МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ВОЛЖСКИЙ ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ (ФИЛИАЛ)
ФЕДЕРАЛЬНОГО ГОСУДАРСТВЕННОГО БЮДЖЕТНОГО ОБРАЗОВАТЕЛЬНОГО
УЧРЕЖДЕНИЯ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОЛГОГРАДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

В. И. Капля, А.Г. Бурцев, С.И. Ефремкин

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ СИСТЕМ УПРАВЛЕНИЯ

Электронное учебное пособие

Волжский
2019

УДК 004.45(07)
ББК 32.81я73
К 203

Р е ц е н з е н т ы:
Филиал ФГБОУ ВО «Национальный исследовательский университет «МЭИ» в г. Волжском,
зав. кафедрой “Автоматизация технологических процессов и производств”, канд. тех. наук, доцент
Болдырев И. А.,
ООО «Волгопромавтоматика»
Инженер-программист второй категории
Поспеев Ю.М.

Печатается по решению редакционно-издательского совета
Волгоградского государственного технического университета

Капля, В. И.
Программное обеспечение систем управления [Электронный ресурс] : учебное пособие / В. И. Капля, А.Г. Бурцев, С.И. Ефремкин ; ВПИ (филиал) ВолгГТУ, – Электрон. текстовые дан. (1 файл: 2,26 МБ). – Волжский, 2019. – Режим доступа: http://lib.volpi.ru. – Загл. с титул. экрана.
ISBN 978-5-9948-3305-6

Содержит сведения о принципах и примерах разработки программного обеспечения для систем управления. Изложение теоретического материала сопровождается примерами решения задач АСУ в среде PC WORX.

Рекомендуется для использования в учебном процессе по техническим специальностям, в том числе по направлению 15.03.04 «Автоматизация технологических процессов и производств» (уровень бакалавриата), при изучении дисциплины «Программное обеспечение систем управления» для студентов дневной, вечерней и заочной форм обучения.

Ил. 80, табл. 5, библиограф.: 17 назв.

ISBN 978-5-9948-3305-6

Содержание

Введение 4
1. Основные определения ПОАСУ 5
2. Жизненный цикл ПОАСУ 7
2.1. Каскадная модель процесса разработки ПОАСУ 8
2.2. Инкрементная модель процесса разработки ПОАСУ 11
2.3. Спиральная модель процесса разработки ПОАСУ 14
3. Классификация программных средств АСУ ТП 18
4. Структура программных средств АСУ ТП 21
5. Среда разработки программного обеспечения PC WORX 6 31
5.1. Учебный стенд ILC 130 STARTERKIT 31
5.2. Контроллер ILC 130 ETH 32
5.3. Создание программного проекта в среде PC WORX 6 37
5.4. Создание и загрузка программы в контроллер средствами среды PC WORX 6 42
5.5. Контроль состояния переменных программы средствами среды PC WORX 6 46
6. Отладка программ средствами среды PC WORX6 в режиме симуляции 50
7. Трассировка переменных программы средствами среды PC WORX6 59
8. Создание функциональных блоков средствами среды PC WORX6 62
9. Создание программ на языке SFC средствами среды PC WORX6 66
9.1. Формирование действий на языке SFC средствами среды PC WORX6 69
9.2. Формирование переходов на языке SFC средствами среды PC WORX6 72
9.3. Формирование ветвлений на языке SFC средствами среды PC WORX6 73
Литература 79

Введение

Автоматизированное производство в качестве центрального связующего и координирующего элемента содержит системы управления производственным оборудованием. Активно применяются современные, постоянно совершенствуемые цифровые системы управления, последовательность действий которых определяется их программным обеспечением. Программируемые логические контроллеры (ПЛК), оснащенные современным программным обеспечением способны реализовать автоматическое логическое управление технологическими процессами, находящимися на высоком уровне сложности, требующими применения интеллектуальных алгоритмов управления. Сетевые структуры формирования и обработки потоков управляющей информации в реальном времени позволяют организовать процесс управления с высокой степенью оперативности и информационной достаточности. Высокое быстродействие цифровых систем управления позволяет возложить на них контроль и обеспечение безопасности реализации технологических процессов. Разработка программного обеспечения автоматизированных систем управления (ПОАСУ) является важным этапом создания АСУ, для реализации которого требуются глубокие знания в программировании ПЛК с учетом принципов и особенной автоматизируемого процесса.

В данном пособии рассматриваются вопросы структуры и алгоритмической реализации программного обеспечения систем управления. Основное внимание уделяется структуре и примерам разработки программного обеспечения систем управления. Изложение теоретического материала сопровождается примерами решения задач АСУ в среде PC WORX.

1. Основные определения ПОАСУ

В мировой практике принято выделять пять уровней управления современным предприятием.

На первом (верхнем) уровне управления осуществляются расчет и анализ финансово-экономических показателей, решаются стратегические административные и логистические задачи. Этот уровень обозначают: ***ERP – Enterprise Resource Planning*** (планирование ресурсов предприятия).

