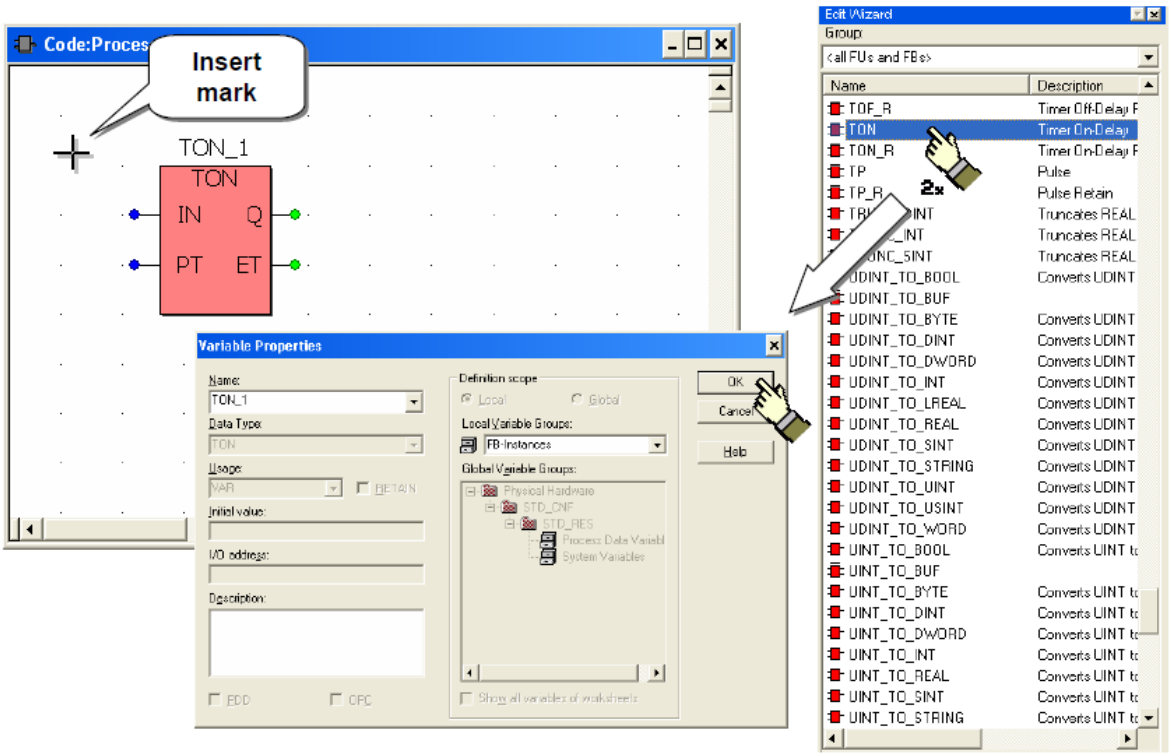


Рисунок 2.10 – Вставка функциональных блоков в языке функциональных блок-схем

На рисунке 2.11 представлены различные возможности для того, чтобы отредактировать блоки (функции и функциональные блоки):

1) добавление входных параметров (обычно только поддерживаемых операторами);
2) инвертирование входных и выходных параметров, основанных на бите (не поддерживается для всех блоков). Когда параметр выбран, показанные сверху кнопки включены;
3) диалоговое окно может быть активировано через двойной щелчок для стандартных блоков и через пункт меню *Properties* контекстного меню для блоков, определяемых пользователем;
4) замена блока может быть сделана через маркировку замещаемого блока и вставку нового. Если заменяемый блок интегрирован в программу, то такие изменения могут привести к графическим ошибкам.

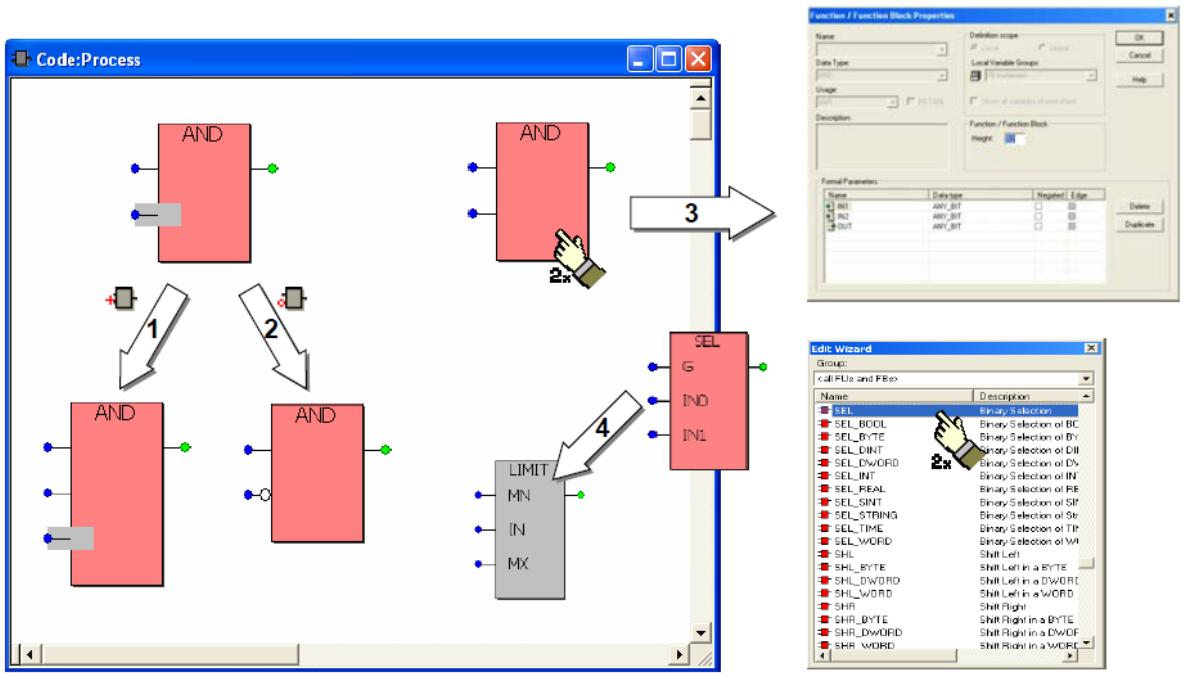


Рисунок 2.11 – Редактирование функциональных блоков

2.4 Созданные пользователем функции и функциональные блоки

Создание функций

Вставка функции в дерево проекта возможна, когда отмечены логическая папка POUs или группа POU (рисунок 2.12).

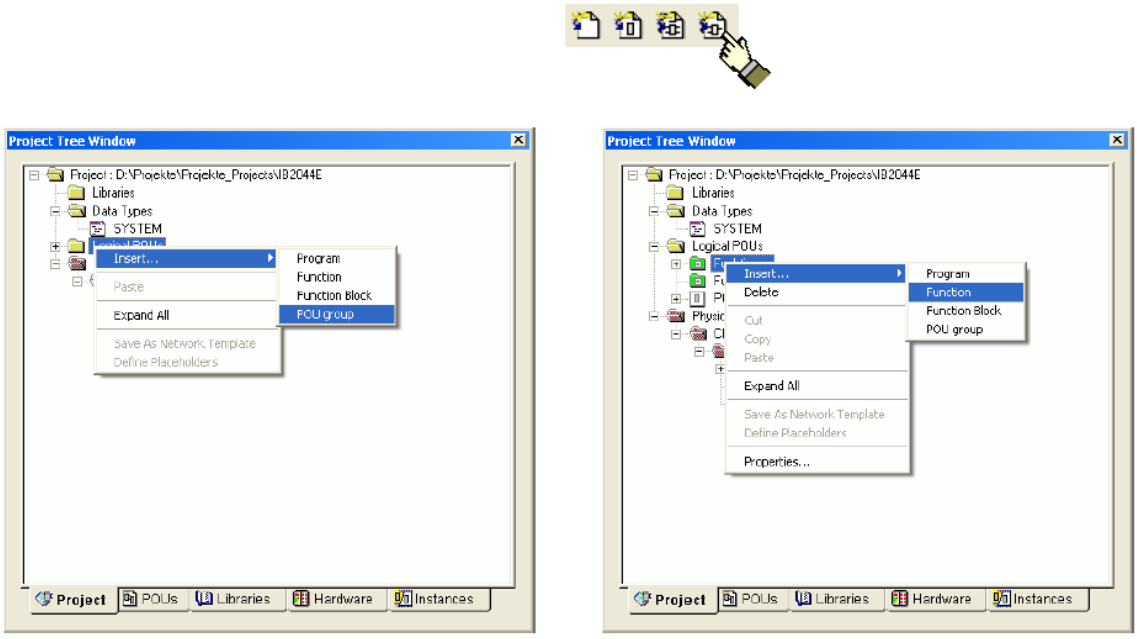


Рисунок 2.12 – Вставка функций в дерево проекта

В зависимости от размера проекта, имеет смысл организовывать блоки, которые созданы пользователем, в группы POU. Эти блоки также могут быть добавлены в группы позже.

Чтобы вставить функцию, должны быть отредактированы следующие параметры (рисунок 2.13):

1. Имя блока функции будет перечислено в списке для выбора в *Edit wizard*. Кроме того, это имя будет использоваться в качестве имени возвращаемого значения для внутреннего программирования блока. Имя должно соответствовать соглашениям для имен переменной.

2. Тип единственного выходного значения функции определен через тип данных возвращаемого значения.

3. Должны быть определены PLC и тип процессора, если нужно получить соответствующий доступ к определенным блокам для выбранного типа. Это маловероятно, поскольку большинство определенных блоков – функциональные блоки и, таким образом, не могут быть вызваны.

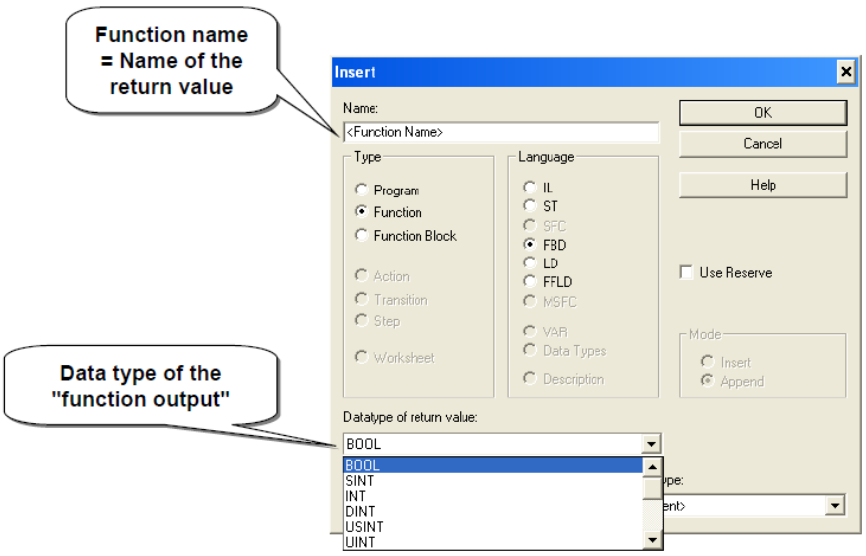


Рисунок 2.13 – Окно настройки свойств функции

После того, как функция была вставлена в дерево проекта, она перечислена с выбранным именем и именами для трех основных элементов, которые основаны на имени (рисунок 2.14):

1) <Имя функции> T для листа комментариев;
2) <Имя функции> V для таблицы переменных;
3) <Имя функции> для первого рабочего листа кода.

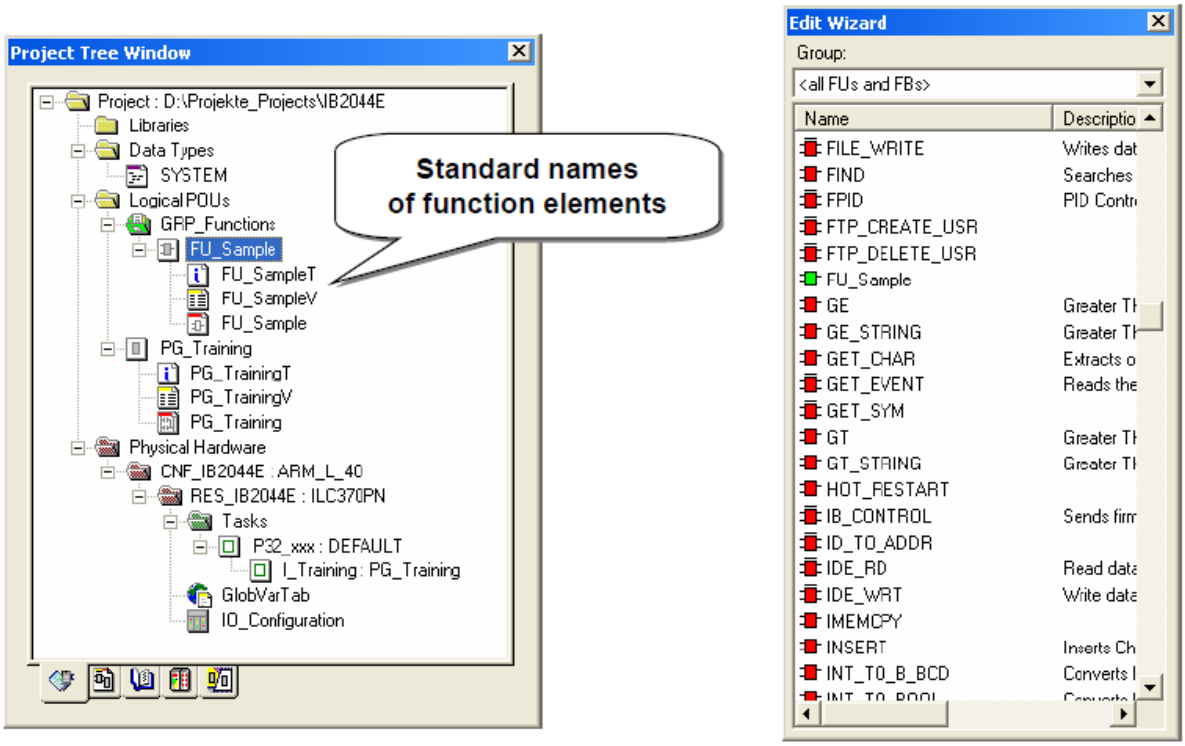


Рисунок 2.14 – Функции в дереве проекта

В дополнение к этому в *Edit wizard* функция обнаруживается в группах <*all FUs and FBs*> и в группе с именем текущего проекта.

Примечание – Для создаваемых пользователем функций нет записи в группах <*Functions*>.

Звездочки после переменной таблицы и рабочего листа кода указывают, что эти элементы еще не были скомпилированы.

Редактирование функций

Сама функция редактируется в ее рабочем листе (ax) согласно стандартной процедуре программирования:

1. Как показано на рисунке 2.15, должен быть объявлен, по крайней мере, один входной параметр.

2. Чтобы сделать доступным для проекта значение, вычисленное посредством функционального алгоритма, должно использоваться возвращаемое значение (имя = имя функции).

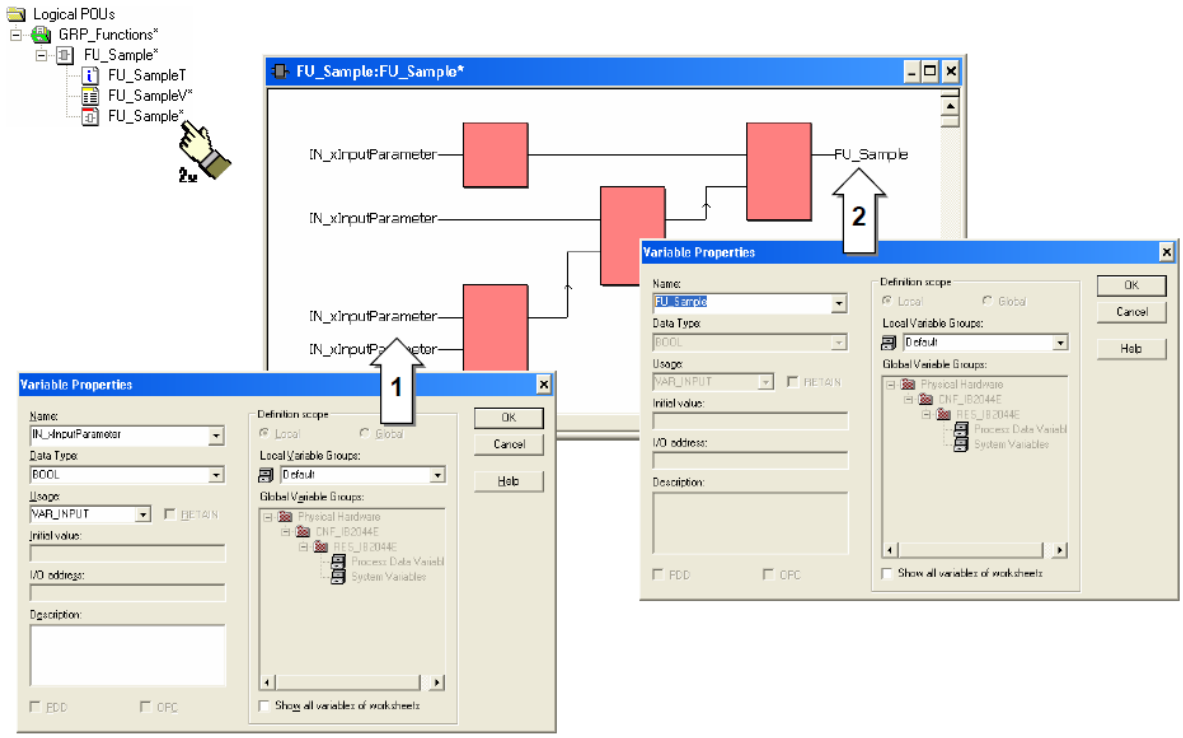


Рисунок 2.15 – Окно редактирования функций

Таблица переменных будет скомпилирована при закрытии окна. Рабочие листы кода могут быть скомпилированы при закрытии или явном использовании соответствующей кнопки или сочетания клавиш *Shift+F9*. После успешной компиляции таблицы переменных функция может быть вызвана для использования в другом POU (рисунок 2.16).

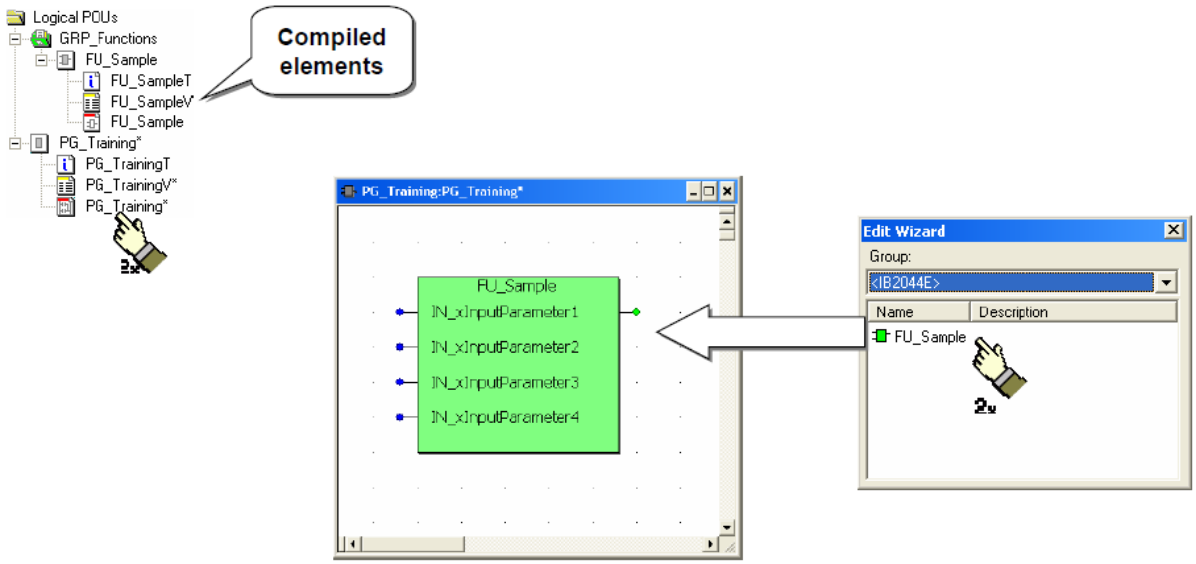


Рисунок 2.16 – Окно работы с функцией

Создание функциональных блоков

Вставка функционального блока в дерево проекта осуществляется при открытой логической папке POUs или группе POU (рисунок 2.17).

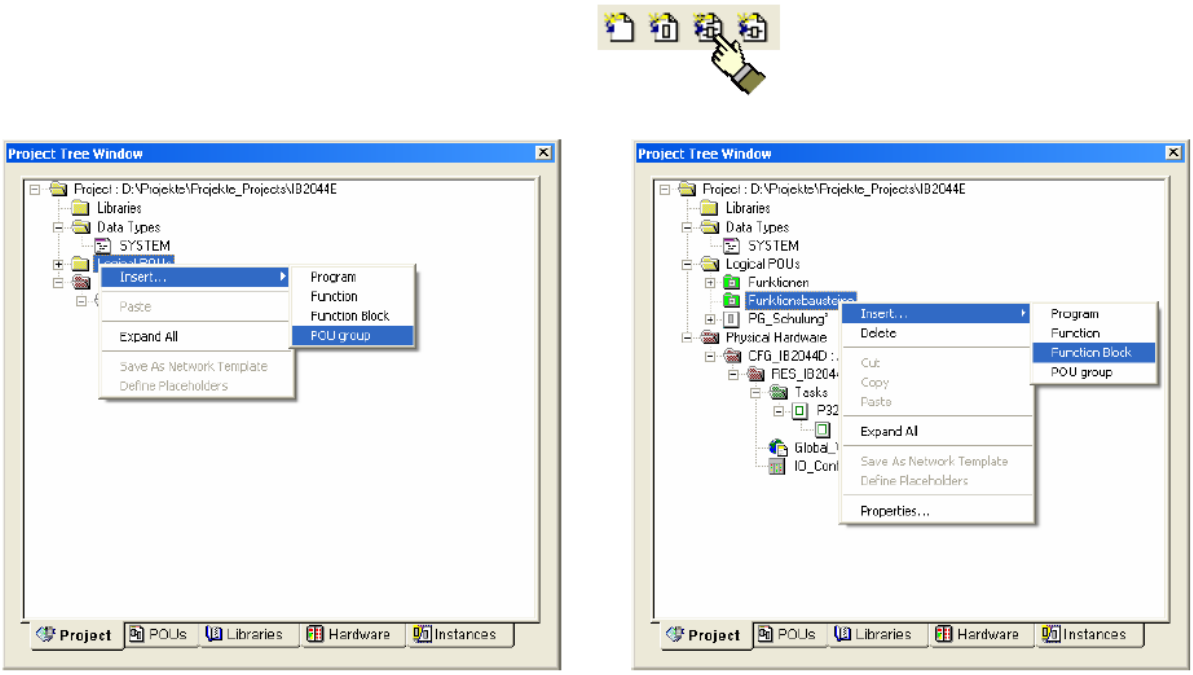


Рисунок 2.17 – Вставка в дерево проекта функциональных блоков

В зависимости от размера проекта иногда имеет смысл организовывать созданные блоки в группы POU. Эти блоки могут также быть добавлены к группам позже.

Для вставки функциональных блоков доступно большее разнообразие языков, чем для вставки функций (рисунок 2.18). Тип данных возвращаемых значений может быть не определен.

Имя функционального блока будет перечислено для выбора в окне *Edit wizard*. Имя не используется для программирования FB, оно должно соответствовать соглашениям для имен переменной.