На втором уровне решаются задачи управления качеством продукции, планирования и контроля последовательности операций технологического процесса, управления производственными и людскими ресурсами в рамках технологического процесса, технического обслуживания производственного оборудования. Этот уровень обозначают: ***MES – Manufacturing Execution Systems*** (системы производственного исполнения).

Эти два уровня относятся к задачам АСУП (автоматизированным системам управления предприятием) и технические средства, с помощью которых эти задачи реализуются – это офисные персональные компьютеры (ПК) и рабочие станции на их основе в службах главных специалистов предприятия.

На следующих трех уровнях решаются задачи, которые относятся к классу АСУ ТП (автоматизированных систем управления технологическими процессами).

На третьем уровне реализуются алгоритмы тактического оперативного управления, основанного на решении задач оптимизации, диагностики, адаптации, прогнозирования и т.п. Этот уровень обозначают: ***SCADA – Supervisory Control and Data Acquisition*** (система сбора данных и супервизорного диспетчерского управления).

На четвертом уровне осуществляется непосредственное (локальное) управление, которое реализуется с помощью панелей и пультов управления, программируемых логических контроллеров, устройствами связи с объектом. Этот уровень обозначают: ***Control-level*** (уровень управления).

На пятом (нижнем) уровне осуществляется опрос датчиков и управление исполнительными механизмами конкретных технологических установок и рабочих машин. Этот уровень обозначают: ***Input/Output*** (Входы/Выходы).

Уровни управления в процессе реализации предполагают активное использование средств визуализации и управления ходом технологического процесса. Данные средства обозначают ***HMI – Human-Machine Interface*** (человеко-машинная связь).

Автоматизированная система управления технологическим процессом (АСУ ТП) – комплекс программных и технических средств, предназначенный для автоматизации управления технологическим оборудованием на предприятиях. Под АСУ ТП обычно понимается комплексное решение, обеспечивающее автоматизацию основных технологических операций на производстве в целом или каком-то его участке, выпускающем относительно завершенный продукт.

В общем случае, систему управления можно рассматривать в виде совокупности взаимосвязанных управленческих процессов и объектов. Обобщенной целью автоматизации управления является повышение эффективности использования потенциальных возможностей объекта управления. Таким образом, можно выделить ряд целей:

1. Предоставление лицу, принимающему решение (ЛПР), релевантных данных для принятия решений.

2. Ускорение выполнения отдельных операций по сбору и обработке данных.
3. Снижение количества решений, которые должно принимать ЛПР.
4. Повышение уровня контроля и исполнительской дисциплины.
5. Повышение оперативности управления.
6. Снижение затрат ЛПР на выполнение вспомогательных процессов.
7. Повышение степени обоснованности принимаемых решений.

2. Жизненный цикл ПОАСУ

Жизненный цикл программного обеспечения АСУ – это период времени, который начинается с момента принятия решения о необходимости создания программного продукта и заканчивается в момент его полного изъятия из эксплуатации.

ПОАСУ является составной неотъемлемой частью АСУ ТП, поэтому жизненный цикл АСУ ТП является определяющим для жизненного цикла ПОАСУ. Изменение структуры или характеристик АСУ ТП неизбежно требует соответствующей коррекции ПОАСУ, а в некоторых случаях полностью сводится к коррекции такого вида. Однако возможна модернизация ПОАСУ по внутренним причинам, например, устранение выявленных недостатков, переход на новое оборудование или на новую операционную систему.

Жизненный цикл можно представить в виде моделей. В настоящее время наиболее распространенными являются: ***каскадная, инкрементная*** (поэтапная модель с промежуточным контролем) ***и спиральная модели жизненного цикла.***

2.1. Каскадная модель процесса разработки ПОАСУ

Каскадная модель – модель процесса разработки программного обеспечения, жизненный цикл которой выглядит как поток, последовательно проходящий фазы анализа требований, проектирования, реализации, тестирования, интеграции и поддержки. Связь этапов каскадной модели показана на рис.2.1.

Рис.2.1. Каскадная модель жизненного цикла ПОАСУ

Процесс разработки реализуется с помощью упорядоченной последовательности независимых шагов. Модель предусматривает, что каждый последующий шаг начинается после полного завершения выполнения предыдущего шага. На всех шагах модели выполняются вспомогательные и организационные процессы и работы, включающие управление проектом, оценку и управление качеством, верификацию и аттестацию, менеджмент конфигурации, разработку документации. В результате завершения шагов формируются промежуточные продукты, которые не могут изменяться на последующих шагах.

Жизненный цикл традиционно разделяют на следующие основные этапы:

- анализ и разработка требований,
- проектирование,
- кодирование (программирование),

- тестирование и отладка,
- эксплуатация и сопровождение.

Достоинства каскадной модели ПОАСУ:

- стабильность требований в течение всего жизненного цикла разработки;
- формирование на каждой стадии законченного набора проектной документации, отвечающей критериям полноты и согласованности;
- определенность и понятность шагов модели и простота её применения;
- выполняемые в логической последовательности этапы работ позволяют планировать сроки завершения всех работ и соответствующие ресурсы (денежные, временные, материальные и людские).

Каскадная модель хорошо зарекомендовала себя при построении относительно простых вариантов ПО, когда в самом начале разработки можно достаточно точно и полно сформулировать все требования к продукту.

Недостатки каскадной модели ПОАСУ:

- сложность чёткого формулирования требований и невозможность их динамического изменения на протяжении жизненного цикла автоматической системы управления;
- низкая гибкость в управлении проектом;
- необходимость возврата к предыдущим шагам для решения возникающих проблем приводит к увеличению затрат и нарушению графика работ;
- непригодность промежуточного продукта для использования;
- невозможность гибкого моделирования уникальных систем;
- позднее обнаружение проблем, связанных со сборкой, в связи с одновременной интеграцией всех результатов в конце разработки;

- недостаточное участие пользователя в создании системы – в самом начале (при разработке требований) и в конце (во время приёмочных испытаний);
- пользователи не могут убедиться в качестве разрабатываемого продукта до окончания всего процесса разработки. Они не имеют возможности оценить качество, т.к. нельзя увидеть готовый продукт разработки;
- у пользователя нет возможности постепенно привыкнуть к системе. Процесс обучения происходит в конце жизненного цикла, когда ПО уже запущено в эксплуатацию;
- каждая фаза является предпосылкой для выполнения последующих действий, что превращает такой метод в рискованный выбор для систем, не имеющих аналогов, т.к. он не поддается гибкому моделированию.

Реализовать каскадную модель жизненного цикла затруднительно ввиду сложности разработки ПОАСУ без возвратов к предыдущим шагам и изменения их результатов для устранения возникающих проблем.

Ограничение области применения каскадной модели определяется её недостатками. Её использование наиболее эффективно в следующих случаях:

- при разработке проектов с четкими, неизменяемыми в течение жизненного цикла требованиями, понятными реализацией и техническими методиками;
- при разработке проекта, ориентированного на построение системы или продукта такого же типа, как уже разрабатывались разработчиками ранее;
- при разработке проекта, связанного с созданием и выпуском новой версии уже существующего продукта или системы;

- при разработке проекта, связанного с переносом уже существующего продукта или системы на новую платформу;
- при выполнении больших проектов, в которых задействовано несколько больших команд разработчиков.

2.2. Инкрементная модель процесса разработки ПОАСУ

Инкрементная модель (англ. increment – увеличение, приращение) подразумевает разработку программного обеспечения с линейной последовательностью стадий, но в несколько инкрементов (версий), т.е. с запланированным промежуточным контролем и улучшением продукта за все время пока жизненный цикл разработки ПОАСУ не подойдет к окончанию. Связь этапов инкрементной модели показана на рис.2.2.

Рис.2.2. Инкрементная модель жизненного цикла ПОАСУ

Разработка программного обеспечения ведется итерациями с циклами обратной связи между этапами. Межэтапные корректировки позволяют учитывать реально существующее взаимовлияние результатов разработки на различных этапах, время жизни каждого из этапов растягивается на весь период разработки.

В начале работы над проектом определяются все основные требования к системе, подразделяются на более и менее важные. После чего выполняется разработка системы по принципу приращений так,

чтобы разработчик мог использовать данные, полученные в ходе разработки ПО. Каждый инкремент должен добавлять системе определенную функциональность. При этом выпуск начинают с компонентов с наивысшим приоритетом. Когда части системы определены, берут первую часть и начинают её детализировать, используя для этого наиболее подходящий процесс. В то же время можно уточнять требования и для других частей, которые в текущей совокупности требований данной работы были заморожены. Если есть необходимость, можно вернуться позже к этой части. Если часть готова, она поставляется клиенту, который может использовать её в работе. Это позволит клиенту уточнить требования для следующих компонентов. Затем занимаются разработкой следующей части системы. Ключевые этапы этого процесса – простая реализация подмножества требований к программе и совершенствование модели в серии последовательных релизов до тех пор, пока не будет реализовано ПО во всей полноте.

Жизненный цикл данной модели характерен при разработке сложных и комплексных систем, для которых имеется четкое видение (как со стороны заказчика, так и со стороны разработчика) того, что собой должен представлять конечный результат. Разработка версиями ведется в силу разного рода причин:

- отсутствия у заказчика возможности сразу профинансировать весь дорогостоящий проект;
- отсутствия у разработчика необходимых ресурсов для реализации сложного проекта в сжатые сроки;
- требований поэтапного внедрения и освоения продукта конечными пользователями. Внедрение всей системы сразу может вызвать у её пользователей неприятие и только «затормозить» процесс перехода на новые технологии. Образно говоря, они могут просто «не переварить большой кусок, поэтому его надо измельчить и давать по частям».

Достоинства и недостатки этой модели (стратегии) такие же, как и у каскадной (классической модели жизненного цикла). Но в отличие от классической стратегии заказчик может раньше увидеть результаты. Уже по результатам разработки и внедрения первой версии он может незначительно изменить требования к разработке, отказаться от нее или предложить разработку более совершенного продукта с заключением нового договора.