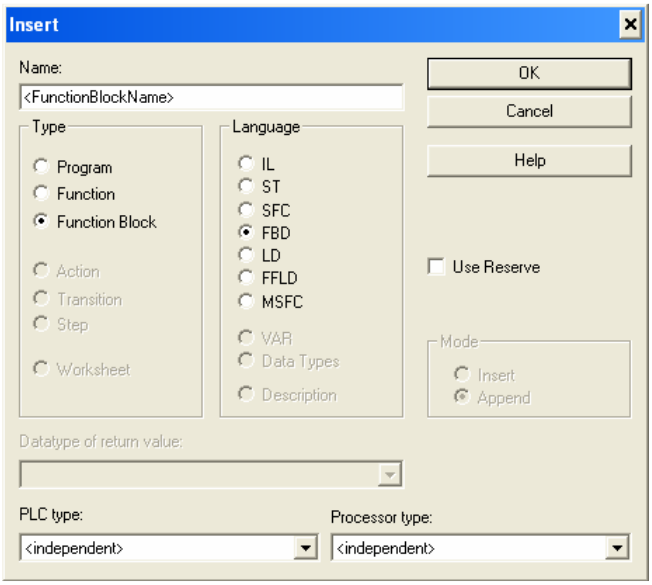


Рисунок 2.18 – Окно настройки свойств функциональных блоков

После того, как функциональный блок был вставлен в дерево проекта, он перечислен с выбранным именем и именами для трех основных элементов, которые основаны на имени (рисунок 2.19):

1) <Имя функционального блока> T для листа комментариев;
2) <Имя функционального блока> V для таблицы переменных;
3) <Имя функционального блока> для первого рабочего листа кода.

В дополнение к этому в *Edit wizard* функция обнаруживается в группах *<all FUs and FBs>* и в группе с именем текущего проекта.

Примечание – Для создаваемых пользователем функциональных блоков нет никакой записи в группе < *Functions block* >.

Звездочки после переменной таблицы и рабочего листа кода указывают, что эти элементы еще не были скомпилированы.

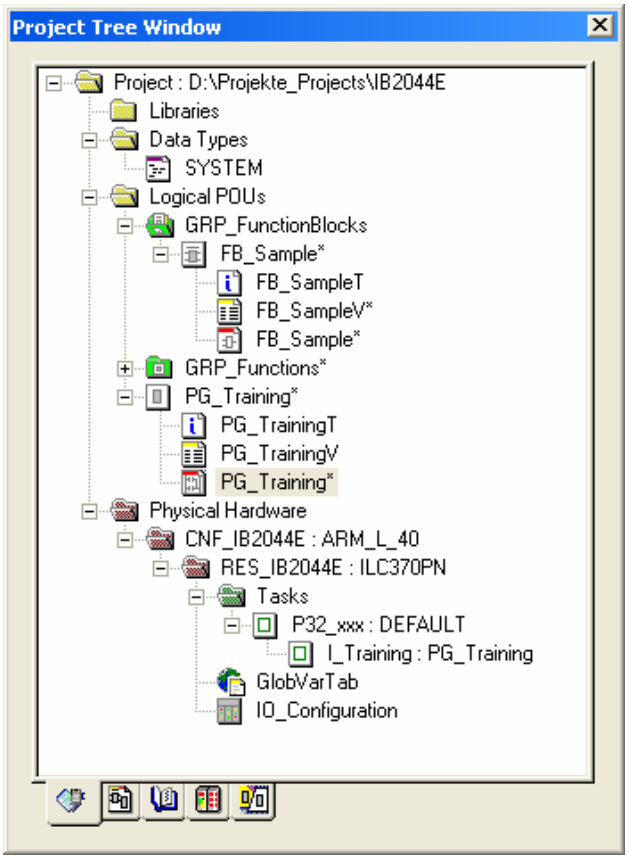


Рисунок 2.19 – Функциональные блоки в дереве проекта

Редактирование функциональных блоков

Функциональный блок редактируется в его рабочем листе (ax) согласно стандартной процедуре для программирования (рисунок 2.20 и 2.21). В отличие от функций, есть значительно более высокая степень свободы относительно проекта. Входные 1 и выходные 2 параметры могут быть объявлены в любом количестве, а также функциональные блоки учитывают доступ к глобальным переменным 3.

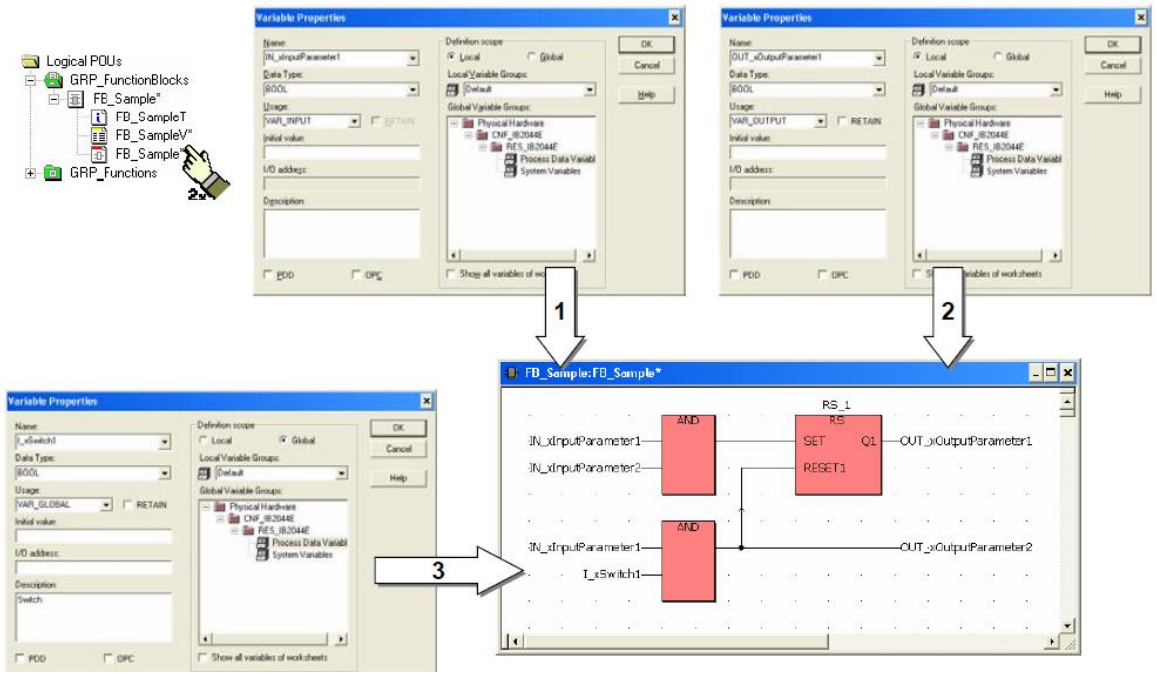


Рисунок 2.20 – Окно редактирования функциональных блоков

Внимание! Следует помнить, что имя функционального блока не должно использоваться в качестве имени переменной.

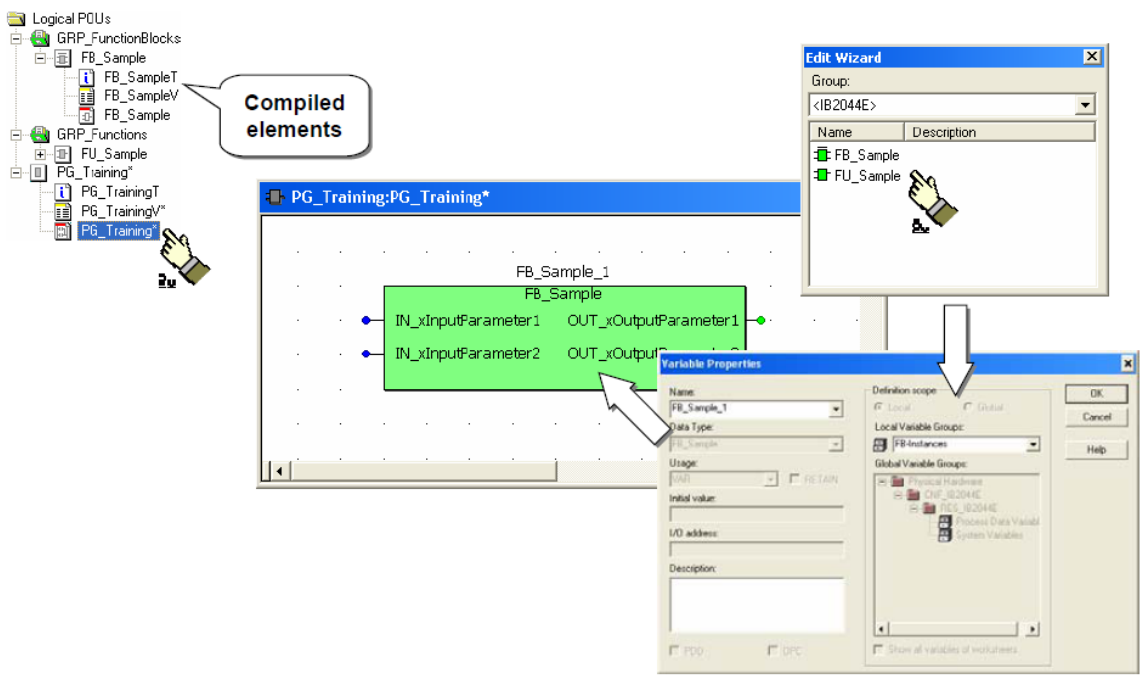


Рисунок 2.21 – Окно работы с функциональным блоком

Таблица переменных будет скомпилирована при закрытии окна. Рабочие листы кода могут быть скомпилированы при закрытии или явном

использовании соответствующей кнопки или сочетания клавиш *Alt+F9*. После успешной компиляции таблицы переменных функция может быть вызвана для использования в другом POU.

2.5 Язык IL (список инструкций)

Элементы языка списка инструкций

Доступ к элементам языка IL главным образом реализуется через *Edit wizard* (Мастер редактирования) (рисунок 2.22). В дополнение к функциям и функциональным блокам в листе инструкций доступна группа операторов (рисунок 2.23).

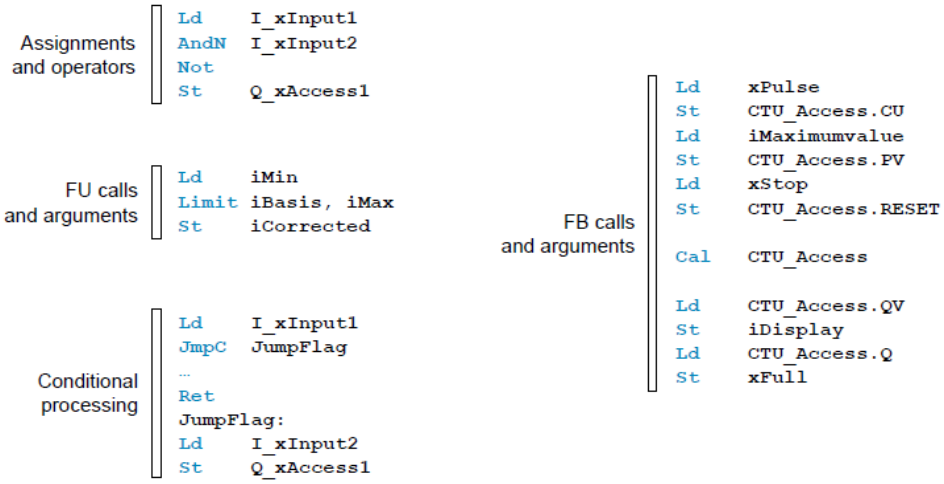


Рисунок 2.22 – Элементы языка для списка инструкций

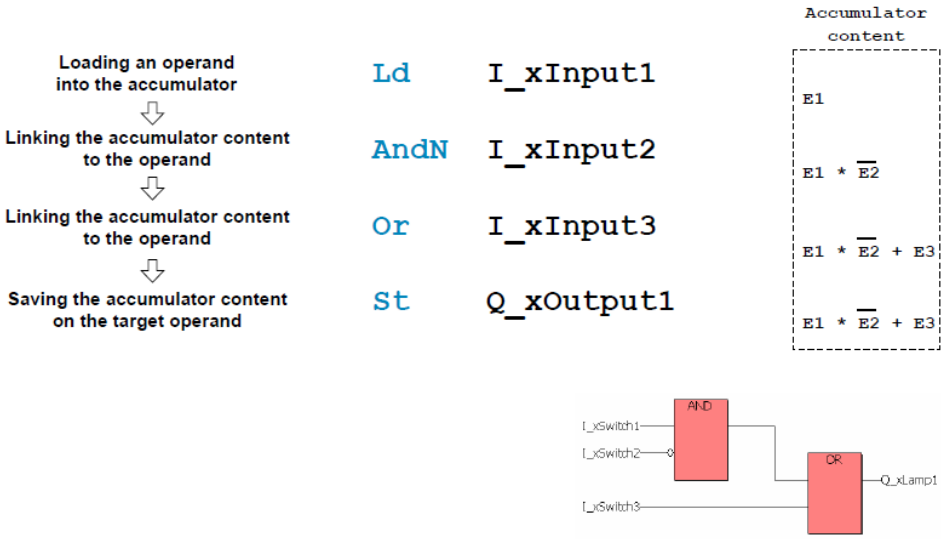


Рисунок 2.23 – Этапы присвоения данных операндам

Список инструкций в PC WORX использует аккумулятор и универсальную пару загрузки и сохранения команд для всех типов данных.

Операторы

На рисунке 2.24 представлен краткий обзор переменных типов данных, которые могут также использоваться с группами операторов. Исключения составляют дополнение *NOT*, функциональный блок *RETURN* и закрывающаяся скобка, которая может использоваться без операнда в той же самой строке.

В классическом программировании PLC операторы S и R записываются однократно в зависимости от значения булевой переменной (1 или 0), соединенной с операндом.

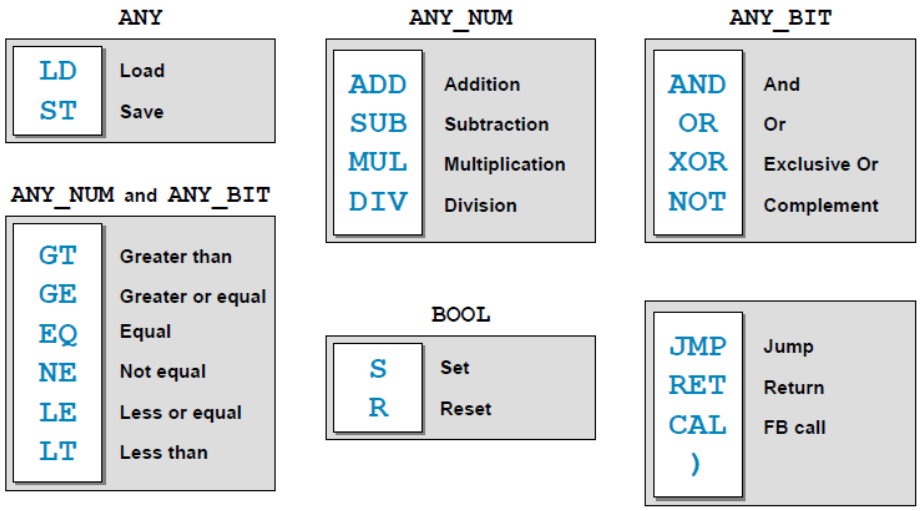


Рисунок 2.24 – Операторы в списке инструкций

Посредством модификатора N, операторы могут получить доступ к памяти (LD и ST), может быть реализован инвертированный доступ к основанным на бите операндам. То же самое актуально для операторов булевых операндов.

Скобки как модификатор приводят к приоритетной обработке кода в скобках (рисунок 2.25).

Примечание – Схема отображает проблему, которая возникает при инвертировании операндов, которые должны использоваться в выражении в скобках. Требуется преобразование уравнения согласно булевой логике.

```
LdN   I_xInput1
AndN  I_xInput2
And(  I_xInput3
Not
Or    I_xInput4
)
St    Q_xOutput1
```

$\overline{E1} * \overline{E2} = \overline{E1 + E2}$

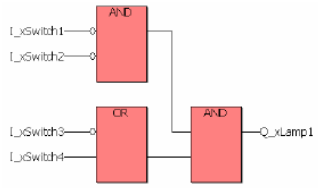


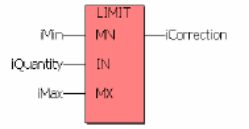
Рисунок 2.25 – Модификация операторов

Вызов функции и функционального блока

Инструкция вызова функции немного отличается от синтаксиса стандартных операций (рисунок 2.26). Первый параметр может быть загружен явно через LD или взят от аккумулятора.

```
Ld      iMin
Limit   iQuantity, iMax
St      iCorrection
```

1. Operand explicitely loaded




```
Ld      wMin
Word_To_Int
Limit   iQuantity, iMax
St      iCorrection
```

1. Operand from the accumulator

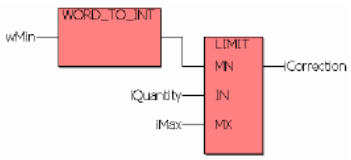


Рисунок 2.26 – Вызов функции

Различие проявляется в использовании функции с количеством входов больше одного. Абсолютные параметры после второго не записываются в отдельные строки, а перечисляются позади имени функции в правильном порядке и разделены запятыми.

Вызов функционального блока в списке инструкции выполняется как на всех других языках в три этапа (рисунок 2.27):

1. Условие значений для входных параметров (импорт данных).

2. Выполнение функциональных блоков, в случае необходимости, при использовании сохраненных данных (вычисление).

3. Сохранение расчетных значений через выходные параметры в созданных переменных (экспорт данных).

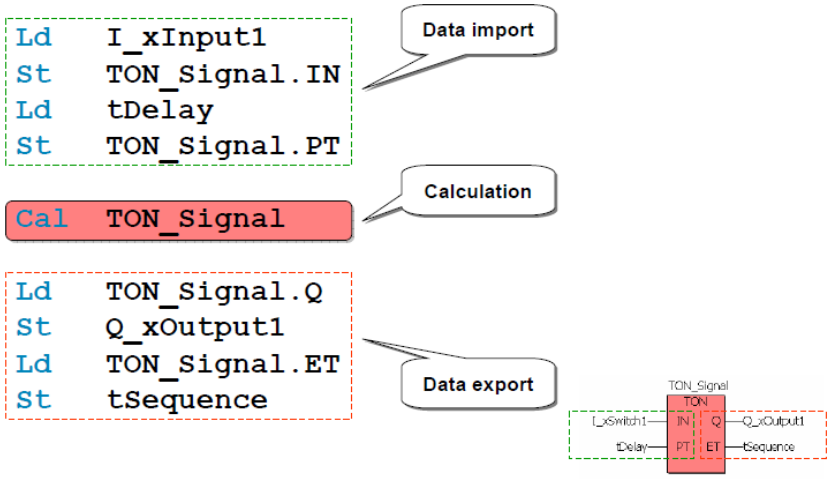


Рисунок 2.6 – Вызов функционального блока

Редактирование функции и функционального блока

Редактирование функции в PC WORX может быть выполнено вводом функционального шаблона синтаксиса посредством клавиатуры (рисунок 2.28).

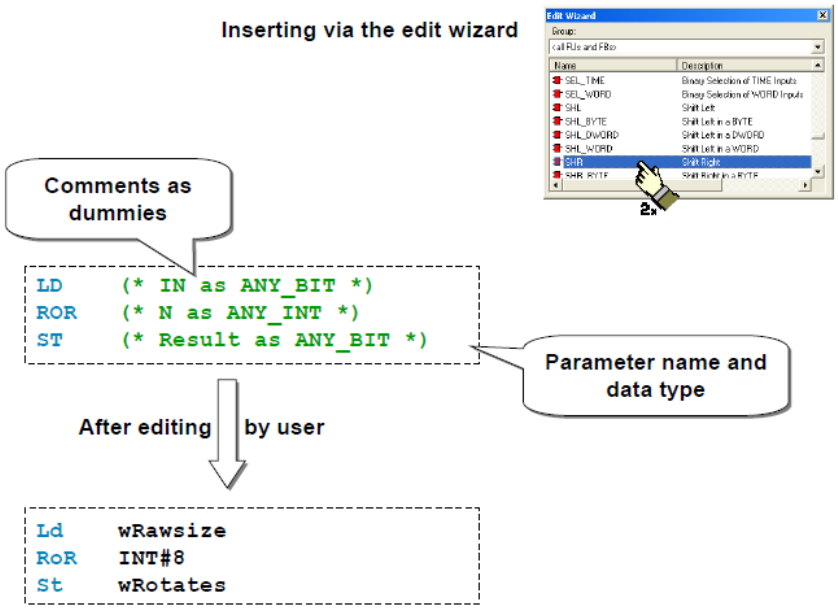


Рисунок 2.28 – Редактирование функций

Если параметры, функциональное написание или порядок аргументов неизвестны, шаблон синтаксиса может быть вставлен в программу через *Edit wizard* (Мастер редактирования). Передаваемые аргументы и переменная вставляются как комментарии с помощью *Edit wizard* и должны быть заменены переменными.

Примечание – Вложение функций не может быть выполнено с использованием *Edit wizard*.

Полезно просмотреть синтаксис пустых строк шаблона функции и затем на основании него написать программу.

Так же, как для функций, редактирование функционального блока в PC WORX может быть выполнено вводом синтаксиса FB через клавиатуру (рисунок 2.29). Следует иметь в виду, что функциональные блоки нужно инстанцировать. Это может быть сделано или вручную через таблицу переменных, или через диалоговое окно объявлением переменной. Тип FB определен как тип данных.

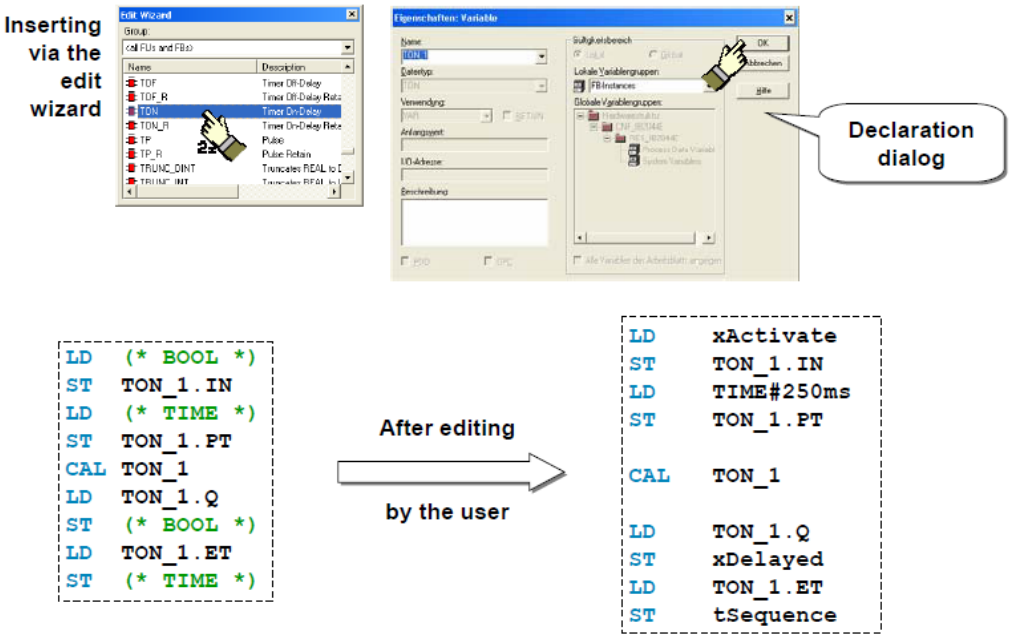


Рисунок 2.29 – Редактирование функционального блока

Если *Edit wizard* используется для добавления FB в программу, тогда выполняется объявление экземпляра через открываемое диалоговое окно объявления. Что касается функций, части синтаксиса вызова будут заменены переменными, вставленными как комментарии.