Достоинства инкрементной модели ПОАСУ:

- затраты, которые получаются в связи с изменением требований пользователей, уменьшаются, повторный анализ и совокупность документации значительно сокращаются по сравнению с каскадной моделью;
- легче получить отзывы от клиента о проделанной работе – клиенты могут озвучить свои комментарии в отношении готовых частей и могут видеть, что уже сделано, т.к. первые части системы являются прототипом системы в целом.
- клиент имеет возможность быстро получить и освоить программное обеспечение – клиенты могут получить реальные преимущества от системы раньше, чем это было бы возможно с каскадной моделью.

Недостатки инкрементной модели ПОАСУ:

- менеджеры должны постоянно измерять прогресс процесса, в случае быстрой разработки не стоит создавать документы для каждого минимального изменения версии;
- структура системы имеет тенденцию к ухудшению при добавлении новых компонентов – постоянные изменения нарушают структуру системы. Чтобы избежать этого требуется дополнительное время и деньги на рефакторинг. Плохая структура делает программное обеспечение сложным и дорогостоящим для последующих изменений.

А прерванный жизненный цикл ПОАСУ приводит еще к большим потерям.

Схема не позволяет оперативно учитывать возникающие изменения и уточнения требований к ПОАСУ. Согласование результатов разработки с пользователями производится только в точках, планируемых после завершения каждого этапа работ, а общие требования к ПОАСУ зафиксированы в виде технического задания на всё время её создания. Таким образом, пользователи зачастую получают программный продукт, не удовлетворяющий их реальным потребностям.

2.3. Спиральная модель процесса разработки ПОАСУ

Спиральная модель состоит в том, что жизненный цикл ПОАСУ состоит из последовательности повторяющихся этапов, которые можно сопоставить спирали, на каждом витке которой выполняется создание очередной версии продукта, уточняются требования проекта, определяется его качество и планируются работы для следующего витка.

Особое внимание уделяется начальным этапам разработки – анализу и проектированию, где реализуемость принятых технических решений проверяется и обосновывается посредством создания прототипов. Связь этапов спиральной модели показана на рис.2.3.

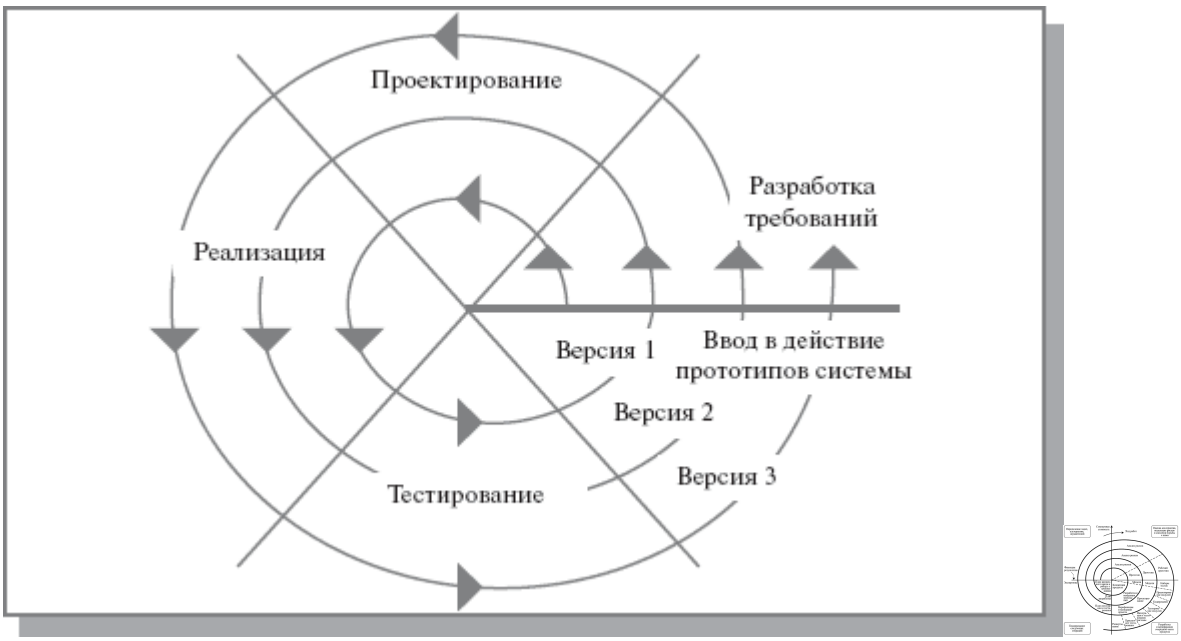


Рис.2.3. Спиральная модель жизненного цикла ПОАСУ

Данная модель представляет собой процесс разработки программного обеспечения, сочетающий в себе как проектирование, так и поэтапное прототипировнаие с целью сочетания преимуществ восходящей и нисходящей концепции, делающий упор на начальные этапы жизненного цикла: анализ и проектирование. Отличительной особенностью этой модели является специальное внимание к рискам, влияющим на организацию жизненного цикла.

На этапах анализа и проектирования реализуемость технических решений и степень удовлетворения потребностей заказчика проверяется путем создания прототипов. Каждый виток спирали соответствует созданию работоспособного фрагмента или версии системы. Это позволяет уточнить требования, цели и характеристики проекта, определить качество разработки, спланировать работы следующего витка спирали. Таким образом, углубляются и последовательно конкретизируются детали проекта, и в результате выбирается обоснованный вариант, который удовлетворяет действующим требованиям заказчика и доводится до реализации.

Жизненный цикл на каждом витке спирали позволяет применять разные модели процесса разработки ПО. В конечном итоге на выходе получается готовый продукт. Модель сочетает в себе возможности модели прототипирования и водопадной модели. Разработка итерациями отражает объективно существующий спиральный цикл создания системы. Неполное завершение работ на каждом этапе позволяет переходить на следующий этап, не дожидаясь полного завершения работы на текущем этапе. Главная задача – как можно быстрее показать пользователям системы работоспособный продукт, что активизирует процесс уточнения и дополнения требований.

Достоинства спиральной модели ПОАСУ:

- модель позволяет быстрее показать пользователям системы работоспособный продукт, тем самым, активизируя процесс уточнения и дополнения требований;
- модель допускает изменение требований при разработке программного обеспечения, что характерно для большинства разработок, в том числе и типовых;
- в модели предусмотрена возможность гибкого проектирования, поскольку в ней воплощены преимущества каскадной модели, и в то же время разрешены итерации по всем фазам этой же модели;
- позволяет получить более надежную и устойчивую систему. По мере развития программного обеспечения ошибки и слабые места обнаруживаются и исправляются на каждой итерации;
- модель разрешает пользователям активно принимать участие при планировании, анализе рисков, разработке, а также при выполнении оценочных действий;
- уменьшаются риски заказчика. Заказчик может с минимальными для себя финансовыми потерями завершить развитие неперспективного проекта;

- обратная связь по направлению от пользователей к разработчикам выполняется с высокой частотой и на ранних этапах модели, что обеспечивает создание нужного продукта высокого качества.

Недостатки спиральной модели ПОАСУ:

- если проект имеет низкую степень риска или небольшие размеры, модель может оказаться дорогостоящей. Оценка рисков после прохождения каждой спирали связана с большими затратами;
- жизненный цикл модели имеет усложненную структуру, поэтому может быть затруднено её применение разработчиками, менеджерами и заказчиками;
- спираль может продолжаться до бесконечности, поскольку каждая ответная реакция заказчика на созданную версию может порождать новый цикл, что отдаляет окончание работы над проектом;
- большое количество промежуточных циклов может привести к необходимости в обработке дополнительной документации;
- использование модели может оказаться дорогостоящим и даже недопустимым по средствам, т.к. время, затраченное на планирование, повторное определение целей, выполнение анализа рисков и прототипирование, может быть чрезмерным;
- могут возникнуть затруднения при определении целей и стадий, указывающих на готовность продолжать процесс разработки на следующей спирали модели.

Основная проблема спирального цикла – определение момента перехода на следующий этап. Для её решения вводятся временные ограничения на каждом из этапов жизненного цикла, а переход осуществляется в соответствии с планом, даже если не вся запланированная работа закончена. Планирование производится на основе статистических данных, полученных в предыдущих проектах и личного опыта разработчиков.

Применение спиральной модели целесообразно в следующих случаях:

- при разработке проектов, использующих новые технологии;
- при разработке новой серии продуктов или систем;
- при разработке проектов с ожидаемыми существенными изменениями или дополнениями требований;
- для выполнения долгосрочных проектов;
- при разработке проектов, требующих демонстрации качества и версий системы или продукта через короткий период времени;
- при разработке проектов, для которых необходим подсчет затрат, связанных с оценкой и разрешением рисков.

3. Классификация программных средств АСУ ТП

В типовой архитектуре АСУ ТП определяют два уровня:

• уровень локальных контроллеров, взаимодействующих с объектом управления посредством датчиков и исполнительных устройств;

• уровень оперативного управления технологическим процессом, основными компонентами которого являются серверы, рабочие станции операторов/диспетчеров, АРМ специалистов.

Каждый из этих уровней функционирует под управлением специализированного программного обеспечения. Разработка ПОАСУ или его выбор из предлагаемых в настоящее время на рынке программных средств зависит от многих факторов, прежде всего от решаемых на конкретном уровне задач. Различают базовое и прикладное программное обеспечение, структура которого показана на рис.3.1.

Рис.3.1. Классификация программных средств системы управления

Базовое программное обеспечение включает в себя различные компоненты, но основным из них является операционная система (ОС) программно-технических средств АСУТП.