Вызов вложенных FB не может быть реализован через *Edit wizard*, который имеет возможность показать только синтаксис отдельного вызова.

Выполнение кода с условиями

Оператор перехода JMP и оператор окончания функционального блока RET позволяют отклоняться от стандартного порядка выполнения в списке инструкций. Для JMP должен быть определен адрес перехода. Маркер перехода может быть введен в той же строке что и оператор или, как показано на рисунке 2.30, помещено перед инструкцией.

Модификатор C учитывает условное выполнение операторов JMP, RET и CAL.

Модификатор N приводит к инверсии условия выполнения.

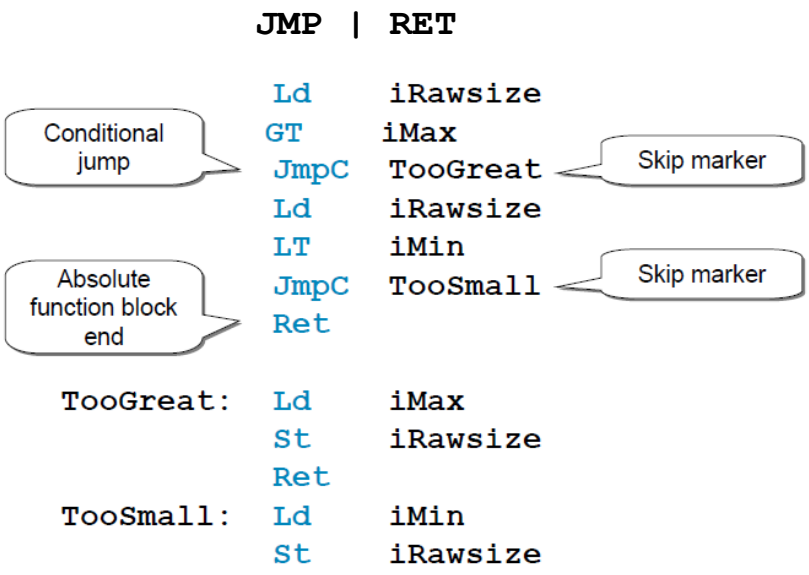


Рисунок 2.30 – Выполнение кода с условиями

Контрольные вопросы

1. Какие языки программирования используются в PC WORX? В чем их различия?
2. Можно ли осуществлять переход между языками?
3. Каким образом группы для POU влияют на программирование и обработку программ?
4. Назовите отличие функций от функциональных блоков.
5. Какие особенности отображения элементов языка функциональных блок-схем в рабочих листах?
6. Опишите процесс вставки функций и функциональных блоков в рабочие листы.
7. Как создается пользовательская функция?
8. Как можно отредактировать функциональные блоки?
9. Перечислите основные операторы языка списка инструкций.
10. Какие основные правила написания программного кода?
11. Чем отличается вызов функции от вызова функционального блока?
12. Опишите процесс редактирования функции, функционального блока.

Задания

Задание 1a. Написать программу на языке FBD

В рабочем листе PG_Course напишите программу, которая отвечает приведенным ниже требованиям.

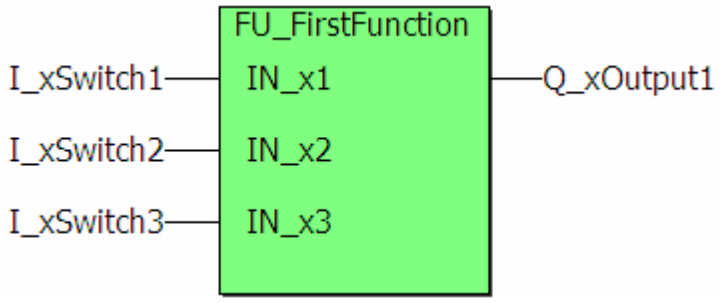
Комбинация	Переключатель1	Переключатель2	Переключатель3	Лампочка
1	True	True	True	True
2	False	True	True	True
3	True	False	True	True
4	False	False	True	True
5	True	True	False	False
6	False	True	False	True
7	True	False	False	False
8	False	False	False	False

Вместо набора имен в таблице используйте переменные, которые созданы автоматически и переименуйте их для присвоения вашего логического соединения.

Справка в *Context menu / Edit wizard / Appendix* (Контекстное меню / Мастер редактирования / Подсказка)

Подсказка: вместо того, чтобы использовать блок NOT, для некоторых блоков (например, для тех, где используется булева логика) входные и выходные параметры могут быть инвертированы.

Задание 1b. Добавить в проект функцию FU_FirstFunction, которая выполняет логику работы программы в задании 1a выше.



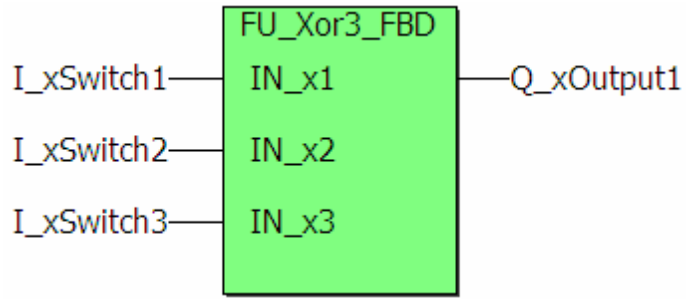
Добавьте группу *Function* (Функций POU) в ваше дерево проекта и в нее добавьте новую функцию с именем *FU_FirstFunction*, которая должна быть запрограммирована в схеме функционального блока (FBD).

Функция должна обеспечить логику предыдущей программы и вместо нее быть вызвана в программе.

Вместо глобальных теперь входные и выходные параметры функции должны использовать локальные переменные.

Примечание – Вместо того, чтобы использовать имена, например, Switch1, Switch2, при присвоении имен входных параметров функций и функциональных блоков, используют более общие названия, например, IN1, IN2. Имя необходимо выбирать основываясь на функции параметра, а не на размер процесса, который случайно соединен в пределах проекта.

Задание 2. Добавить в проект функцию FU_Xor3_FBD.



Добавьте функцию с именем *FU_Xor3_FBD* в дерево проекта. Она должна удовлетворять следующим требованиям:

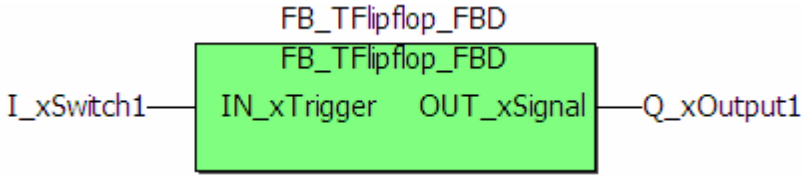
Комбинация	IN_x1	IN_x2	IN_x3	FU_Xor3_FBD
1	True	True	True	False
2	False	True	True	False
3	True	False	True	False
4	False	False	True	True
5	True	True	False	False
6	False	True	False	True
7	True	False	False	True
8	False	False	False	False

Примечание – справка в *Context menu / Edit wizard / Appendix* (Контекстное меню / Мастер редактирования / Приложение)

Подсказка: вместо того, чтобы использовать блок NOT, для некоторых блоков (например, для тех, где используется булева логика) входные и выходные параметры могут быть инвертированы.

Для систематической процедуры проверьте случаи, в которых выходной параметр должен иметь значения True и представьте эти случаи отдельно.

Задание 3. Добавить в проект функциональный блок FB_TFlipflop_FBD



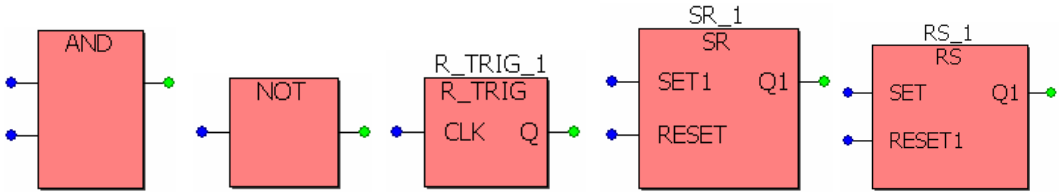
Добавьте группу *Function block* (Функциональные блоки POU) в дерево проекта. Затем, добавьте новый функциональный блок с именем *FB_TFlipflop_FBD* в эту группу.

Блок должен вести себя следующим образом: Если нарастающий фронт обнаружен во входном параметре IN, то выходной параметр OUT должен быть инвертирован.

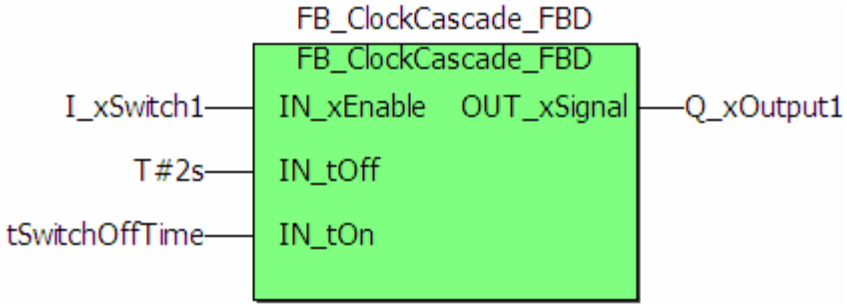
Примечание – справка в *Context menu / Edit wizard / Appendix* (Контекстное меню / Мастер редактирования / Приложение)

Подсказка: разделите задачу на две подзадачи и затем соедините их.

Блоки, которые могут быть полезными для создания, но не все они должны использоваться:



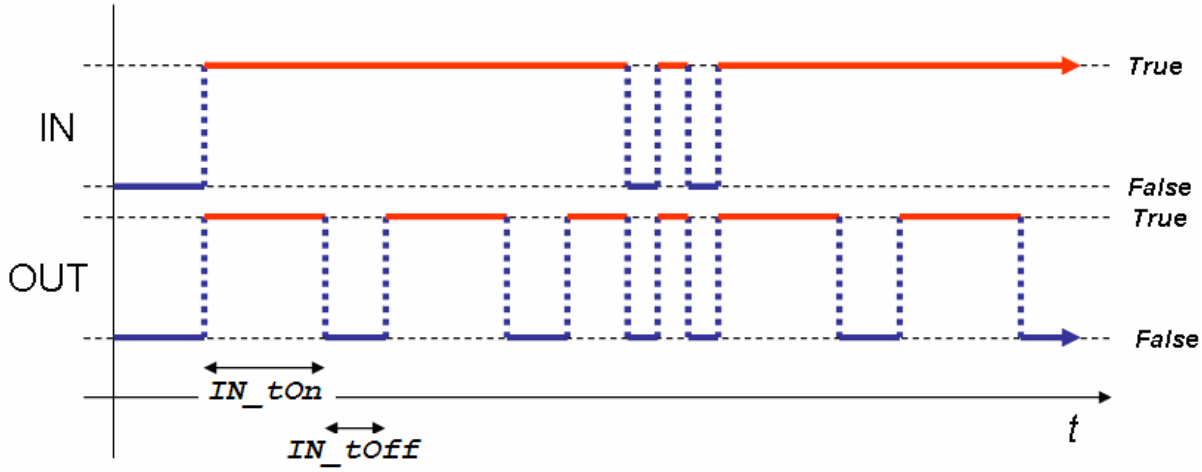
Задание 4. Добавить в проект функциональный блок FB_ClockCascade_FBD



Добавьте функциональный блок с именем *FB_ClockCascade_FBD* в дерево проекта.

Блок должен вести себя следующим образом: Если входные параметры IN будут установлены в True, то выходной параметр OUT должен сразу переключиться в True. Пока IN остается установленным в True, OUT остается в True для периода времени T_ON и затем переключается на False для периода времени T_OUT. Это переключение должно продолжаться до смены IN с True на False. В этом случае, OUT немедленно переключается в False.

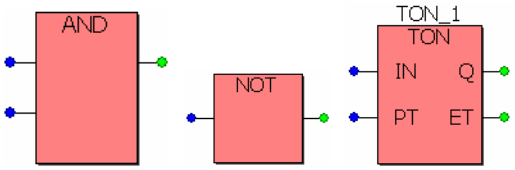
Следующая схема отображает это поведение:



Примечание – справка в *Context menu / Edit wizard / Appendix* (Контекстное меню / Мастер редактирования / Приложение)

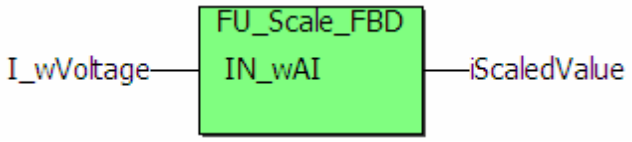
Подсказка: используйте временную диаграмму, чтобы создать функциональный блок по частям, т. е. изменение за изменением.

Блоки, которые могут быть полезными для создания:



Задание 5. Добавить в проект функции для обработки аналоговых значений, при программировании использовать язык FBD.

[a] Основные функции: FU_Scale_FBD



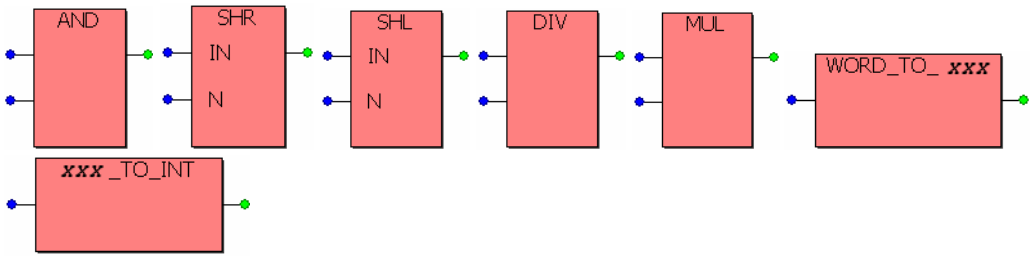
Добавьте функцию с именем *FU_Scale_FBD* в дерево проекта. Эта функция должна масштабировать аналоговое значение от доступного модуля аналогового входа согласно следующему образцу:

Значения аналогового входа	Дисплей	Масштабируемое значение
0-10V	x 12Bit-Analog Value x x x	0...100decimal

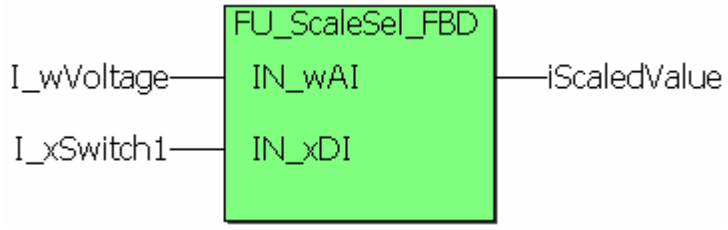
Примечание – справка в *Context menu / Edit wizard / Appendix Device data sheet* (Контекстное меню / Мастер редактирования / Приложение таблица данных устройства)

Подсказка: учтите, что для масштабирования требуется преобразование типов данных. Эти типы данных выбраны для использования арифметических функций и обеспечения диапазона значений, который достаточен для вычисления.

Блоки, которые могут быть полезными для создания, но не все они должны использоваться:



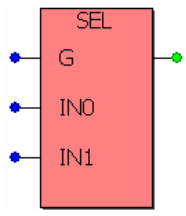
[b] Расширение двоичным выбором диапазона: FU_ScaleSel_FBD



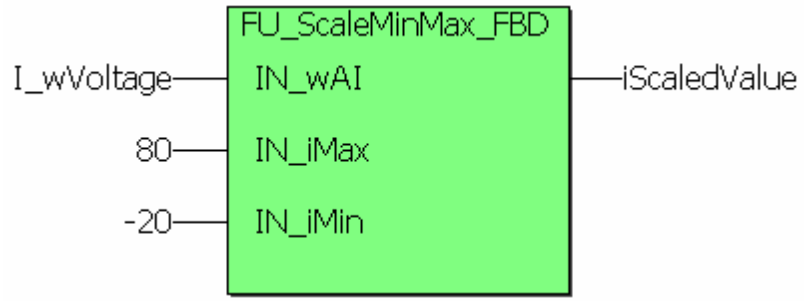
В дереве проекта создайте копию функции, запрограммированной в предыдущем задании. Поменяйте имя POU на *FU_ScaleSel_FBD* и добавьте входной параметр IN_xScale в программу. Таким образом, масштабируемое значение изменяется в пределах 1…100, если IN_xScale = False или в пределах 1…1000, если IN_xScale = True.

Значения аналогового входа	Масш-таб	Дисплей	Масштабируемое значение
0-10V	False	x 12Bit+Analog Value x x x	0…100decimal
0-10V	True	x 12Bit+Analog Value x x x	0…1000decimal

Дополнительный блок, который может бы быть полезным:



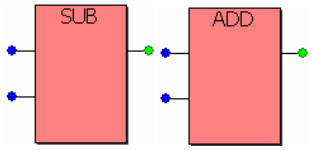
[c] Расширение для определяемого пользователем масштабирования: FU_ScaleMinMax_FBD



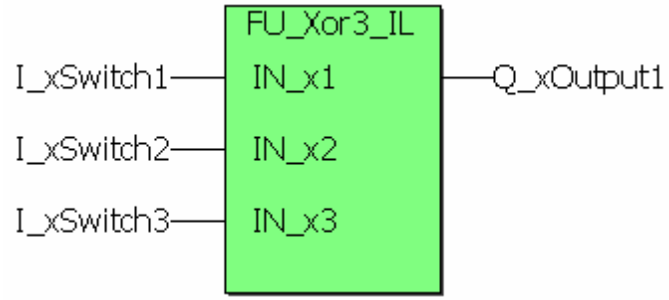
В дереве проекта создайте копию функции, запрограммированной в предыдущем задании. Поменяйте имя POU на *FU_ScaleMinMax_FBD* и добавьте два входных параметра IN_iMax и IN_iMin в программу. Они позволяют пользователю динамически корректировать верхнее значение и нижнее значение масштабируемого значения через определенные значения. Масштабирование должно быть линейным.

Значения аналогового входа	Min..Max	Дисплей	Масштабируемое значение
0-10V	-20…80	x 12Bit+Analog Value x x x	-20…80decimal

Дополнительные блоки, которые могут быть полезными:

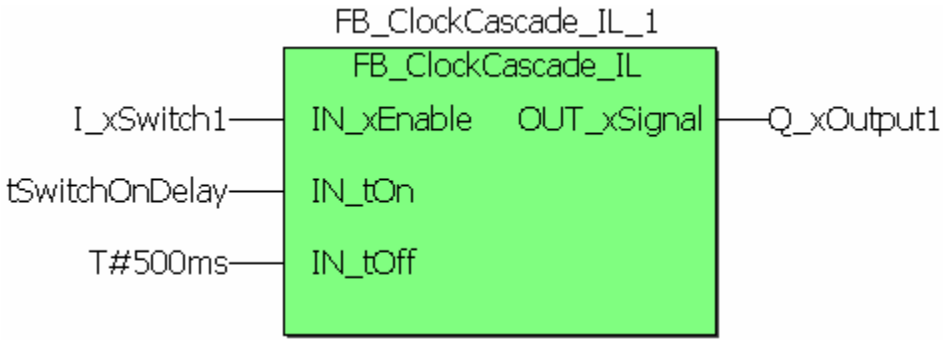


Задание 6. Добавить в проект функцию FU_Xor3_IL, при программировании использовать язык IL.



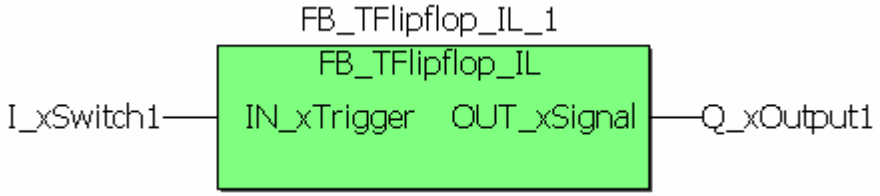
Добавьте функцию с именем *FB_Xor3_IL* в дерево проекта. Эта функция должна работать подобно *FB_Xor3_FBD* (см. *задание 2*).

Задание 7. Добавить в проект функциональный блок FB_ClockCascade_IL.



Добавьте функцию с именем *FB_ClockCascade_IL* в дерево проекта. Эта функция должна работать подобно *FB_ClockCascade_FBD* (см. *задание 4*).

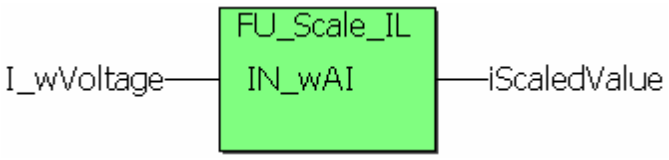
Задание 8. Добавить в проект функциональный блок FB_TFlipflop_IL



Добавьте функцию с именем FU_TFlipflop_IL в дерево проекта. Эта функция должна работать подобно *FU_TFlipflop_FBD* (см. *задание 3*).

Задание 9. Добавить в проект функции для обработки аналоговых значений, при программировании использовать язык IL.

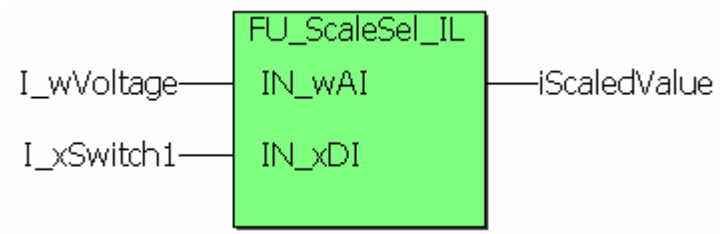
[a] Основные функции: FU_Scale_IL



Добавьте функцию с именем *FU_Scale_IL* в дерево проекта. Эта функция должна масштабировать аналоговое значение подобно *FU_Scale_FBD* (см. *задание 5a*).

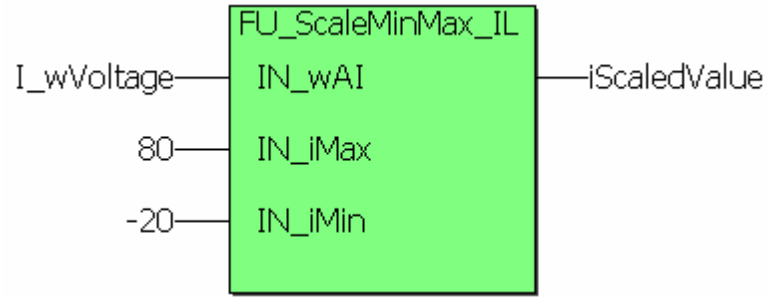
Подсказка: для операций, которые должны выполняться параллельно, необходимо использовать скобки и локальные переменные (маркеры) в списках инструкций

[b] Расширение двоичным выбором диапазона: FU_ScaleSel_IL



Добавьте функцию с именем *FU_ScaleSel_IL* в дерево проекта. Эта функция должна работать подобно *FU_ScaleSel_FBD* (см. *задание 5b*).

[c] Расширение для определяемого пользователем масштабирования: FU_ScaleMinMax_IL



Добавьте функцию с именем *FU_ScaleMinMax_IL* в дерево проекта. Эта функция должна работать подобно *FU_ScaleMinMax_FBD* (см. *задание 5c*).