Каждый уровень АСУТП представлен «своими» программно-техническими средствами: на нижнем уровне речь идет о контроллерах, тогда как основным техническим средством верхнего уровня является компьютер. В соответствии с этим в кругу специалистов появилась и такая классификация: встраиваемое и настольное программное обеспечение.

Требования, предъявляемые к встраиваемому и настольному ПО, различны. Контроллер в системе управления наряду с функциями сбора информации решает задачи автоматического непрерывного или логического управления. В связи с этим к нему предъявляются жесткие требования по времени реакции на состояние объекта и выдачи управляющих воздействий на исполнительные устройства. Контроллер должен гарантированно откликаться на изменения состояния объекта за заданное время.

Выбор операционной системы программно-технических средств верхнего уровня АСУТП определяется прикладной задачей (ОС общего пользования или ОСРВ). Но наибольшую популярность и распространение

получили различные варианты ОС Windows. Ими оснащены программно-технические средства верхнего уровня АСУТП, представленные персональными компьютерами разной мощности и конфигурации – рабочие станции операторов/диспетчеров и специалистов, серверы баз данных (БД) и т. д.

Основные преимущества ОС Windows:

• Windows имеет очень широкое распространение в мире, в связи с чем легко найти специалиста, который мог бы сопровождать системы на базе этой ОС;

• Windows имеет множество приложений, обеспечивающих решение различных задач обработки и представления информации;

• ОС Windows и Windows-приложения просты в освоении и обладают типовым интуитивно понятным интерфейсом;

• приложения, работающие под управлением Windows, поддерживают общедоступные стандарты обмена данными;

• системы на базе ОС Windows просты в эксплуатации и развитии, что делает их экономичными как с точки зрения поддержки, так и при поэтапном росте;

• Microsoft развивает информационные технологии для Windows высокими темпами, что позволяет компаниям, использующим эту платформу «идти в ногу со временем».

Неотъемлемой частью верхнего уровня АСУ ТП является человек, время реакции которого на события недетерминировано и зачастую достаточно велико. Да и сама проблема реального времени на верхнем уровне не столь актуальна.

Для функционирования системы управления необходим и еще один тип ПО – прикладное программное обеспечение (ПОАСУ). Известны два пути разработки прикладного программного обеспечения систем управления:

• создание собственного прикладного программного обеспечения с использованием средств традиционного программирования (стандартные языки программирования, средства отладки и т.д.);
• использование для разработки прикладного ПО существующих (готовых) инструментальных средств.

Программные средства верхнего уровня АСУТП (SCADA-пакеты) предназначены для создания прикладного программного обеспечения пультов контроля и управления, реализуемых на различных компьютерных платформах и специализированных рабочих станциях.

SCADA–пакеты позволяют при минимальной доле программирования на простых языковых средствах разрабатывать многофункциональный интерфейс, обеспечивающий оператора/диспетчера не только полной информацией о технологическом процессе, но и возможностью им управлять.

В своем развитии SCADA–пакеты прошли тот же путь, что и программное обеспечение для программирования контроллеров. Начиная с 90-х годов, появились универсальные (открытые) SCADA–программы. Понятие открытости является фундаментальным, когда речь идет о программно-аппаратных средствах для построения многоуровневых систем автоматизации.

Перечень наиболее популярных в России SCADA–пакетов:

• Trace Mode/Трейс Моуд (AdAstrA) – Россия;
• InTouch (Wonderware) – США;
• FIX (Intellution) – США;
• Genesis (Iconics Co) – США;
• Citect (CI Technology) – Австралия;
• WinCC (Siemens) – Германия;
• RTWin (SWD Real Time Systems) – Россия.

4. Структура программных средств АСУ ТП

Задача организации многопользовательских систем управления, то есть систем, способных поддерживать достаточно большое количество АРМ пользователей (клиентов), была решена путем создания клиент–серверной технологии и/или архитектуры. Клиент-серверная архитектура характеризуется наличием двух взаимодействующих самостоятельных процессов – клиента и сервера, которые, в общем случае, могут выполняться на разных компьютерах, обмениваясь данными по сети. По такой схеме могут быть построены системы управления технологическими процессами, системы обработки данных на основе СУБД.

Клиент-серверная архитектура предполагает, что вся информация о технологическом процессе от контроллеров собирается и обрабатывается на сервере ввода/вывода (сервер базы данных), к которому по сети подключаются АРМ клиенты, как показано на рис.4.1.

Рис.4.1. Клиент-серверная архитектура

Под станцией-сервером в этой архитектуре следует понимать компьютер со специальным программным обеспечением для сбора и хранения данных и последующей их передачи по каналам связи оперативному персоналу для контроля и управления технологическим процессом, а также всем заинтересованным специалистам и руководителям. По определению сервер является поставщиком информации, а клиент – ее потребителем. Таким образом, рабочие станции операторов/диспетчеров, специалистов, руководителей являются станциями-клиентами.

Обычно клиентом служит настольный ПК, выполняющий программное обеспечение конечного пользователя. ПО клиента – это любая прикладная программа или пакет, способные направлять запросы по сети серверу и обрабатывать получаемую в ответ информацию. Естественно, функции клиентских станций, а следовательно, и программное обеспечение различны и определяются функциями рабочего места, которое они обеспечивают.

Количество операторских станций, серверов ввода/вывода (серверов БД) определяется на стадии проектирования и зависит, прежде всего, от объема перерабатываемой в системе информации. Для небольших систем управления функции сервера ввода/вывода и станции оператора (HMI) могут быть совмещены на одном компьютере. В сетевых распределенных системах средствами SCADA/HMI стало возможным создавать станции (узлы) различного функционального назначения: станции операторов/диспетчеров, серверы с функциями HMI, «слепые» серверы (без функций HMI), станции мониторинга (только просмотр без прав на управление) для специалистов и руководителей и другие.

SCADA-программы имеют в своем составе два взаимозависимых модуля: Development (среда разработки проекта) и Runtime (среда исполнения). В целях снижения стоимости проекта эти модули могут устанавливаться на разные компьютеры. Например, станции оператора, как правило, являются узлами Runtime (или View) с полным набором функций человеко-машинного интерфейса. При этом хотя бы один компьютер в сети должен быть типа Development. На таких узлах проект разрабатывается, корректируется, а также может и исполняться. Некоторые SCADA-системы допускают внесение изменений в проект без остановки работы всей системы. Программное обеспечение SCADA-серверов позволяет создавать полный проект системы управления, включая базу данных и HMI.

Важным аспектом в структурном построении сетевых систем управления является структура базы данных реального времени (централизованная или распределенная). Каждая из структур в SCADA/HMI-системах реализуется разными разработчиками по-разному. От реализации существенно зависят эффективность обеспечения единства и целостности базы данных, ее надежность, возможности модификации и скорость обработки запросов.

В одних случаях для доступа к данным на компьютере-клиенте создается «своя» база данных, копируемая с удаленных серверов. Дублирование данных может привести к определенным проблемам с точки зрения целостности базы данных и производительности системы управления. При модификации базы данных с такой организацией, например, при введении дополнительной переменной потребуются изменения в каждой сетевой копии, использующей эту переменную.

В других случаях компьютерам-клиентам не требуются копии баз данных. Они получают необходимую им информацию по сети от сервера, в задачу которого входит подержание базы данных. Серверов может быть несколько, и любая часть данных хранится только в одном месте, на одном сервере. Поэтому и модификация базы данных производится только на одном компьютере – сервере базы данных, что обеспечивает ее единство и целостность. Такой подход к структурному построению системы снижает нагрузку на сеть и дает еще целый ряд преимуществ.

С точки зрения структурного построения SCADA-пакетов различают:

• системы, обеспечивающие полный набор базовых функций HMI;

• системы, состоящие из модулей, реализующих отдельные функции HMI.

Системы, обеспечивающие полный набор базовых функций, могут комплектоваться дополнительными опциями, реализующими

необязательные в применении функции контроля и управления, как показано на рис.4.2.

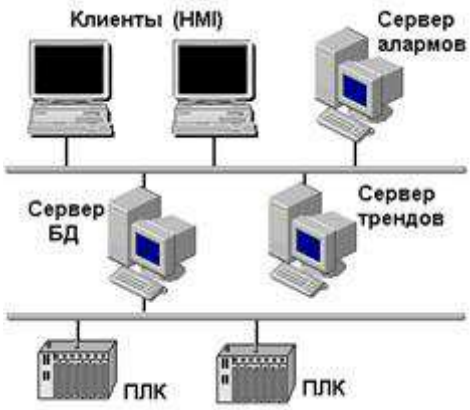


Рис.4.2. Архитектура модульной SCADA

Во втором случае система создается полностью модульной (сервер ввода/вывода, сервер алармов, сервер трендов, и т.д.). Для небольших проектов все модули могут исполняться на одном компьютере. В проектах с большим количеством переменных модули можно распределить на несколько компьютеров в разных сочетаниях. Вариант клиент-серверной архитектуры такой системы представлен на рис.4.2.

В клиент-серверной архитектуре системы управления, представленной на рис.4.2, функции сбора и хранения данных, управления алармами и трендами распределены между тремя серверами. Функция HMI реализуется на станциях-клиентах.

Например, SCADACitect имеет в своем составе пять функциональных модулей (серверов или клиентов):

- I/O – сервер ввода/вывода. Обеспечивает передачу данных между физическими устройствами ввода/вывода и другими модулями Citect.
- Display – клиент визуализации. Обеспечивает операторский интерфейс: отображение данных, поступающих от других модулей Citect, и управление выполнением команд оператора.

- Alarms – сервер алармов (тревог). Отслеживает данные, сравнивает их с допустимыми пределами, проверяет выполнение заданных условий и отображает алармы на соответствующем узле визуализации.
- Trends – сервер трендов. Собирает и регистрирует трендовую информацию, позволяя отображать развитие процесса в реальном масштабе времени или в ретроспективе.
- Reports – сервер отчетов. Генерирует отчеты по истечении определенного времени, при возникновении определенного события или по запросу оператора.