Лабораторная работа №3. Язык релейных диаграмм. Язык последовательных функциональных схем. Пользовательские типы данных

Цель: изучение пользовательских типов данных, изучение принципов составления функциональных схем.

3.1 Релейно-контактные схемы

Основные элементы релейно-контактных схем (рисунок 3.1) могут быть добавлены и отредактированы через строку меню LD. Существующие программы могут быть также расширены и дополнены использованием этих кнопок. В зависимости от элемента, выбранного в рабочей области, будут доступны (активны) соответствующие кнопки.

Диалоговое окно для изменения свойств контакта/реле (рисунок 3.2) незначительно отличается от стандартного диалогового окна переменных. Дополнительно доступны элементы управления для настройки элемента LD. Кроме того могут быть выбраны только те типы данных, которые имеют доступ к булевому параметру.

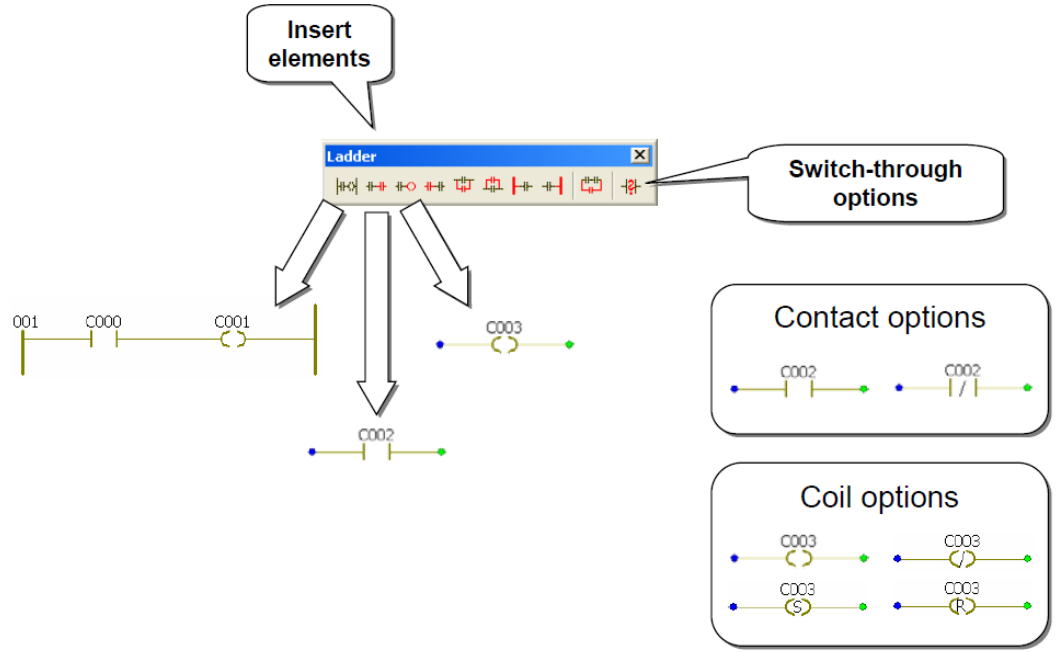


Рисунок 3.1 – Базовые элементы релейно-контактной схемы

В дополнение к типу данных *Bool* возможны другие битовые типы данных в случае неявного битового доступа, например *I_wFeedback.x13* –

доступ к 13-му биту входного слова *Feedback*. Могут также использоваться элементы определенного пользователем типа данных.

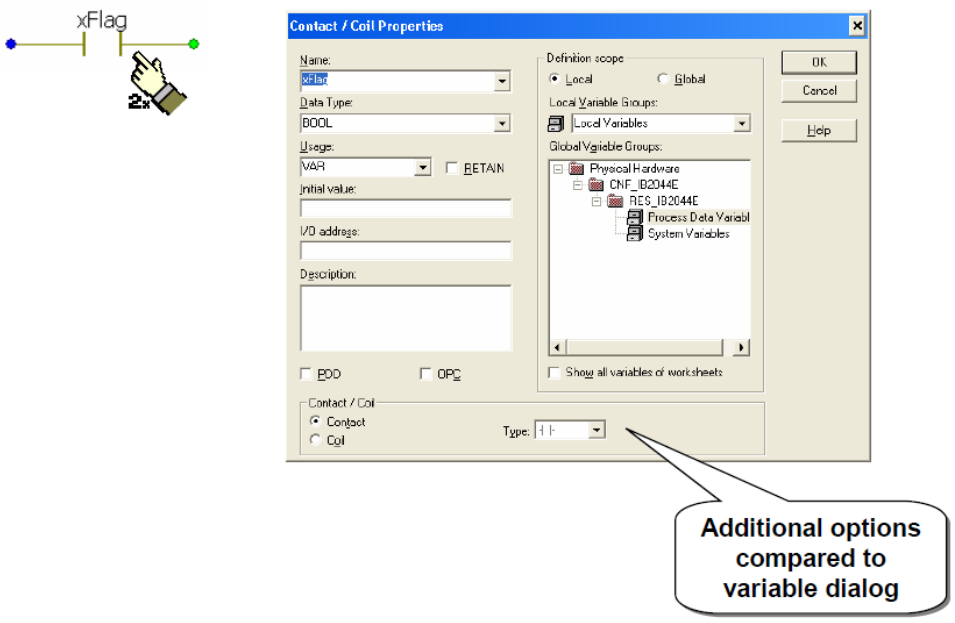


Рисунок 3.2 – Диалоговое окно контакт/реле

Поскольку используется тот же самый редактор, элементы функциональных блоковых диаграмм (переменные, функции и функциональные блоки) без проблем могут использоваться в релейно-контактных схемах (рисунок 3.3). Для получения подробных сведений о редактировании и обработке функциональных блоковых диаграмм, пожалуйста, обратитесь к разделу FBD – функциональные блоковые диаграммы.

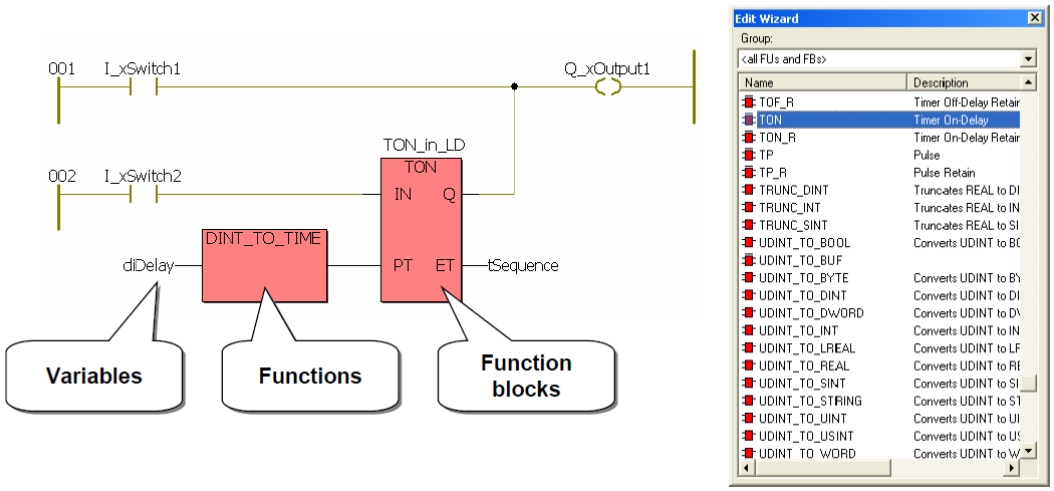


Рисунок 3.3 – Функции и функциональные блоки в релейно-контактных схемах

Обратите внимание, что чрезмерное смешивание этих двух языков в пределах одной программы может привести к результатам, которые не смогут ясно интерпретироваться компилятором, и таким образом приведет к сообщениям об ошибке.

Через пункт меню *Stretch/Compress* (растяжение/сжатие) из меню *Edit* релейно-контактная схема может быть отформатирована (рисунок 3.4). Доступны две функции:

1. Функция *Arrange Powerrails* выравнивает схемы по крайней правой.

2. Функция *Arrange Left Powerrails* выравнивает схемы по левому краю рабочей области.

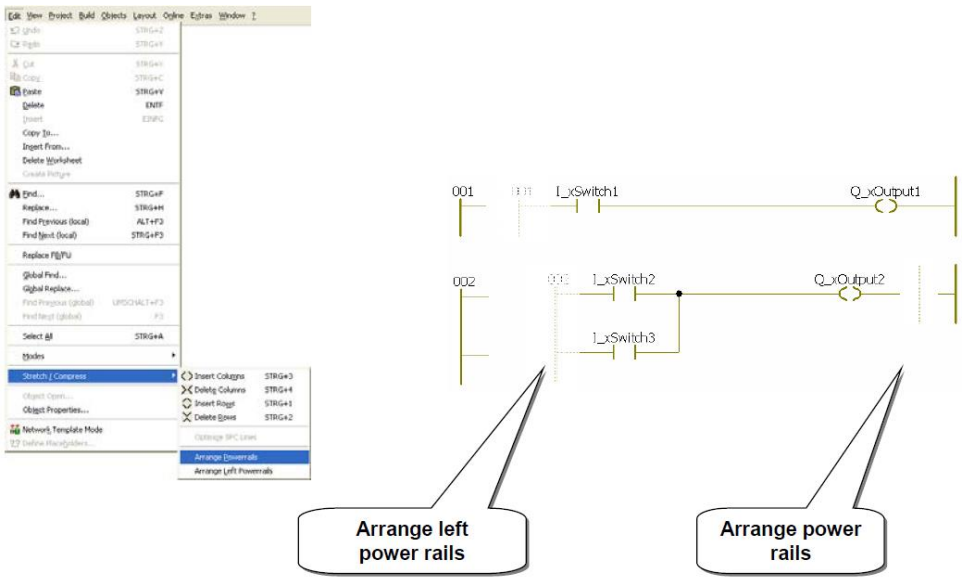


Рисунок 3.4 – Форматирование релейно-контактных схем

3.2 Последовательная функциональная диаграмма

В отличие от структуры программных блоков в других языках IEC 61131, последовательная функциональная диаграмма включает дополнительные элементы. В дополнение к текстовому редактору, таблице переменных и рабочей области, есть две папки, которые содержат существующие переходы и папки (рисунок 3.5).

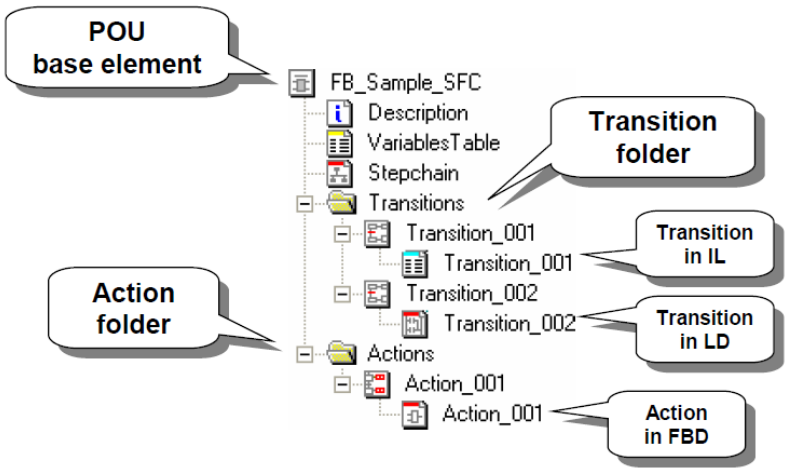


Рисунок 3.5 – Программные блоки последовательной функциональной диаграммы в структуре проекта

3.3 Базовая структура последовательной функциональной диаграммы

Базовая структура может быть добавлена в рабочую область с помощью кнопки, показанной на рисунке 3.6. Базовая структура состоит из двух основных элементов: Step (шаг) (здесь: S001) и Transition (переход) (здесь: T001) с Action block (блоком действия), соответствующим шагу (здесь: A001). Названия всех элементов могут быть изменены.

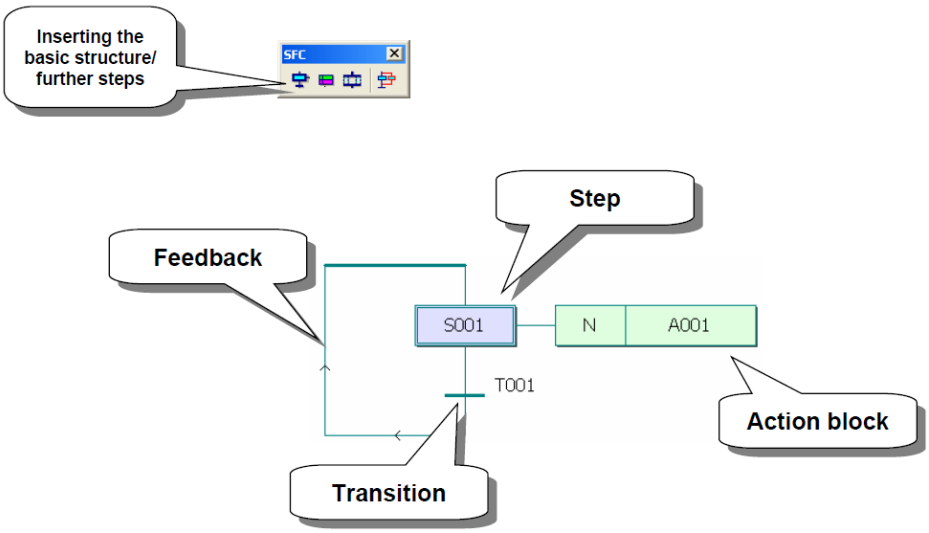


Рисунок 3.6 – Базовая структура последовательной функциональной диаграммы

Цепочка программы строится из последовательности шагов и переходов, использование блоков действий не обязательно (Шаг без действия – это шаг ожидания).

Первый шаг – часть базовой структуры, добавленная автоматически. Впоследствии система автоматически вставляет только стандартные шаги. Для каждого шага может быть добавлен комментарий через диалоговое окно шага.

Однако только флаг шага <Stepname>.x автоматически объявляется для каждого шага. Этот параметр представляет статус, т.е. действие шага, и может быть прочитан и, в некоторых случаях, написан. Это действительно в пределах программного блока.

Через диалоговое окно шага можно произвести необходимые настройки.

3.4 Базовые элементы последовательной функциональной диаграммы

На рисунке 3.7 представлены базовые элементы последовательной функциональной диаграммы.

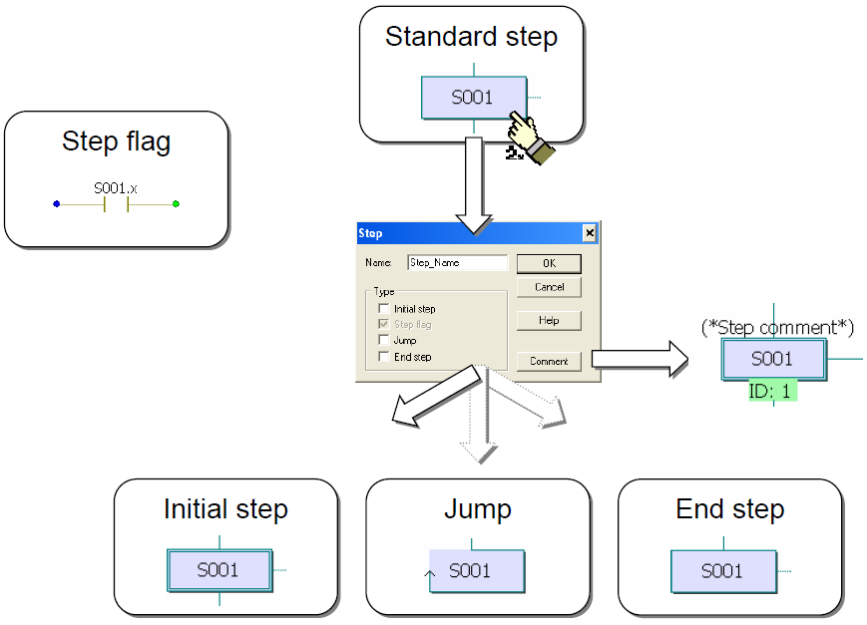


Рисунок 3.7 – Базовые элементы последовательной функциональной диаграммы

Начальный шаг

У каждой цепочки программы (в случае более одного рабочего листа) должен быть хотя бы один определенный начальный шаг. Если начало образовано параллельными ветками, то начальный шаг требуется для каждого параллельного ответвления. Блокам действия можно назначить инициализацию шага.

Скачок

В отличие от других типов шага, скачок не представляет обработку ситуации. Скачок расценивается как прямой переход в назначенный шаг, обозначенный выше имени скачка. Для скачка исключение сохраняется, он должен иметь то же самое имя, что и другой шаг.

Конечный шаг

Для последовательности программы скачок представляет тупик. При его достижении новое выполнение цепочки программы может быть достигнуто только путем управления цепочкой программы (установка/сброс флагов шага).

Цепочка программы, заканчивающаяся конечным шагом, используется для инициализации. Блокам действия можно назначить окончание шагов.

Как для скачка, так и для конечного шага прерывается связь структуры программы. Впоследствии это изменение не может быть полностью возвращено.

3.5 Типы данных, определенные пользователем (пользовательские типы данных)

Объявление типов данных, определенных пользователем, выполняется в рабочей области папки *Data Types* (Типы данных) (рисунок 3.8). Рабочая область *sys_flag_types* включена в каждый шаблон проекта в PC WORX и необходим для системных типов данных системы управления. Рабочая область может быть переименована (*SYSTEM*), но нельзя исправлять объявленные в ней типы данных.

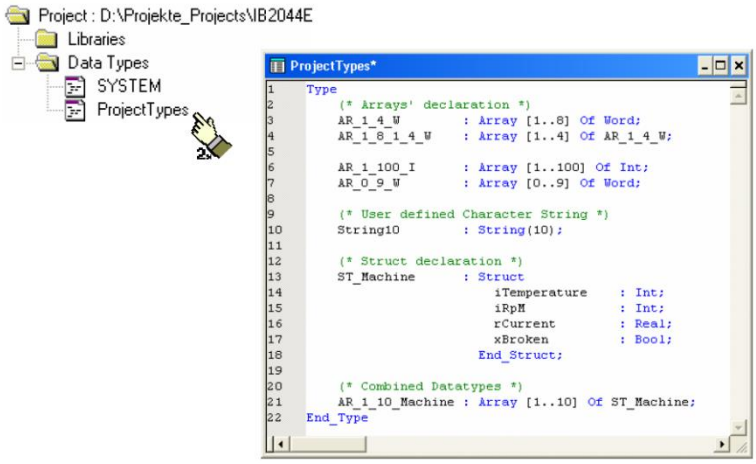


Рисунок 3.8 – Рабочая область для пользовательских типов данных

В основном все объявления переменных для одного проекта могут быть сделаны в одной рабочей области. Однако для упорядочения и правильной организации проекта рекомендуется создавать отдельные рабочие листы для каждой функции.

3.6 Области данных

Объявление типа данных класса массивов суммируют элементы в одном исходном типе данных. Пример на рисунке 3.9 показывает соединение четырех переменных типа данных Word (слово) в одной переменной массива, основанной на новом типе данных.

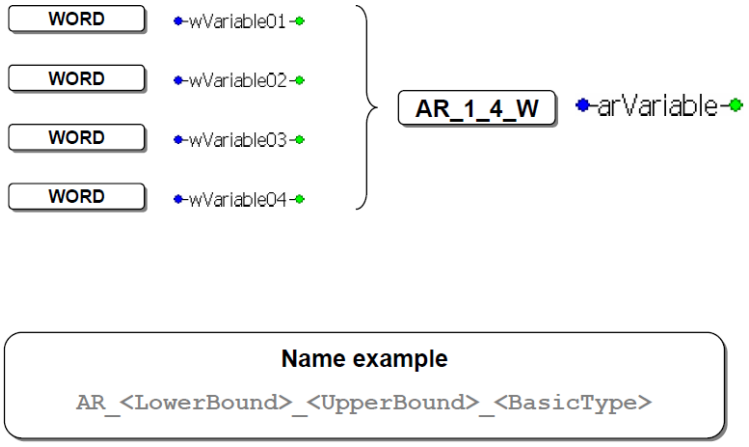


Рисунок 3.9 – Принципы организации областей

Название нового типа данных в области данных может быть выбрано любое. Однако, как и в любых других случаях программирования,

рекомендуется использовать соответствующее название для типов данных, определенных пользователем.

3.7 Массивы

Для объявления массивов используется определенный формат, шаблон которого можно вызвать с помощью редактора объявления типа данных. Необходимо ввести желаемое имя типа данных, нижнюю и верхнюю границы множества в квадратных скобках (как положительное целочисленное значение) и тип исходных данных.

Объявление многомерных массивов обычно возможно с помощью процедуры, описанной на рисунке 3.10.

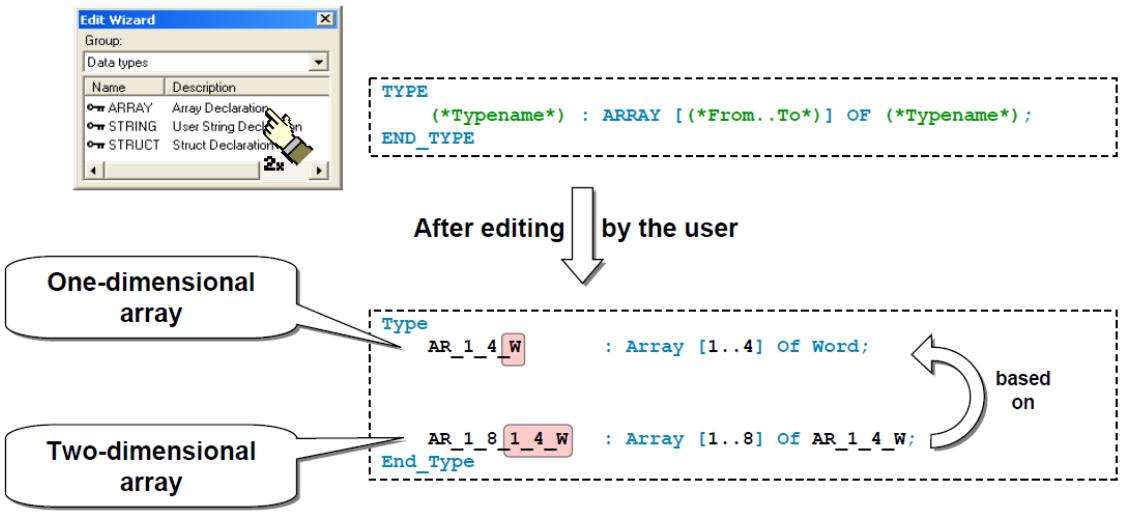


Рисунок 3.10 – Объявление массивов

После закрытия или компилирования рабочего листа типа данных (*Alt+F9*), недавно объявленные типы данных доступны в списке выбора типов данных. Две переменные того же самого типа данных, справедливо также для массивов, могут быть присвоены друг другу. Вызов отдельных элементов может быть сделан или после объявления переменной массива в программном блоке и использовании целой константы, или использовании целой переменной, которая служит в качестве индекса для массива.

Если переменные используются для того, чтобы получить доступ к отдельным элементам, необходимо обеспечить посредством программирования чтобы не были нарушены границы массива (рисунок 3.11).

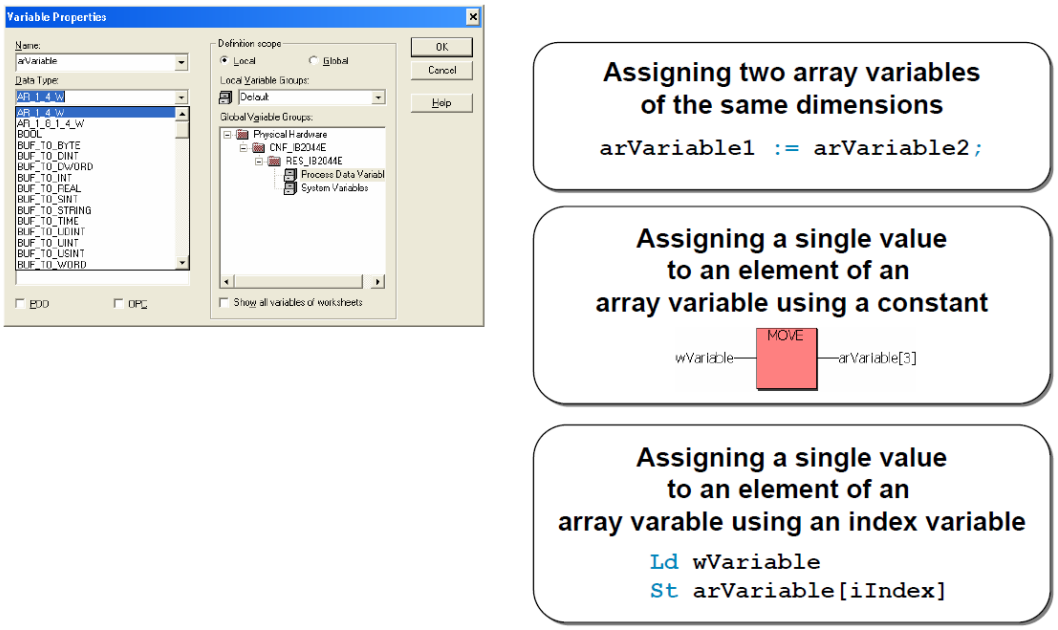


Рисунок 3.11 – Использование массивов при программировании

3.8 Структуры данных

Типы данных класса структуры (Structures) могут содержать элементы одного или нескольких исходных типов данных. На рисунке 3.12 показано соединение переменных четырех различных типов данных в одной переменной структуры, основанной на новом типе данных.

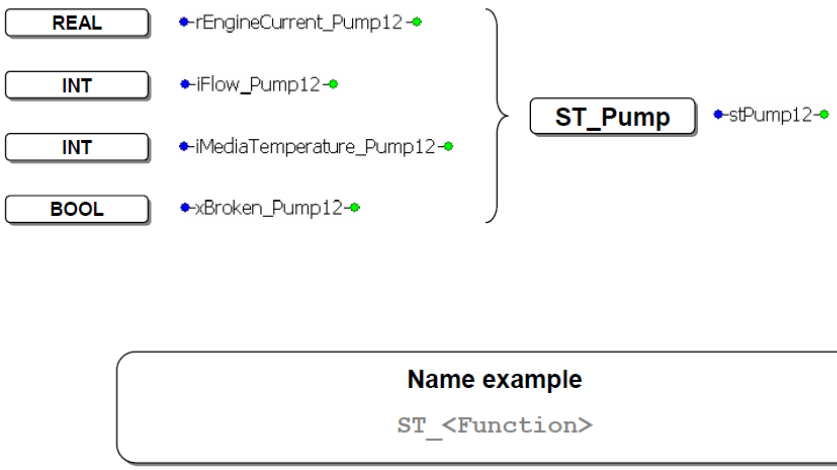


Рисунок 3.12 – Принципы организации структур

Название нового типа данных класса структуры может быть выбрано любое. Однако, как и в любых других случаях программирования,

рекомендуется использовать соответствующее название для типов данных, определенных пользователем.

Объявление структур возможно с помощью процедуры, изложенной на рисунке 3.13.

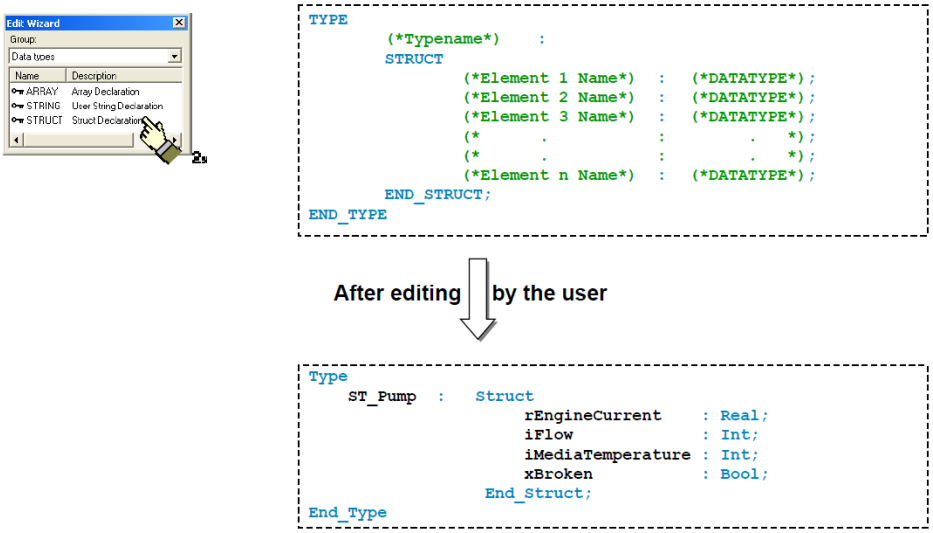


Рисунок 3.13 – Объявление структур

3.9 Использование структур для программирования

Для объявления структур используется определенный формат, шаблон которого можно вызвать с помощью редактора объявления типа данных. Необходимо ввести желаемое имя типа данных и элементы структуры. Возможно также объявление многомерных структур (рисунок 3.14).

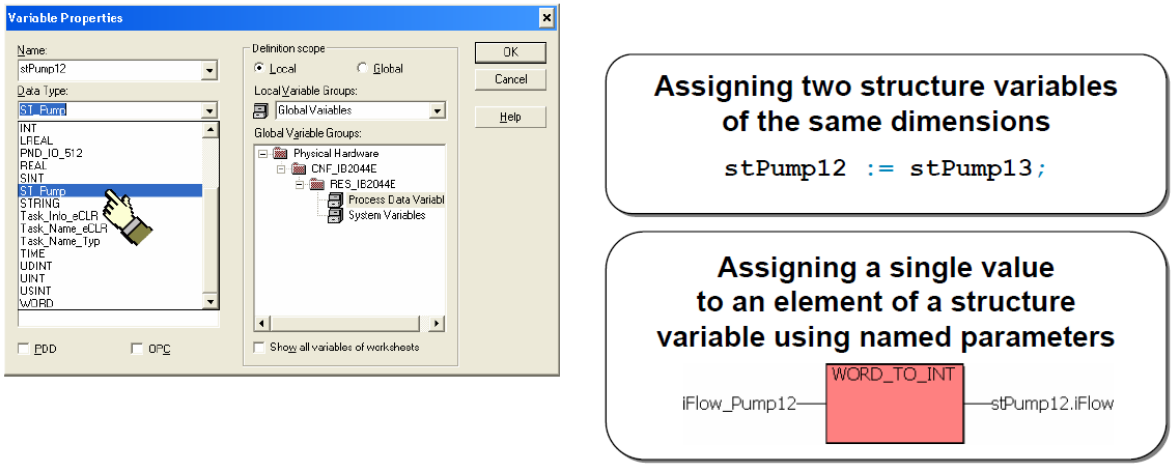


Рисунок 3.14 – Использование структур при программировании

После закрытия или компилирования рабочего листа типа данных *(Alt+F9)* недавно объявленные типы данных доступны в списке выбора типов данных. Две переменные того же самого типа данных, справедливо также для структур, могут быть присвоены друг другу. Вызов отдельных элементов может быть сделан через названный параметр после объявления переменной типа структуры в программном блоке.

Во многих проектах объединяются объявления типов данных массивов и структур. На рисунке 3.15 показано объявление типа данных, который включает переменные данные насоса. Массив содержит 20 элементов структуры насоса, определенных как 1 – 20.

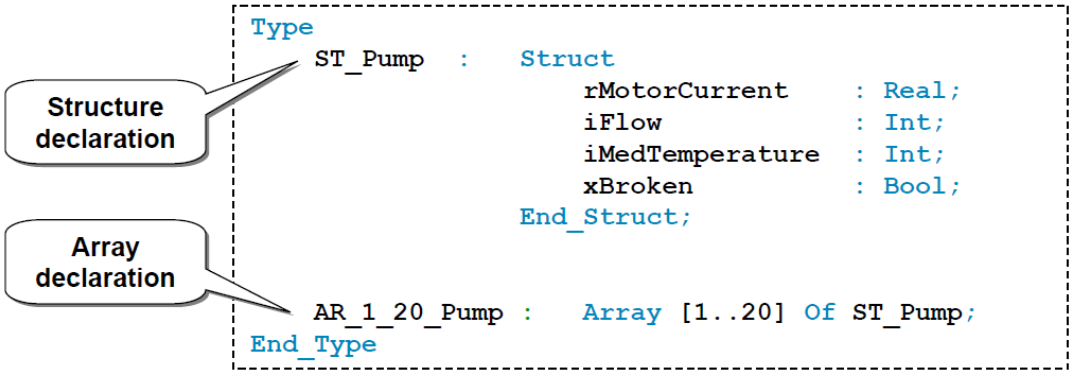


Рисунок 3.15 – Объявление смешанных пользовательских типов данных

3.10 Использование смешанных пользовательских типов данных при программировании

Доступ к элементу переменной, относящейся к массиву данных насоса, придерживается правила «от большего к меньшему». Это означает, что вначале необходимо определить, к какому насосу из массива должен быть получен доступ (целочисленная переменная или целочисленная константа в квадратных скобках), и затем должен быть определен параметр указанного насоса (через названный параметр). На рисунке 3.16 показан алгоритм использования смешанных пользовательских типов данных при программировании.

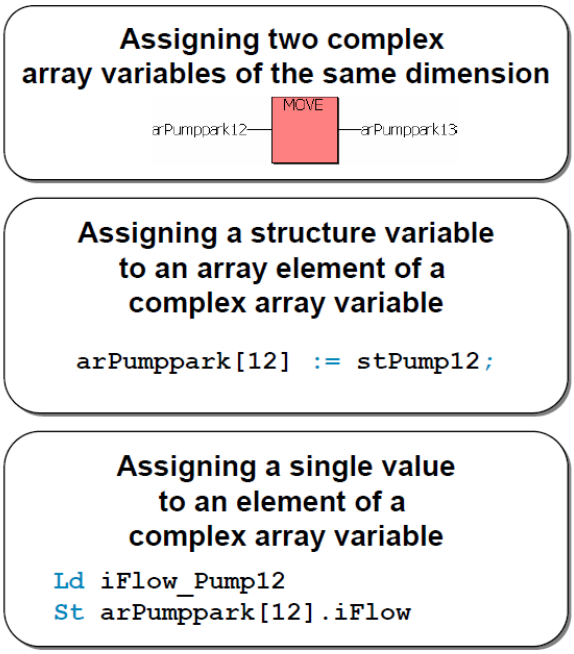


Рисунок 3.16 – Использование смешанных пользовательских типов данных при программировании

Контрольные вопросы

1. Базовые элементы релейно-контактной схемы?
2. Можно ли использовать функции и функциональные блоки в релейно-контактных схемах?
3. Что такое последовательная функциональная диаграмма?
4. Базовые элементы последовательной функциональной диаграммы?
5. Основные элементы базовой структуры последовательной функциональной диаграммы?
6. Могут ли типы данных класса структуры (Structures) содержать элементы нескольких исходных типов данных?
7. Как происходит объявление массивов?
8. Как происходит использование структур при программировании?
9. Как происходит объявление смешанных пользовательских типов данных?
10. Что означает правило «от большего к меньшему»?

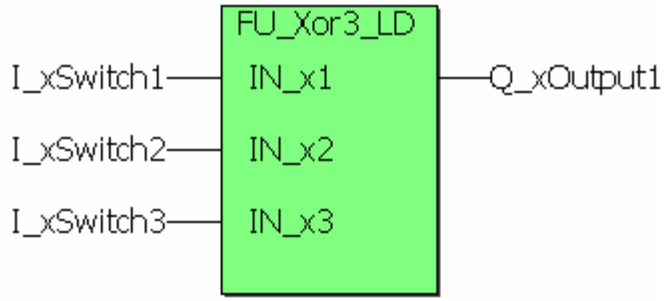
Задания

Задание 1. Добавить в проект функциональный блок FB_TFlipflop_LD, при программировании использовать язык LD.

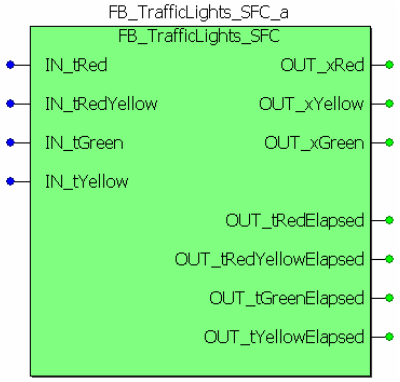
Добавьте функцию с именем *FU_TFlipflop_LD* в дерево проекта. Эта функция должна работать подобно *FU_TFlipflop_FBD* (см. *задание 3* к лабораторной работе №2).

Задание 2. Добавить в проект функцию FU_Xor3_LD.

Добавьте функцию с именем *FB_Xor3_LD* в дерево проекта. Эта функция должна работать подобно *FB_ Xor3_FBD* (см. *задание 2* к лабораторной работе №2).



Задание 3. Добавить в проект функциональный блок FB_TrafficLight_SFC, при программировании использовать язык SFC.



Основные функции

Добавьте функциональный блок с именем *FB_TrafficLight_SFC* в проект дерева.

Основные функции одного устройства сигнала должны быть реализованы на первом этапе, т. е. последовательность *Красный* | *красно-желтый* | *зеленый* | *желтый*.

Для блока с помощью входных параметров (тип данных *Time*) должно быть доступно управление длинами фаз переключения.

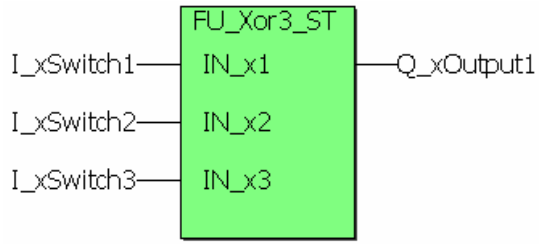
Отдельные цвета элементов должны изменяться в зависимости от выходных параметров.

Последовательность соответствующих фаз используется как переход состояний.

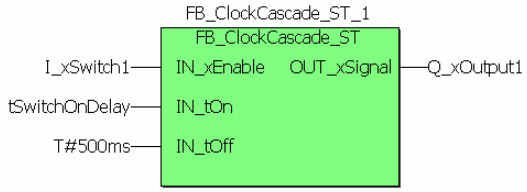
Задание 4. Добавить в проект функцию FU_Xor3_ST, при программировании использовать язык ST.

Добавить функцию с именем *FU_Xor3_ST* в дерево проекта. Эта функция выполняет ту же задачу что и соответствующая функция на *FBD* (см. *задание 2* к лабораторной работе №2).

.



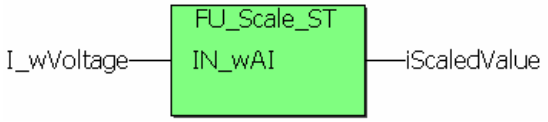
Задание 5. Добавить в проект функциональный блок FB_ClockCascade_ST.



Добавьте функциональный блок с именем *FB_ClockCascade_ST* в дерево проекта. Эта функциональный блок должен выполнять ту же задачу, что и аналогичный функциональный блок задачи *FBD* (см. *задание 4* к лабораторной работе №2).

Задание 6. Добавить в проект функции для обработки аналоговых значений процесса, при программировании использовать язык ST.

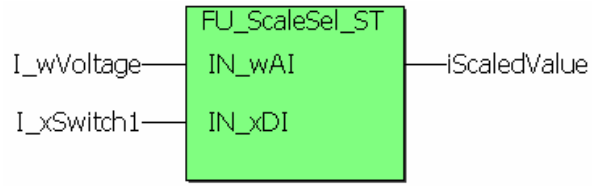
[a] Основные функции: FU_Scale_ST



Добавьте функцию с именем *FU_Scale_ST* в дерево проекта. Эта функция выполняет ту же задачу, что и функция задачи *FBD* (см. *задание 5a* к лабораторной работе №2).

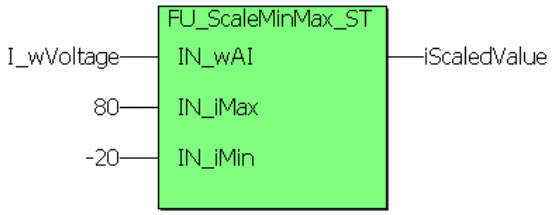
Совет: для функций, которые должны выполняться параллельно и последовательно, вы должны использовать скобки или локальные переменные в структурированном тексте.

[b] Расширение двоичным выбором диапазона: FU_ScaleSel_ST



Добавьте функцию с именем *FU_ScaleSel_ST* в дерево проекта. Эта функция выполняет ту же задачу, что и функция задачи *FBD* (см. *задание 5b* к лабораторной работе №2).

[c] Расширение для определяемого пользователем масштабирования: FU_ScaleMinMax_ST



Добавьте функцию с именем *FU_ScaleMinMax_ST* в дерево проекта. Эта функция выполняет ту же задачу, что и функция задачи *FBD* (см. *задание 5c* к лабораторной работе №2).

Лабораторная работа №4. Составление релейно-контактных схем управляющих программ

Цель: изучение принципов составления программы управления промышленным оборудованием (конвейеры, толкатели, манипуляторы и т.п.).

4.1 Переходы

Для этого вида программирования на интерфейсе шага переход переключен на прямое соединение (*Direct connection*) через диалоговое окно свойств (рисунок 4.1). Переход представляет собой точку соединения с функциональными блоками и функциями. Для выполнения требований переключения необходимо назначить точке переключения булевый параметр. Для этого могут использоваться релейно-контактные схемы, функции и функциональные блоки (см. Лабораторные работы №2 и №3).

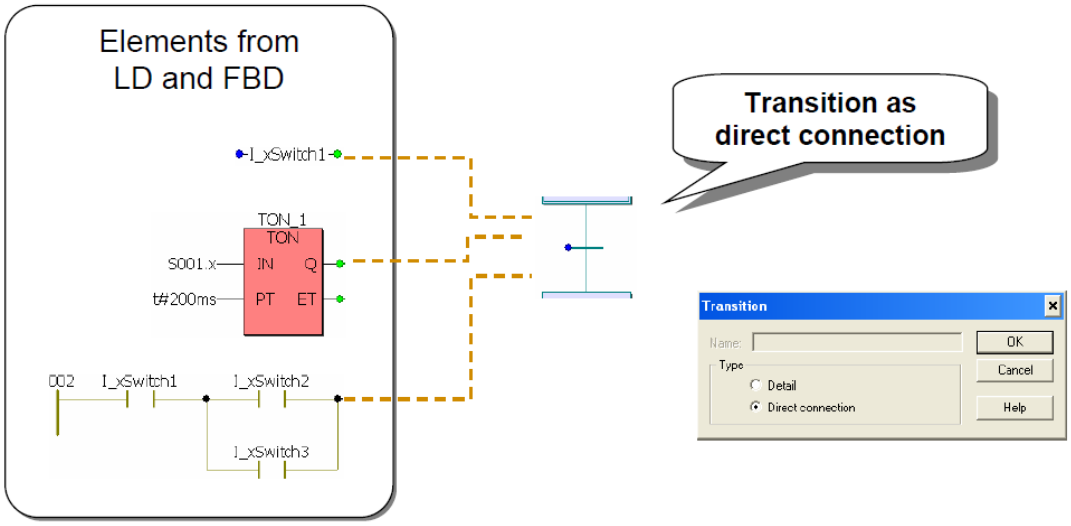


Рисунок 4.1 – Прямое соединение переходов

В пределах цепи программы переходы выполняют условия перехода из одной ситуации процесса в следующую (рисунок 4.2). Есть два способа программировать эти условия.

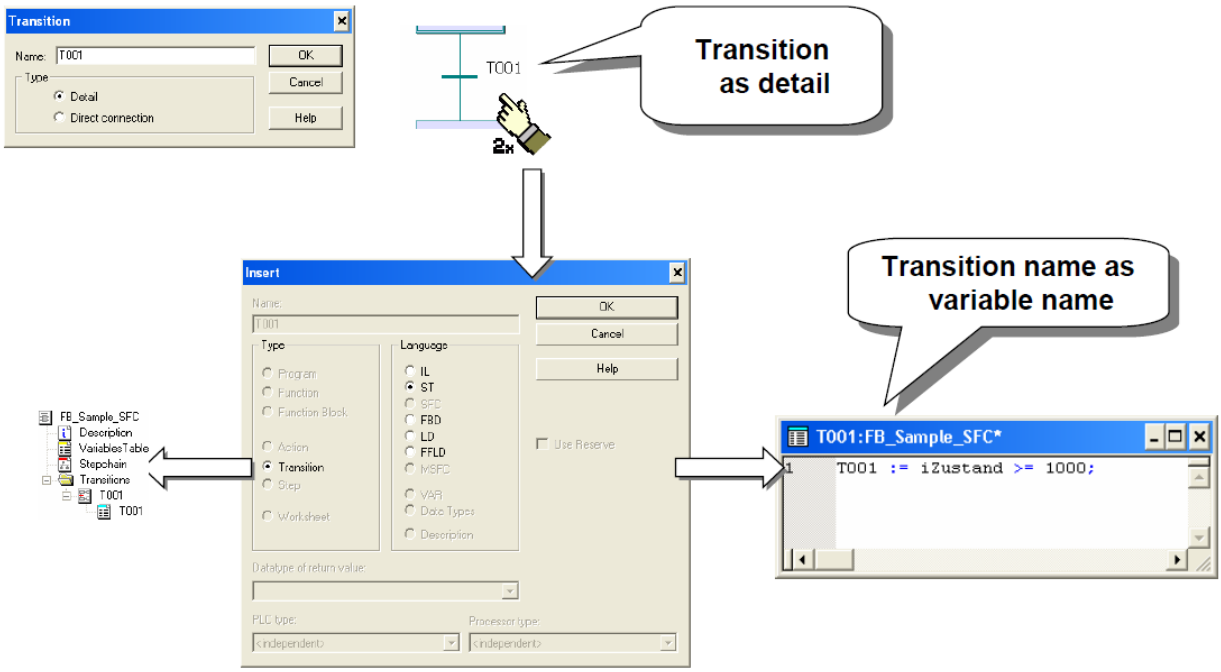


Рисунок 4.2 – Переходы

Элемент перехода представляет один или больше рабочих областей, которые могут быть запрограммированы с помощью одного из языков стандарта IEC 61131, за исключением SFC (последовательной функциональной диаграммы). После изменения имени перехода вы можете открыть диалог вставки для нового элемента двойным щелчком на имени перехода. После изменения языка, переход помещается в папку перехода POU (программных блоков). В это время первый рабочий лист открыт. Программирование в переходе должно создать требование переключения, которое – подобно возвращаемому значению для функций – должно быть сохранено в переменной с названием перехода. В отличие от имен функции, этот параметр не сохраняется автоматически в диалоговом окне переменной, а должен быть введен вручную. Однако, система идентифицирует правильно введенное имя и затем блокирует элементы управления, которые обычно доступны для объявления переменной.

Для последующих изменений элемент перехода может быть открыт из программы или через соответствующий элемент в дереве проекта.

4.2 Блоки действий

Блоки действий (рисунок 4.3) влияют на ситуацию процесса так же, как шаги представляют ситуации процесса. Они могут быть выполнены как элемент действия (стандартный цвет: светло-зеленый). Это означает, что они могут использоваться в качестве программы в отдельных рабочих листах или для того, чтобы управлять булевыми параметрами (стандартный цвет: бледно-розовый). Вариант использования может быть установлен в диалоговом окне свойств блока действия.

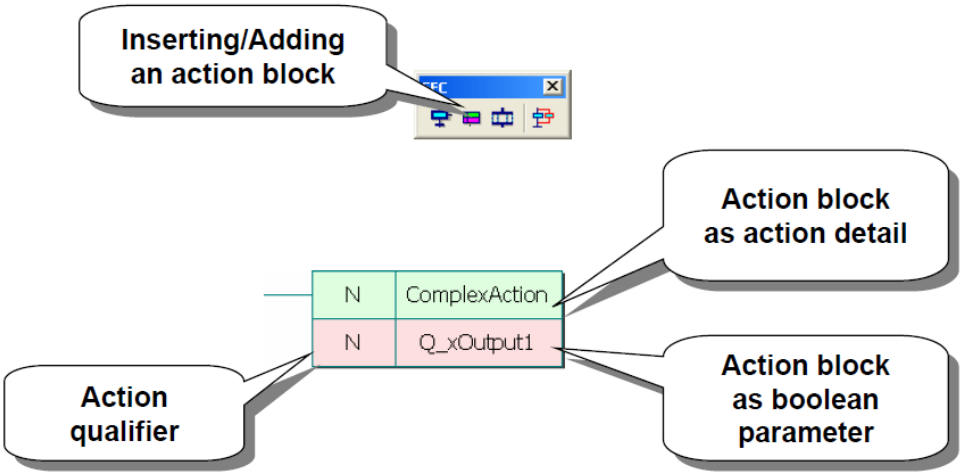


Рисунок 4.3 – Блоки действий

Кроме того, статус шага (определитель действия) важен для выполнения действия. Помимо символа N (N = несохраненный, действие активное, пока шаг является активным) все остальные символы согласно IEC 61131-3 также могут быть использованы:

- *S – set.* Это действие продолжается, пока оно явно не будет сброшено символом R (даже если соответствующий шаг становится неактивным);

- *R – reset.* Заканчивает выполнение действия, которое было предварительно начато с помощью символа S;

- *L – time limited.* После активации соответствующего шага действие выполняется в течение установленного времени или до тех пор, пока шаг становится неактивным;

- *D – time delayed.* Задержка начинается, когда шаг становится активным. После того, как установленное время прошло, действие выполняется до тех пор, пока шаг не будет дезактивирован. Если шаг

становится неактивным прежде, чем закончилось установленное время, действие не будет выполнено;

- *P – pulse.* Выполнение начинается, когда шаг становится активным/неактивным, и выполняется однократно;

- *SD – stored + time delayed.* Действие выполняется, пока не сброшено символом R, но начнется только после того, как временная задержка прошла. Даже если шаг является активным в течение меньшего времени, чем временная задержка;

- *DS – time delayed + stored.* Действие выполняется, пока не сброшено символом R, но начнется только после того, как временная задержка прошла. В отличие от предыдущего символа, действие не произойдет, если шаг становится неактивным прежде, чем пройдет временная задержка;

- *SL – stored + time limited.* Действие выполняется в течение установленного времени, даже если шаг является активным в течение более короткого времени. Действие может быть сброшено символом R.

4.3 Переменные действия

Как только действие добавлено к шагу как переменная, или действие установлено в переменную в меню свойств блока действия, переменная должна быть объявлена или вызвана после известной процедуры (рисунок 4.4).

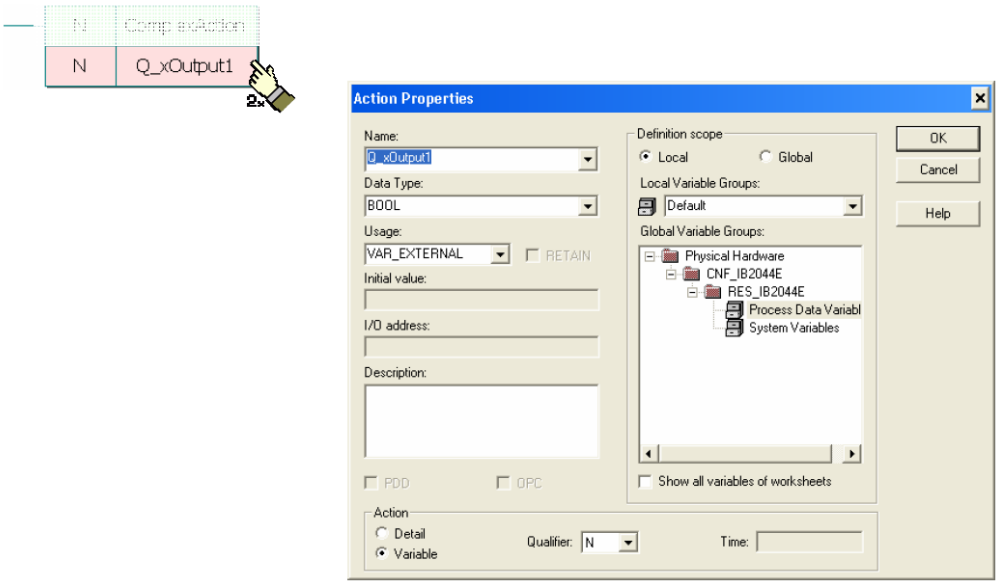


Рисунок 4.4 – Переменные действия

При этом могут использоваться логические переменные или – как в релейно-контактной схеме – булевые параметры для получения неявного доступа к отдельным битам переменных слова, байта или двойного слова, или булевым параметрам структуры и переменных поля.

Если блок действия используется в качестве элемента действия (рисунок 4.5), имя, которое будет использоваться для сложного действия, должно быть отредактировано через контекстное меню в диалоговом окне. В этом контексте сложное действие относится ко всему, что превышает управление отдельной логической переменной.

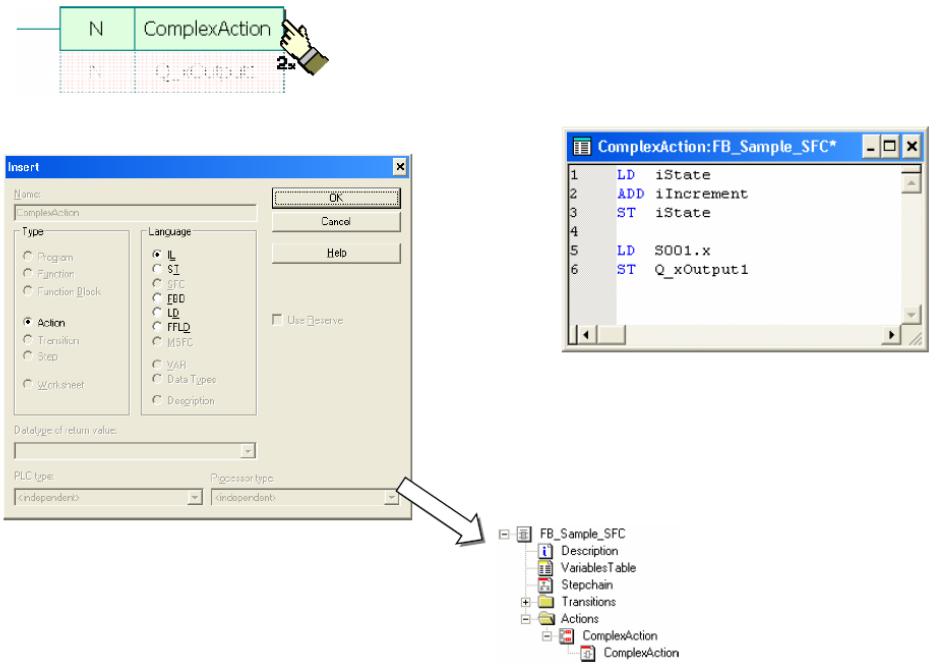


Рисунок 4.5 – Элемент действия

Как только имя назначено, элемент может быть создан двойным щелчком. В открывшемся диалоговом окне должен быть выбран только язык программирования. За исключением последовательной функциональной диаграммы, доступны все языки IEC 61131. При открытии первого рабочего листа элемента действия создается соответствующий элемент в папке *Actions POU*. Теперь рабочие листы могут быть выбраны в дереве проекта или непосредственно из программы при использовании блока действия.

4.4 Ветвление последовательной функциональной диаграммы

Последовательные функциональные диаграммы позволяют создавать параллельные и альтернативные ответвления (рисунок 4.6).

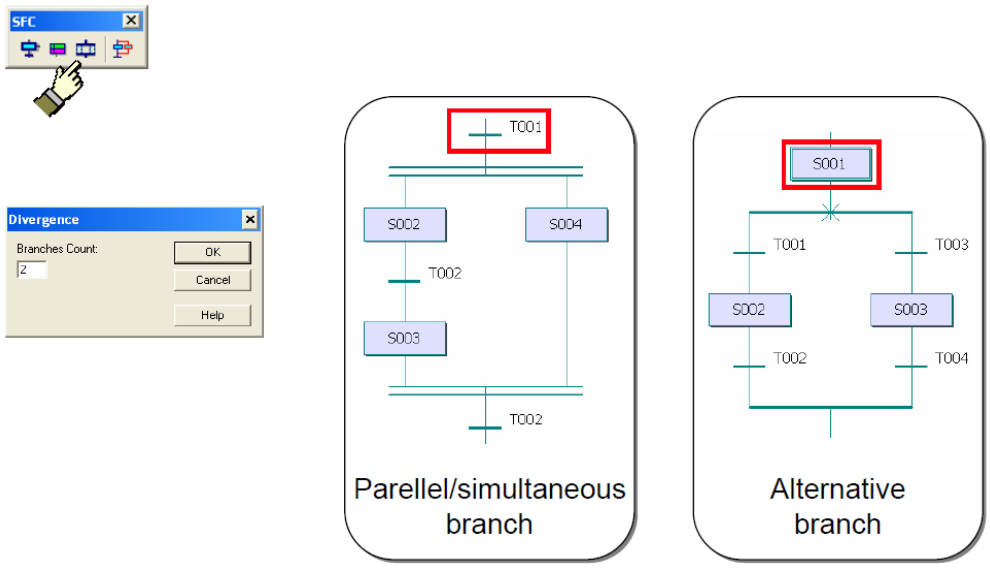


Рисунок 4.6 – Ветвление последовательной функциональной диаграммы

Параллельная ветка

Для создания параллельной ветки необходимо выбрать шаг, после которого должно начаться ответвление. Число ветвей, которые могут быть добавлены, зависит от размера рабочей области. Ветви выполняются независимо друг от друга. Выполнение будет сгруппировано, если конечные шаги активны во всех ветках и отвечают требованию переключения сгруппированного перехода.

Объединение скачков и конечных шагов в параллельных ответвлениях будет определено PC WORX как недопустимое программирование.

Альтернативная ветка

Для создания альтернативной ветки необходимо выбрать шаг, после которого должно начаться ответвление. Число ветвей, которые могут быть добавлены, зависит от размера рабочей области. В первую очередь будет выполняться ветка, в которой начальное условие перехода отвечает требованию. Если одновременно соблюдается больше чем одно условие, выполнение будет производиться согласно порядку выполнения в

графической рабочей области, т. е. в установленном приоритете слева направо.

В альтернативных ответвлениях возможно объединить скачки и конечные шаги.

После выбора кнопки, показанной на рисунке 4.7, указатель мыши изменяется на символ (1).

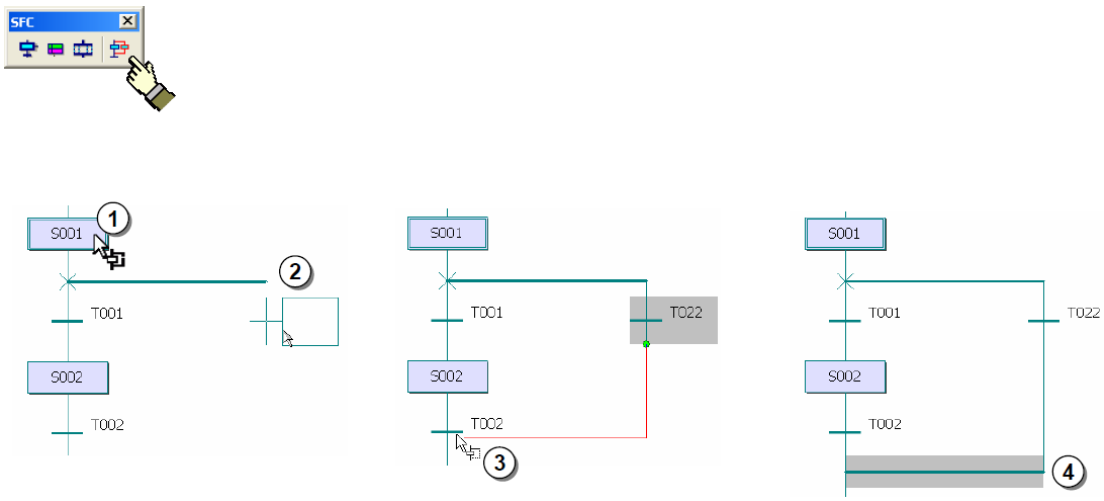


Рисунок 4.7 – Добавление ветвлений

Для создания параллельной ветки Вам нужно выбрать шаг 1, после которого должно начаться ответвление, затем 2, следующим щелчком мыши выберите горизонтальное положение нового ответвления. Последний клик необходимо сделать на шаге, перед которым должно закончиться ответвление, или на переходе 3, позади которого должно закончиться ответвление, процедура 4 заканчивается.

Для создания альтернативной ветки необходимо провести ту же самую процедуру, что и для параллельной. Однако в качестве первого элемента необходимо выбрать переход, за которым должно начинаться ответвление. Ответвление закрывается после выбранного шага или перед выбранным переходом.

4.5 Описание лабораторного макета

Внешний вид лабораторного макета конвейерной линии, которая состоит из 4 отдельных макетов-конвейеров, представлен на рисунке 4.8. Все датчики и управляющие элементы пронумерованы (подробнее см. задания ниже).

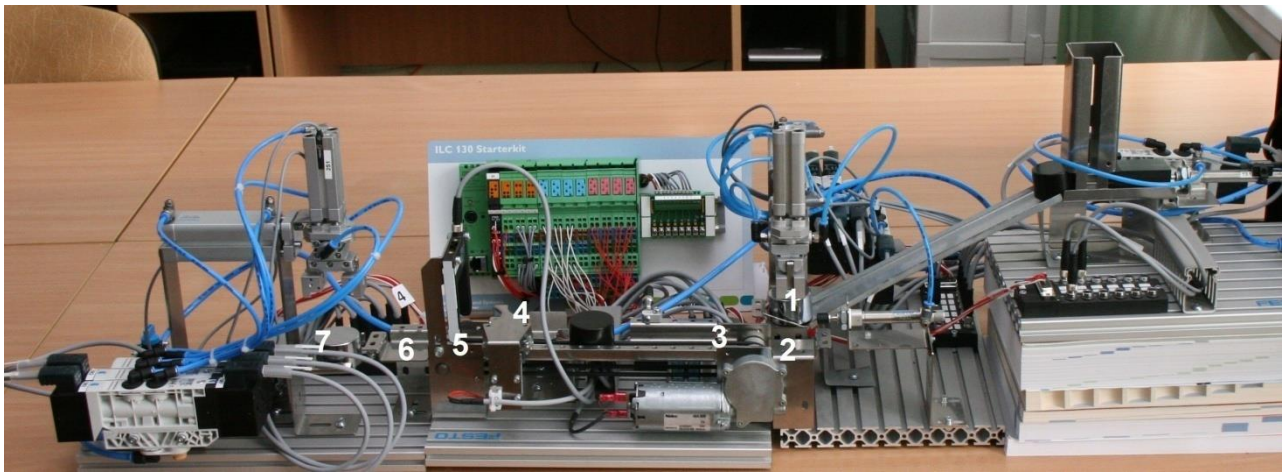


Рисунок 4.8 – Внешний вид лабораторных макетов

Контрольные вопросы

1. Для чего необходимы переходы? Приведите пример.
2. Как объявить переменную действия?
3. Назовите основные блоки действий.
4. Как происходит ветвление последовательной функциональной диаграммы?
5. Что такое параллельная ветка?
6. Что такое альтернативная ветка?
7. Отличия параллельной и альтернативной веток?

Задания

Задание 1. Написать управляющую программу для первого макета (рисунок 4.9).

При поступлении фронта сигнала входного бита 0 первый макет должен начать работу. При этом если индуктивный датчик 1S1 будет находиться в состоянии TRUE, толкатель фишек должен вытолкнуть фишку из приемника – перейти в положение 2 (см. рисунок 4.9). Если индуктивный датчик находится в состоянии FALSE и фишка вытолкнута, то толкатель должен вернуться в исходное положение 1 (см. рисунок 4.9). Для остановки работы толкателя необходимо бит 0 перевести в состояние FALSE. Алгоритм работы макета представлен в таблицах 4.1 и 4.2 (входы и выходы контроллера).

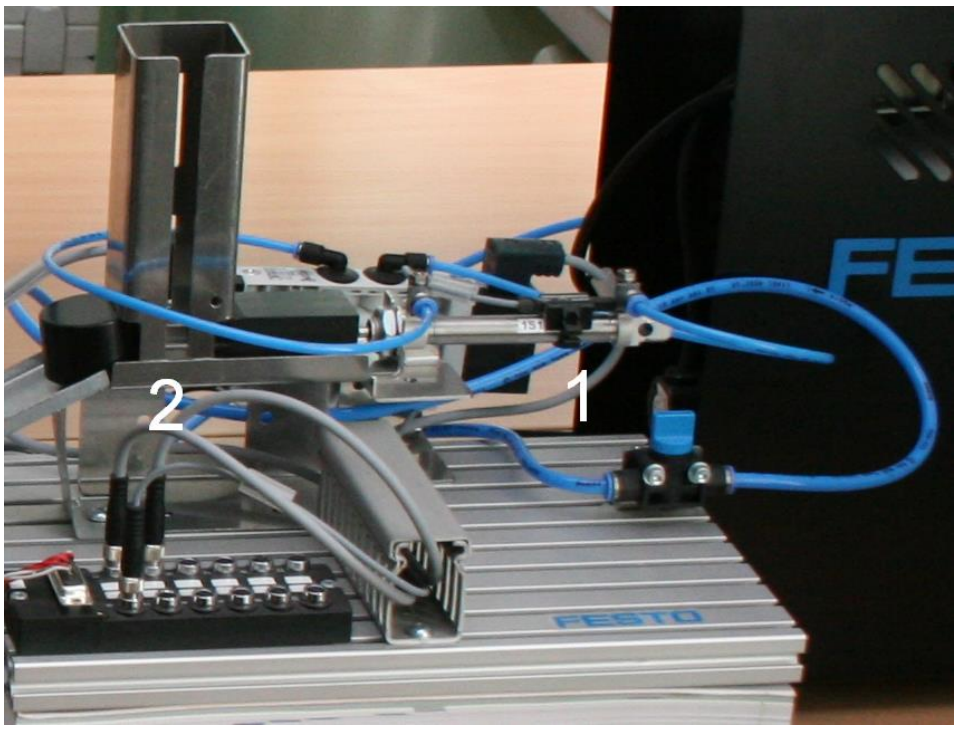


Рисунок 4.9 – Внешний вид первого макета

Таблица 4.1 – Первый контроллер (входы)

Входы I	Описание
I_0_1_1_1_1	Толкатель в исходном положении

Таблица 4.2 – Первый контроллер (выходы)

Выходы Q	Описание
Q_0_2_1_1	Вернуться в исходное состояние
Q_0_2_2_1	Выдвинуть цилиндр (рабочий ход)

Задание 2. Написать управляющую программу для второго макета (рисунок 4.10).

При поступлении фронта сигнала входного бита 0 второй макет должен начать работу. Основная задача заключается в том, чтобы переместить фишку из позиции 1 в позицию 2 (см. рисунок 4.10). Затем необходимо столкнуть фишку с позиции 2, не изменив нормальный режим работы манипулятора. Для остановки толкателя необходимо бит 0 перевести в состояние FALSE. Алгоритм работы контроллера для второго задания представлен в таблицах 4.3 и 4.4 (входы и выходы контроллера).

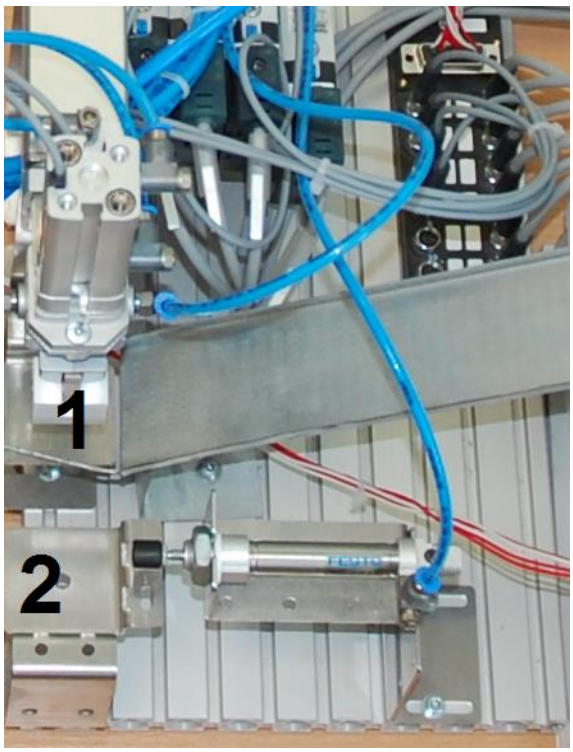


Рисунок 4.10 – Внешний вид второго макета

Таблица 4.3 – Второй контроллер (входы)

I	Описание
I_0_1_1_1_1	Манипулятор выдвинут (координата по горизонтали)
I_0_1_1_1_4	Захват опущен (координата по вертикали)
I_0_1_1_2_1	Манипулятор в исходном положении (координата по горизонтали)
I_0_1_1_2_4	Захват в исходном состоянии (координата по вертикали)

Таблица 4.4 – Второй контроллер (выходы)

Q	Описание
1	2
Q_0_2_1_1	Вернуть захват в исходное положение (координата по вертикали)
Q_0_2_1_4	Манипулятор выдвинуть (координата по горизонтали)
Q_0_2_2_1	Захватить фишку

1	2
Q_0_2_2_4	Вернуть манипулятор в исходное положение (координата по горизонтали)
Onboard_Output_Bit_0	Опустить захват (координата по вертикали)
Onboard_Output_Bit_1	Выдвинуть цилиндр (Столкнуть фишку)

Задание 3. Написать управляющую программу для третьего макета (рисунок 4.11).

При поступлении фронта сигнала оптического датчика третий макет (см. рисунок 4.11) должен начать работу. Данный макет предназначен для сортировки металлических и неметаллических фишек.

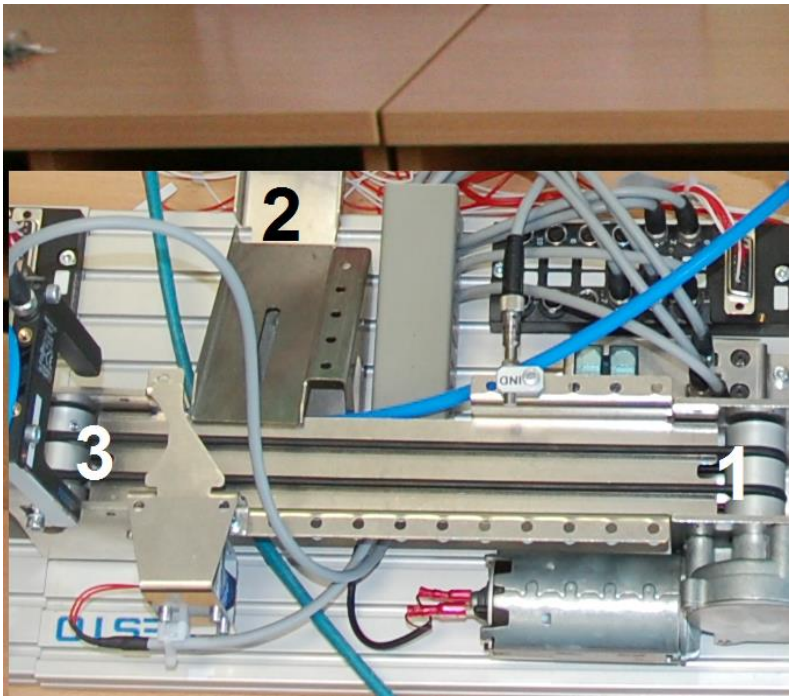


Рисунок 4.11 – Внешний вид третьего макета

Основная задача заключается в том, чтобы подать фишки на позицию 1 (см. рисунок 4.11) – оптический датчик. Двигатель при этом должен запуститься автоматически. Если фишка металлическая, то сработает индуктивный датчик и соленоид-сортировщик должен не дать фишке пройти на позицию 3, фишка скатывается на позицию 2. Если фишка неметаллическая, то индуктивный датчик не сработает и соленоид пропускает неметаллическую фишку на позицию 3. Двигатель должен автоматически остановиться при отсутствии фишек на конвейере.

Алгоритм работы контроллера для третьего задания представлен в таблицах 4.4 и 4.5 (входы и выходы контроллера).

Таблица 4.5 – Третий контроллер (входы)

I	Описание
I_0_1_1_1_1	Оптический датчик «рамка» (наличие любой детали на конвейере)
I_0_1_1_2_1	Индуктивный датчик (наличие металлической детали на конвейере)

Таблица 4.6 – Третий контроллер (выходы)

Q	Описание
Q_0_2_1_1	Соленоид-сортировщик
Q_0_2_1_4	Включить конвейер

Задание 4. Написать управляющую программу для четвертого макета (рисунок 4.12).

При поступлении фронта сигнала входного бита 0 четвертый макет должен начать работу. Основная задача заключается в том, чтобы переместить фишку из позиции 1 в позицию 2 (см. рисунок 4.12). Алгоритм работы контроллера для четвертого задания представлен в таблицах 4.7 и 4.8 (входы и выходы контроллера).

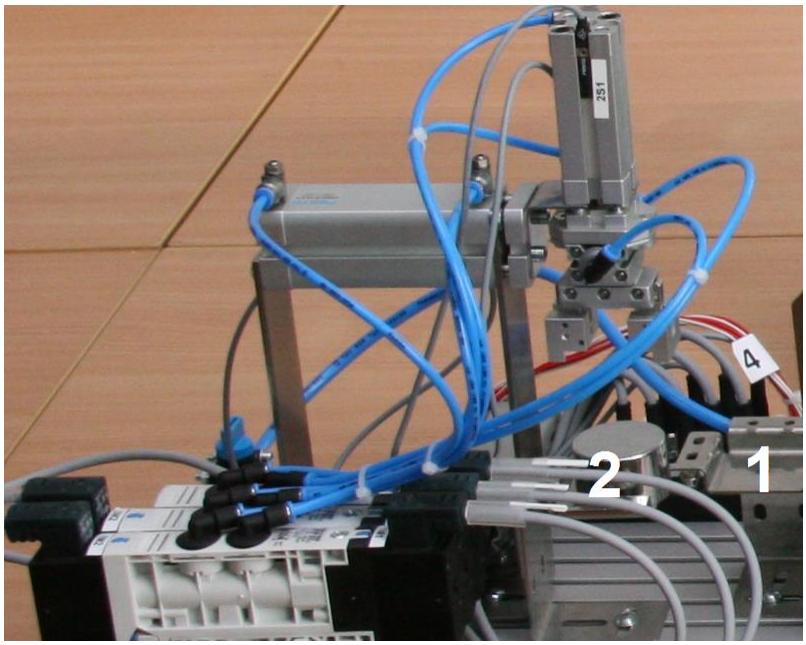


Рисунок 4.12 – Внешний вид четвертого макета

Таблица 4.7 – Четвертый контроллер (входы)

I	Описание
I_0_1_1_1_1	Манипулятор выдвинут (координата по горизонтали)
I_0_1_1_1_4	Манипулятор в исходном положении (координата по горизонтали)
I_0_1_1_2_1	Захват опущен (координата по вертикали)
I_0_1_1_2_4	Захват в исходном состоянии (координата по вертикали)

Таблица 4.8 – Четвертый контроллер (выходы)

Q	Описание
Q_0_2_1_1	Захватить фишку
Q_0_2_1_4	Манипулятор выдвинуть (координата по горизонтали)
Q_0_2_2_1	Вернуть захват в исходное положение (координата по вертикали)
Q_0_2_2_4	Вернуть манипулятор в исходное положение (координата по горизонтали)
Onboard_Output_Bit_3	Опустить захват (координата по вертикали)

Лабораторная работа №5. Микропроцессорная система управления технологическим процессом

Цель: изучение принципов построения микропроцессорной системы управления технологическим процессом.

5.1 Язык программирования ST (структурированного текста)

Доступ к языковым элементам структурированного текста главным образом обеспечивается через редактор *Edit Wizard* (рисунок 5.1). В дополнение к функциям и функциональным блокам, в структурированном тексте доступна группа *Keywords* (Ключевые слова).

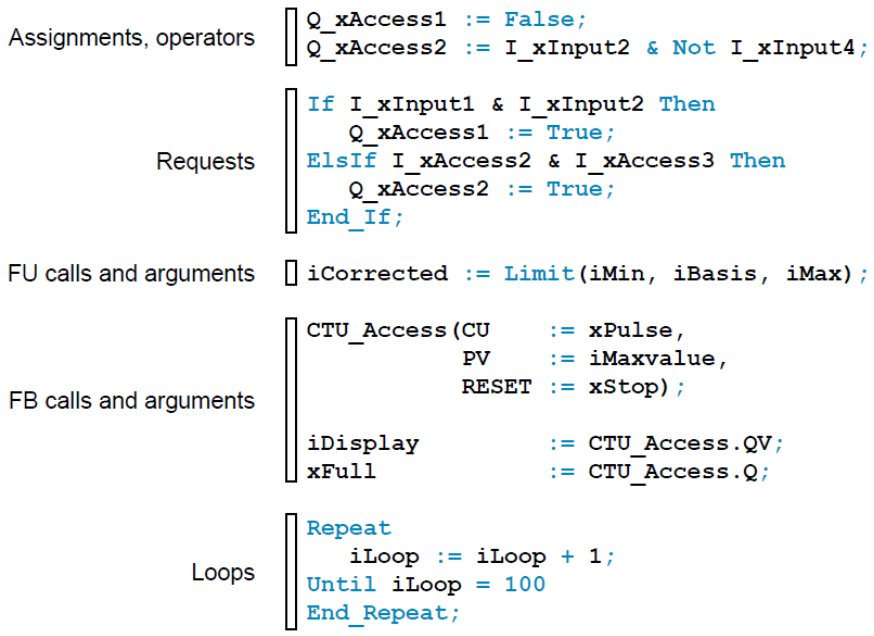


Рисунок 5.1 – Элементы языка структурированного текста

Существенное различие между графическими и текстовыми языками – то, что назначения выражений (константы, переменные и вычисления) выполняются слева направо в графических языках и справа налево в текстовых языках. Относительно набора команд, операторов и языковых элементов нет никаких различий между этими двумя типами языков программирования.

Главные элементы, которые являются основанием для присвоения – это всегда оператор присвоения := и точка с запятой для того, чтобы закончить инструкцию (рисунок 5.2).

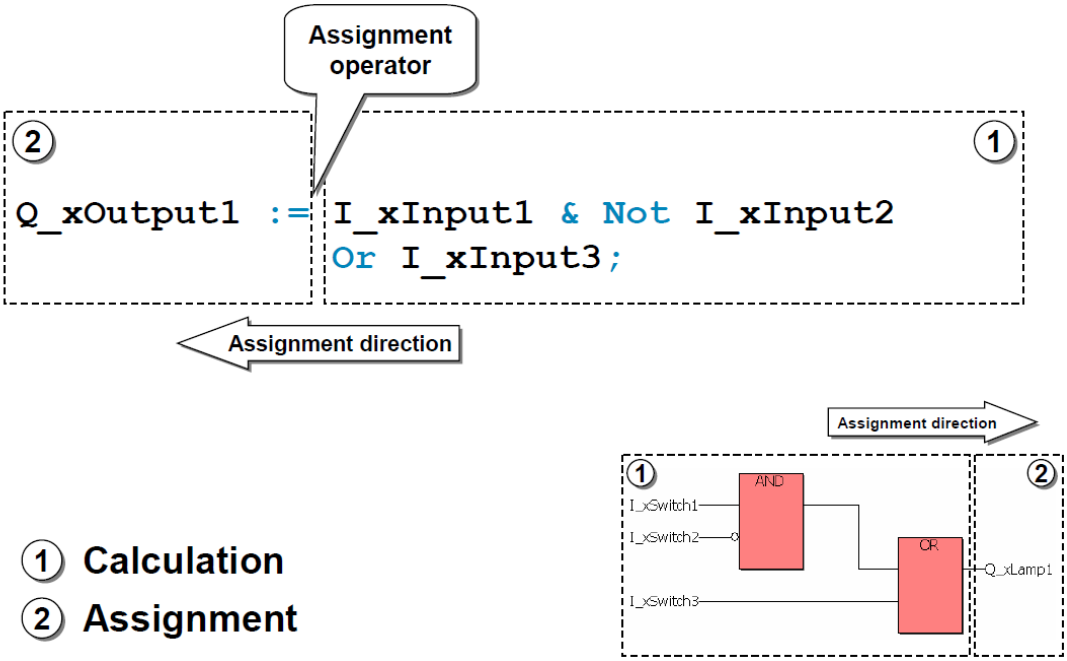


Рисунок 5.2 – Операторы структурированного языка

5.2 Иерархия операторов

Операторы, доступные в структурированном тексте, выполняются в строго определенном порядке (рисунок 5.3). Это может иметь существенное значение, если в одном выражении используется больше, чем один оператор. Используя скобки, пользователь всегда может изменять эти приоритеты. Однако скобки также могут использоваться для достижения высокой ясности в программировании.

Колонка *Data type group* (группа типа данных) в вышеупомянутом списке говорит Вам, для каких типов данных или групп типов данных могут использоваться операторы.

Priority	Operation	Symbol	Data type group
	Brackets	(Expression)	ANY
	Function evaluation	Function(Arguments)	*
	Potentialization	iNumber1 ** iNumber2*	NUM
	Negation Complement	-iNumber NOT wCode	BIT
	Multiplication Division Modulo	iNumber1 * iNumber2* rNumber1 / rNumber2* iNumber1 MOD iNumber2	NUM
	Addition Subtraction	iNumber1 + iNumber2* rNumber1 - rNumber2*	
	Comparison	diA > diB | wC < wD | iE >= iF | iG <= iH	ANY
	Equality Inequality	iNumber1 = iNumber2* rNumber1 <> rNumber2*	
	Boolean AND	xVar1 & xVar2 wCode1 AND wCode2	BIT
	Boolean exclusive OR	xVar1 XOR xVar2	
	Boolean OR	bVar1 OR bVar2	

Рисунок 5.3 – Иерархия операторов

5.3 Использование функций в структурированном тексте

Использование функций в структурированном тексте подчиняется тем же правилам, которые относятся и к другим языкам. Всем входным параметрам нужно присвоить в правильном порядке аргументы. После имени функции передаваемые аргументы перечисляются в скобках и отделяются запятыми. Рекомендуется следовать образцу, показанному на рисунке 5.4, – использовать одну строку для аргумента.

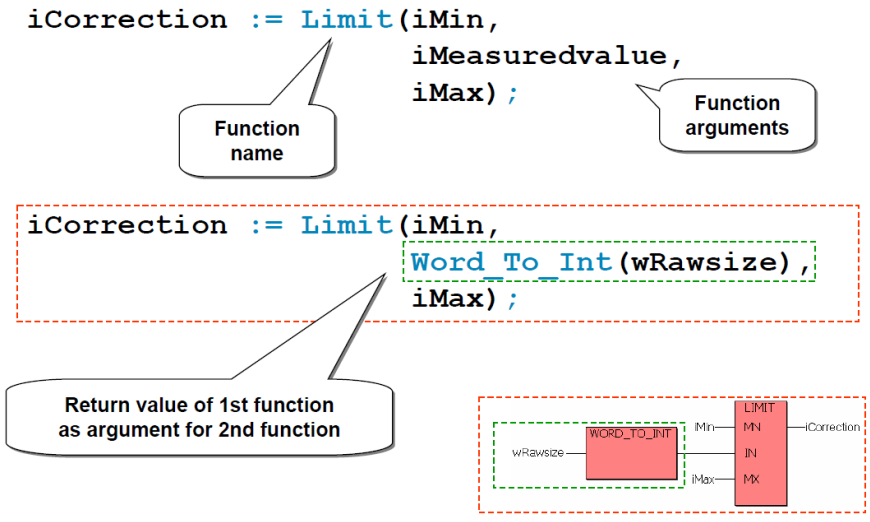


Рисунок 5.4 – Вызов функции

В самом простом случае возвращаемый функцией результат присваивается целевой переменной через оператор присвоения.

Второй пример показывает вложенные функции. Результат Word_To_Int в этом примере используется в качестве второго аргумента для функции Limit.

Вызов функционального блока в структурированном тексте выполняется, как и на других языках, в три этапа (рисунок 5.5):

1. Предоставление значений входных параметров (импорт данных).

2. Выполнение функциональности блока, в случае необходимости, при использовании сохраненных данных (вычисление).

3. Сохранение полученных значений в созданные переменные (экспорт данных).

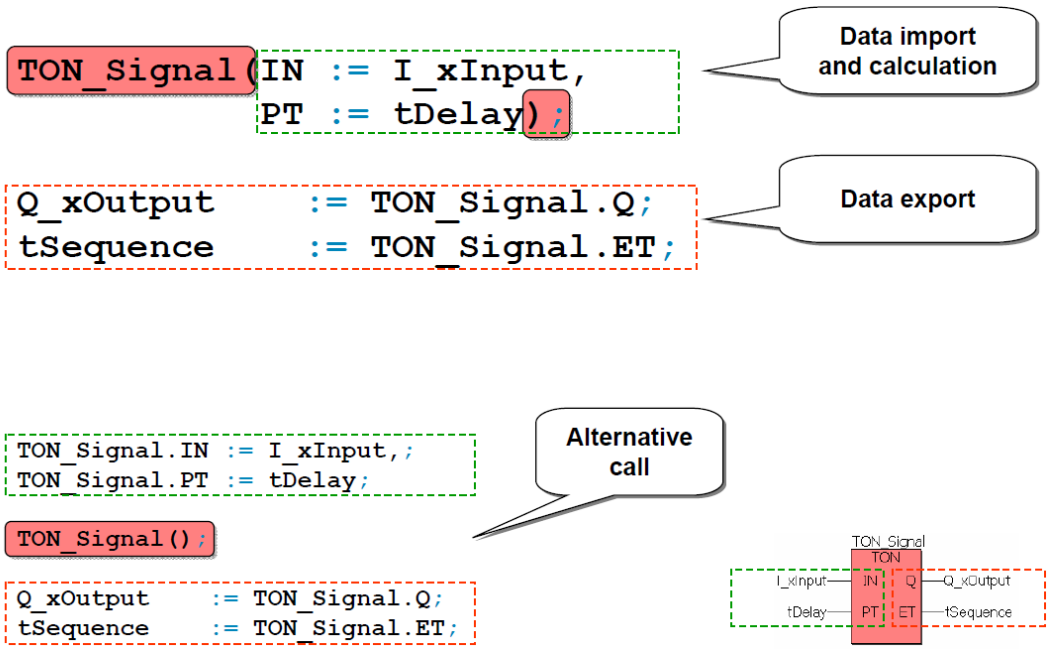


Рисунок 5.5 – Вызов функциональных блоков

При первом вызове функционального блока фактические параметры в скобках связаны с формальными параметрами функционального блока. Поэтому название блока не должно быть написано перед ними, как для альтернативного вызова. В любом случае скобки при вызове блока должны быть, даже если импорт данных производится отдельно (альтернативный вызов).

5.4 Использование функциональных блоков в структурированном тексте

На рисунке 5.6 показано сравнение вызова функций и функциональных блоков. Аргументы для функции назначаются в строгой последовательности, а возможности функциональных блоков основаны на доступе к названным параметрам входа и выхода.

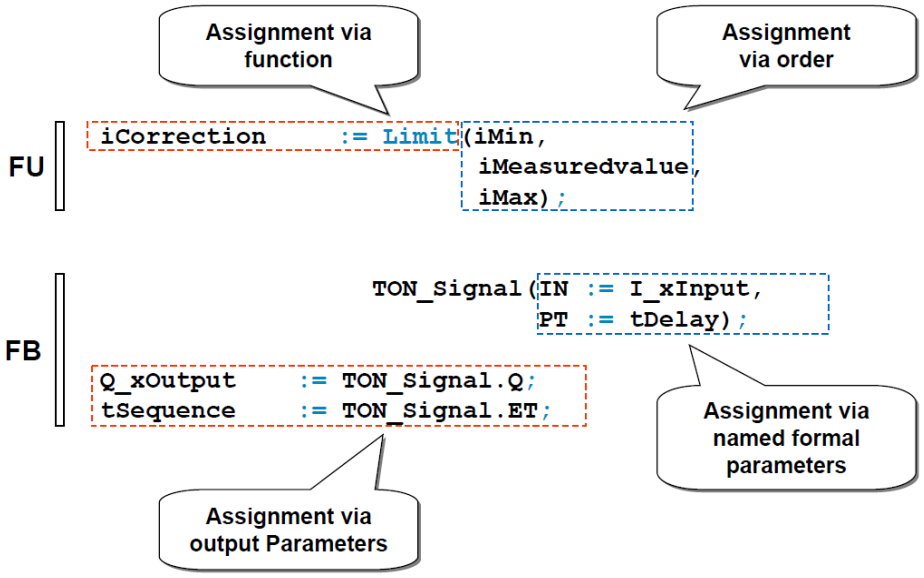


Рисунок 5.6 – Сравнение вызова функций и функциональных блоков

5.5 Операторы условий

Оператор If (если) позволяет пользователю создать программный код, выполняемый в зависимости от булевого условия (рисунок 5.7).

Эта структура – важное дополнение к остальным элементам, известным из других языков, например, инструкциям, функциям и функциональным блокам. Однако это не заменяет их.

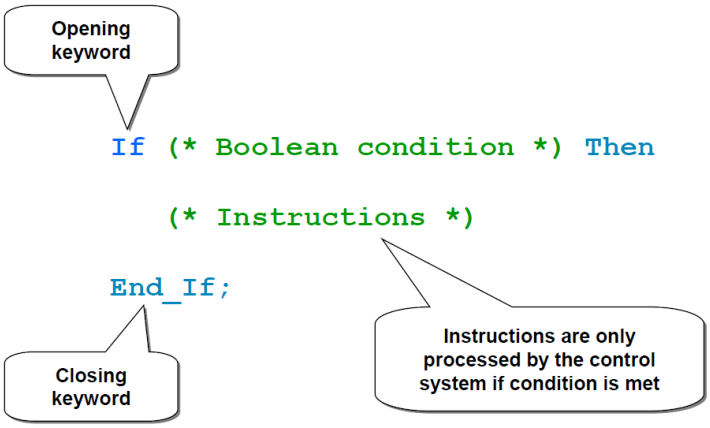


Рисунок 5.7 – Оператор условия If

Базовая структура может быть расширена при использовании двух ключевых слов:

- *ElsIf* обеспечивает альтернативное условие, если условие открытия оператора If не выполнилось, может использоваться неограниченное количество раз;

- *Else*, в отличие от *ElsIf*, может использоваться только один раз: если в пределах структуры *If* ни одно из предыдущих условий *If* и *Elsif* не было выполнено, условие *Else* будет выполнено.

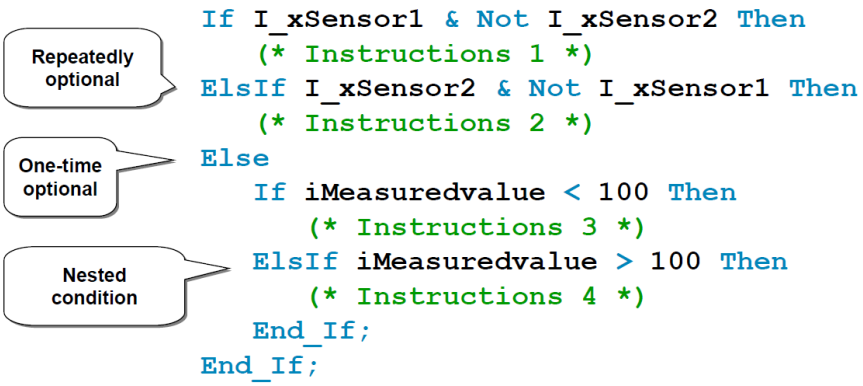


Рисунок 5.8 – Пример вложенного оператора If

Оператор Case требует целочисленного типа данных. На рисунке 5.9 показано, что в качестве определения случаев в операторе Case можно использовать целочисленные величины, отделенные запятыми величины, диапазоны величин и смешанные способы записи.

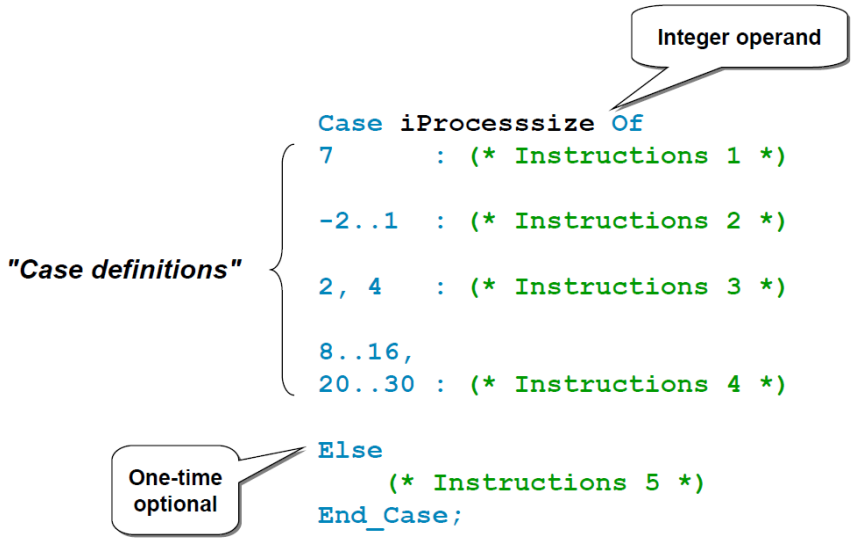


Рисунок 5.9 – Оператор Case

В операторе Case будет выполнено то задание, условие которого будет соответствовать. Условия выбора не должны повторяться.

Блок заданий Else будет выполнен только в том случае, если значение переменной не будет соответствовать требуемым условиям.

Пример использования операторов If b Case представлен на рисунке 5.10.

```
Case iProcessstep Of
0    : (* Initialize *)
       If xInit_completed Then iProcessstep := 10;
       End_If;

10   : (* Execute Prozess 1 *)
       If xProcess1_completed Then iProcessstep := 20;
       End_If;

20   : (* Execute Prozess 2 *)
       If xProcess2_completed Then iProcessstep := 30;
       End_If;

(* etc. *)

500  : (* Exceptionalhandling *)
       iProcessstep := 0;
End_Case;
```

Рисунок 5.10 – Использование операторов If и Case

5.6 Операторы цикла

При использовании оператора цикла For осуществляется циклическое повторение выполнения команд при верном условии (рисунок 5.11). Все величины от начальной до конечной присваиваются переменной цикла (здесь iIndex) через шаг приращения. Все величины должны быть целочисленными. Если ширина шага не обозначена, установлено стандартное значение INT#1.

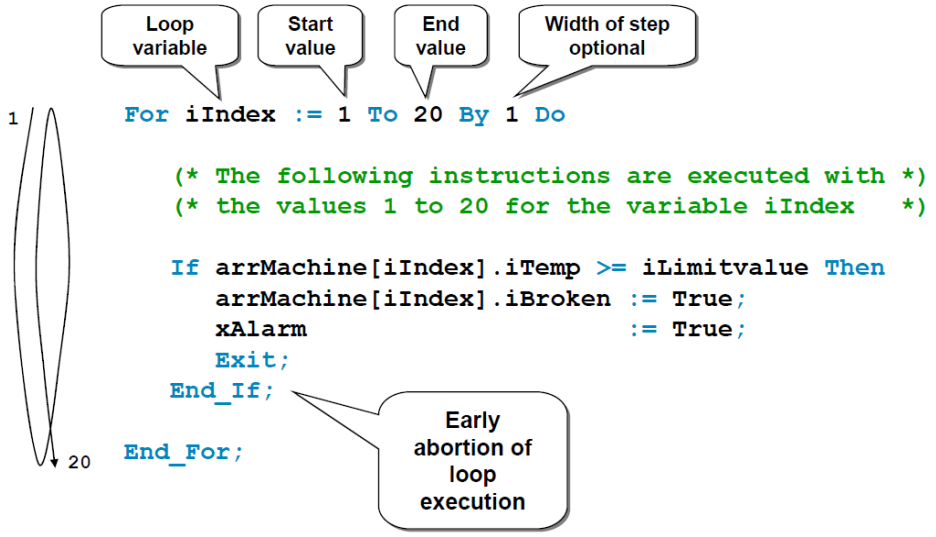


Рисунок 5.11 – Использование цикла For

Циклы Repeat и While, в отличие от цикла For, не являются предопределенными циклами (рисунок 5.12). В зависимости от значения булевой величины определяется, будет ли выполняться цикл еще раз.

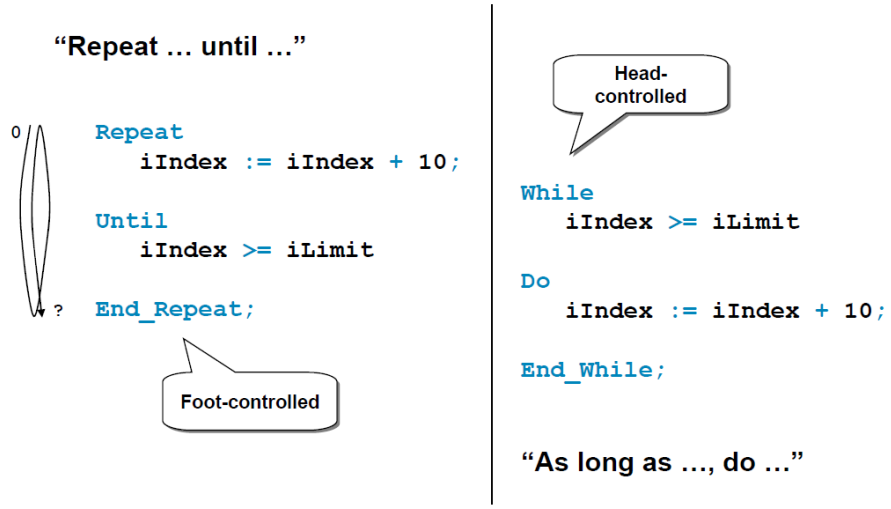


Рисунок 5.12 – Использование операторов цикла Repeat и While

5.7 Описание лабораторного макета

Общий вид лабораторного макета конвейерной линии представлен на рисунке 5.13. Линия состоит из четырех макетов, работа которых подробно описана в заданиях к Лабораторной работе №4.

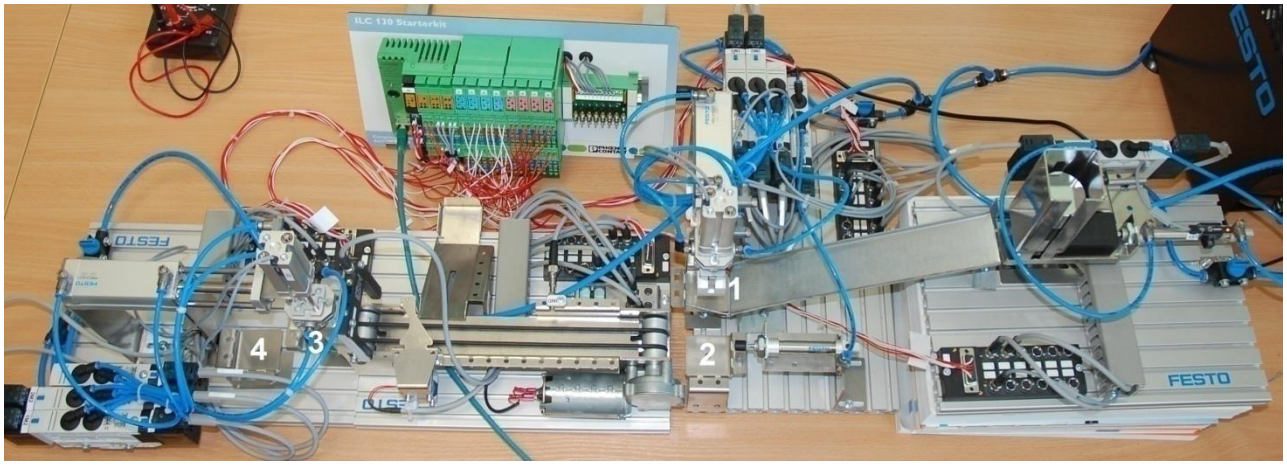


Рисунок 5.13 – Внешний вид лабораторного макета

Контрольные вопросы

1. Основные различия между графическими и текстовыми языками?
2. Иерархия операторов в структурированном тексте?
3. Как происходит использование функций в структурированном тексте?
4. Как вызвать функциональный блок в структурированном тексте?
5. Объясните использование оператора If
6. Объясните использование оператора Case
7. Приведите пример использования оператора цикла.
8. В чем отличия операторов цикла Repeat и While?

Задание

Написать управляющую программу для манипуляторов макетов 1–4 конвейерной линии (см. рисунок 5.13).

Из исходного состояния необходимо захватить фишку из позиции 1 и переместить в позицию 2 (рисунок 5.13): для этого необходимо опустить захват (координата по вертикали), захватить фишку, вернуться в исходное

положение, выдвинуть манипулятор (координата по горизонтали), опустить захват (координата по вертикали), положить фишку (на позицию 2), и вернуться в исходное состояние.

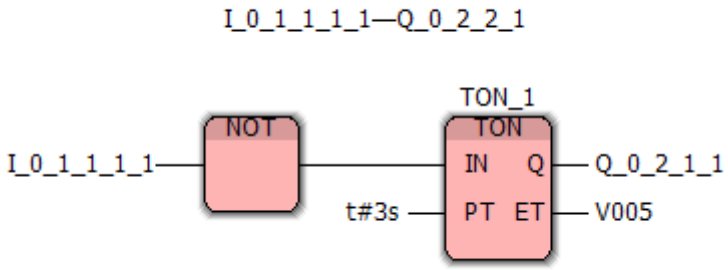


Рисунок 5.14 – Алгоритм работы первого макета

При функционировании второго манипулятора необходимо подать фишку от первого конвейера на позицию 3 второго конвейера, не нарушив нормальный режим работы второго конвейера.

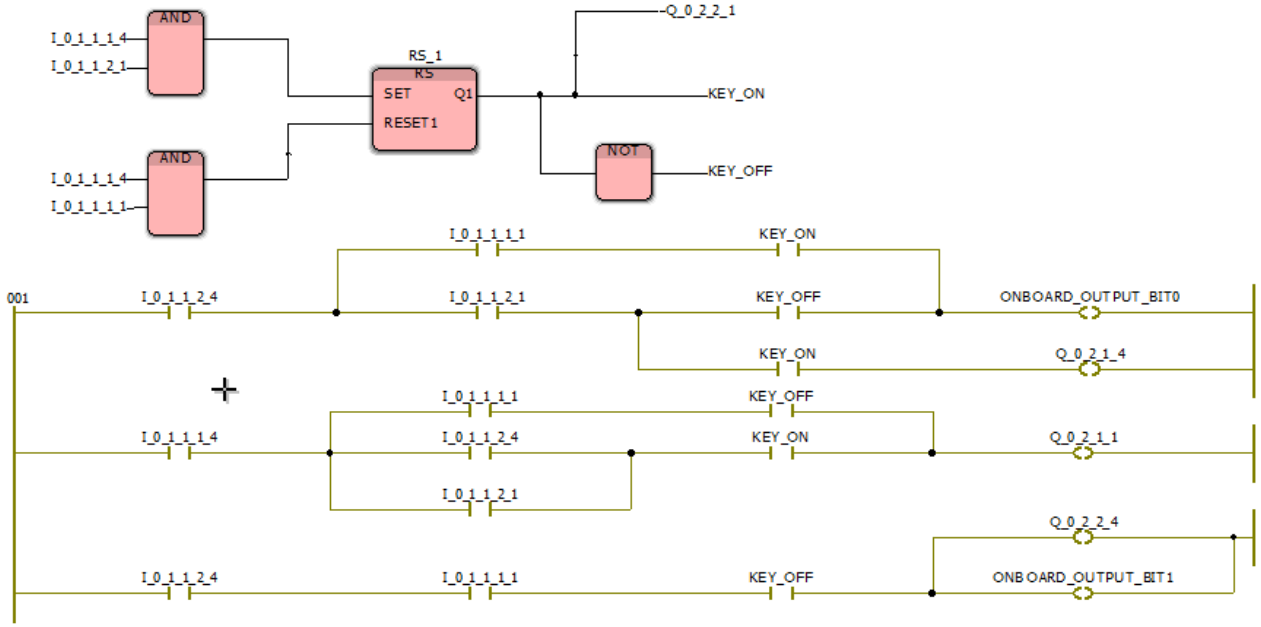


Рисунок 5.15 – Алгоритм работы второго макета

Когда фишка перемещена на позицию 2, ее необходимо столкнуть на сортировщик фишек 4 (см. рисунок 4.8). В соответствие с вариантом задания необходимо отсортировать металлические и неметаллические

фишки. Отобранные фишки, пройдя через оптический датчик (рамку 5), перемещаются на четвертый манипулятор.

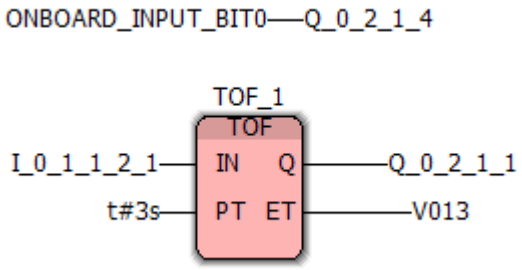


Рисунок 5.16 – Алгоритм работы третьего макета

Четвертый манипулятор по сигналу от оптического датчика 5 (рамки) должен забрать фишку с позиции 6 и переместить ее на позицию 7 (см. рисунок 4.8). Для этого четвертому манипулятору необходимо проделать следующие действия: из исходного положения необходимо выдвинуть манипулятор (координата по горизонтали), затем опустить захват (координата по вертикали), захватить фишку, вернуть захват в исходное положение (координата по вертикали), вернуться в исходное положение, опустить захват (координата по вертикали), положить фишку (на позицию 7) и вернуться в исходное положение.

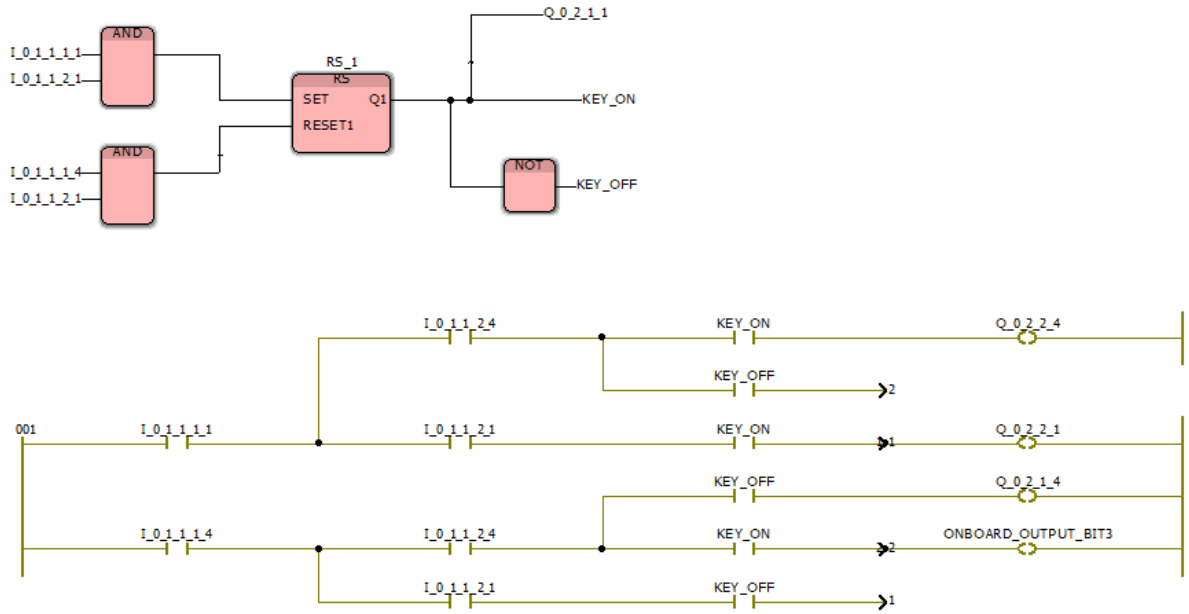


Рисунок 5.17 – Алгоритм работы четвертого макета

Примечание – В первую очередь программу необходимо написать для манипулятора 2 (см. рисунок 5.13).

Программа управления конвейерной линией представлена на рисунке 5.18.

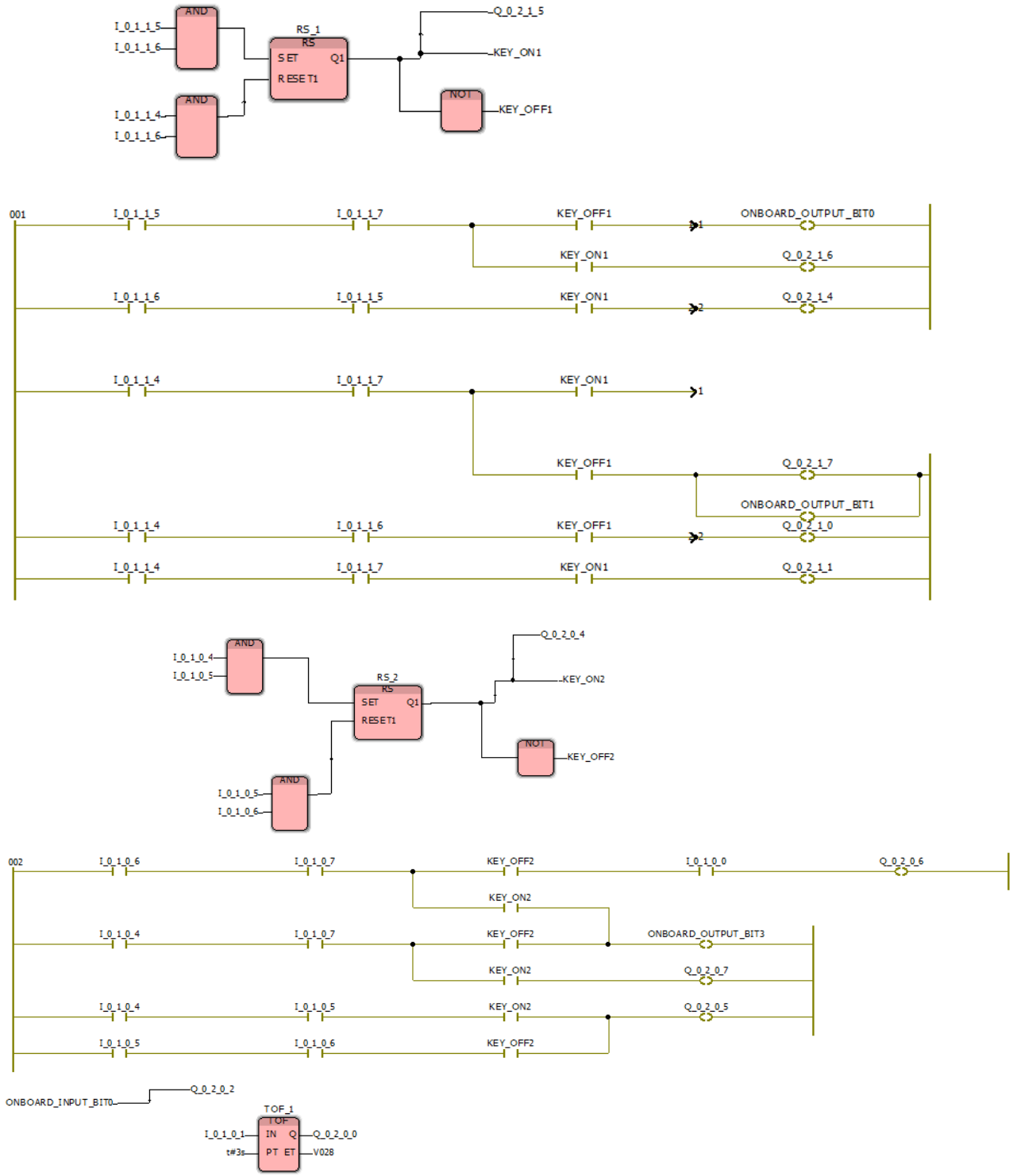


Рисунок 5.18 – Программа управления конвейерной линией

ПРИЛОЖЕНИЕ А

(справочное)

Обзор рабочих характеристик компактных устройств управления

Таблица А.1 – Обзор рабочих характеристик компактных устройств управления

Название характеристики	Значения
1	2
Интерфейсы Локальная шина INTERBUS-Master (ведущ.) Ethernet Задание параметров / обслуживание / диагностика	 Распределитель Inline Гнездо RJ45 RS-232-C, 6-контактный разъем MINI-DIN (PS/2), Ethernet 10/100 (RJ45)
INTERBUS, ведущий Количество возможных каналов параметрирования Количество точек ввода – вывода Количество поддерживаемых оконечных устройств	 Макс. 8 Макс. 4096 Макс. 63
Прямые входы/выходы Количество входов Количество выходов	 8 4
Исполняющая система, соотв. МЭК-61131 Программа Скорость обработки данных Память для программ Память для данных Память для постоянного хранения данных Количество модулей данных Количество таймеров, счетчиков Количество задач управления	 PC WORX в IEC 61131 1,7 мс (1 К смешанных команд, команда размером 1 кбит за 90 мкс) 192 Кбайта (16 К команд (IL)) 192 Кбайта 8 Кбайт (NVRAM) в зависимости от объема памяти для данных в зависимости от объема памяти для данных 8

1	2
Часы реального времени	Да
Питание Электропитание Диапазон напряжения питания Потребляемый ток, типовой	 24 В DC 19,2 В DC ... 30 В DC 210 мА (в режиме холостого прогона оконечные устройства к локальной шине не подключены, шина неактивна)
Общие характеристики ширина высота глубина степень защиты температура окружающей среды (при эксплуатации)	 80 мм 119,8 мм 71,5 мм IP20 – 25 °C ... 60 °C

ПРИЛОЖЕНИЕ Б

(справочное)

Типы данных PC WORX

Таблица Б.1 – Численные типы данных

Обозначение	Тип данных	Размер (биты)	Диапазон значений
SINT	8 bit integer (со знаковым битом)	8	–128...127
INT	integer (со знаковым битом)	16	–32 768...32 767
DINT	Double integer (со знаковым битом)	32	–2 147 483 648...2 147 483 647
LINT	64 bit integer (со знаковым битом)	64	–9 223 372 036 854 775 808... 9 223 372 036 854 775 807
USINT	8 bit integer (без знакового бита)	8	0...255
UINT	Integer (без знакового бита)	16	16 0...65 535
UDINT	Double integer (без знакового бита)	32	0...4 294 967 295
ULINT	64 bit integer (без знакового бита)	64	0…18 446 744 073 709 551 615
REAL	32-bit с плавающей точкой	32	$\pm 1.5 \cdot 10^{-45}$... $\pm 3.4 \cdot 10^{38}$
LREAL	64-bit с плавающей точкой	64	–

Таблица Б.2 – Битовые типы данных

Обозначение	Тип данных	Размер (биты)	Диапазон значений
BOOL	Bool	1	0…1HEX (и правда/ложь)
BYTE	Строка 8 бит	8	0...FFHEX
WORD	Строка 16 бит	16	0...FFFFHEX
DWORD	Строка 32 бита	32	0...FFFF FFFFHEX
LWORD	Строка 64 бита	64	0...FFFF FFFF FFFF FFFFHEX

Таблица Б.3 – Дополнительные типы данных

Обозначение	Тип данных	Размер (биты)	Диапазон значений
TIME	Время	32	0...4 294 967 295 ms (см. UDINT)
DATE	Дата	–	–
TOD	Time of day (время дня)	–	–
DT	Date and time (дата и время)	–	–
STRING	Строка (стандартно)	80 байтов	–

Литература

1. PC WORX IEC 61131-Programming [Электронный ресурс]. – Режим доступа: PCWORX_Edunet_1.0.pdf.
2. Компоненты и системы AUTOMATION [Электронный ресурс]. – Режим доступа: 52004909 AX Automation.pdf.
3. ILC 130 Starterkit [Электронный ресурс]. – 2013. – 1 электрон. опт. диск (CD-ROM).

Учебное издание

Шведова Ольга Александровна
Шило Мария Владимировна

МИКРОПРОЦЕССОРНЫЕ СИСТЕМЫ УПРАВЛЕНИЯ И СЕТЕВЫЕ ТЕХНОЛОГИИ

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

Редакторы *Е.С. Чайковская, М.А. Зайцева*
Корректор
Компьютерная правка, оригинал-макет

Подписано в печать 2015. Формат 60×84 1/16. Бумага офсетная. Гарнитура «Таймс».
Отпечатано на ризографе. Усл. печ. л.4,88. Уч.-изд. л. 5,2. Тираж 70 экз. Заказ 349.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники».
Свидетельство о государственной регистрации издателя, изготовителя,
Распространителя печатных изданий №1/238 от 24.03.2014.
№2/113 от 07.04.2014, №3/615 от 07.04.2014.
ЛП №02330/264 от 14.04.2014.
220013, Минск, П.Бровки, 6.