В одной сети можно использовать только один сервер алармов, сервер трендов и сервер отчетов. В то же время допускается использование нескольких серверов ввода/вывода (I/O Server). Количество компьютеров с установленным модулем Display (обеспечивающим операторский интерфейс) в сети практически не ограничено.

SCADA-система может функционировать как открытая система. Распространение архитектуры «клиент-сервер» стало возможным благодаря развитию и широкому внедрению в практику концепции открытых систем, показанных на рис.4.3. Главной причиной появления и развития концепции открытых систем явились проблемы взаимодействия программно-аппаратных средств в локальных компьютерных сетях. Решить эти проблемы можно только путем международной стандартизации программных и аппаратных интерфейсов.

Концепция открытых систем предполагает свободное взаимодействие программных средств SCADA с программно-техническими средствами разных производителей. Это актуально, так как для современных систем автоматизации характерна высокая степень интеграции большого количества компонент. В системе автоматизации кроме объекта управления задействован целый комплекс программно-аппаратных средств: датчики и исполнительные устройства, контроллеры,

серверы баз данных, рабочие места операторов, АРМы специалистов и руководителей, как показано на рис.4.3. При этом в одной системе могут быть применены технические средства разных производителей.

Рис.4.3. Интеграция SCADA в систему управления

Очевидно, что для эффективного функционирования в этой разнородной среде SCADA-система должна обеспечивать высокий уровень сетевого взаимодействия.

Реализация этой задачи требует от SCADA-системы наличия типовых протоколов обмена с наиболее популярными промышленными сетями такими, как Profibus, ControlNet, Modbus и другими. С другой стороны, SCADA-системы должны поддерживать интерфейс и со стандартными информационными сетями (Ethernet и др.) с использованием стандартных протоколов (TCP/IP и др.) для обмена данными с компонентами распределенной системы управления.

Практически любая SCADA-система имеет в своем составе базу данных реального времени и подсистему архивирования данных. Но подсистема архивирования не предназначена для длительного хранения больших массивов информации (месяцы и годы). Информация в ней периодически обновляется, иначе для нее просто не хватит места. Рассматриваемый класс программного обеспечения (SCADA-системы) предназначен для обеспечения текущей и архивной информацией

оперативного персонала, ответственного за непосредственное управление технологическим процессом.

Информация, отражающая хозяйственную деятельность предприятия (данные для составления материальных балансов установок, производств, предприятия в целом и т. п.), хранится в реляционных базах данных (РБД) типа Oracle, Sybase и т. д. В эти базы данных информация поставляется либо с помощью ручного ввода, либо автоматизированным способом (посредством SCADA-систем). Таким образом, выдвигается еще одно требование к программному обеспечению SCADA – наличие в их составе протоколов обмена с типовыми базами данных.

Наиболее широко применимы два механизма обмена:

• ODBC (Open Data Base Connectivity – взаимодействие с открытыми базами данных) – международный стандарт, предполагающий обмен информацией с РБД посредством ODBC-драйверов. Как стандартный протокол компании Microsoft, ODBC поддерживается и наиболее распространенными приложениями Windows;

• SQL (Structured Query Language) – язык структурированных запросов.

Программное обеспечение SCADA должно взаимодействовать с контроллерами для обеспечения человеко-машинного интерфейса с системой управления, как показано на рис.4.3. К контроллерам через модули ввода/вывода подключены датчики технологических параметров и исполнительные устройства (на рис.4.3 не показаны).

Информация с датчика записывается в регистр контроллера. Для ее передачи в базу данных SCADA-сервера необходима специальная программа, называемая драйвером. Драйвер, установленный на сервере, обеспечивает обмен данными с контроллером по некоторому физическому каналу. Но для реализации обмена необходим и логический протокол.

После приема SCADA-сервером сигнал попадает в базу данных, где производится его обработка и хранение. Для отображения значения сигнала на мониторе рабочей станции оператора информация с сервера должна быть передана по сети клиентскому компьютеру. И только после этого оператор получит информацию, отображенную изменением значения, цвета, размера, положения и т. п. соответствующего объекта операторского интерфейса.

Большое количество контроллеров с разными программно-аппаратными платформами и постоянное увеличение их числа заставляло разработчиков включать в состав SCADA-системы большое количество готовых драйверов (до нескольких сотен) и инструментарий для разработки собственных драйверов к новым или нестандартным устройствам нижнего уровня.

Организация доступа к SCADA-приложениям является важным организационным и техническим вопросом. SCADA-приложения, по определению, являются потребителями технологических данных, но, с другой стороны, они должны быть и их источником. Информация со SCADA-приложений потребляется многочисленными клиентами (прежде всего, специалистами и руководителями среднего звена).

Для автоматизированного доступа к информации реального времени с любого рабочего места необходимо установить компьютер, подключенный к локальной сети. Организованное таким образом автоматизированное рабочее место (АРМ) предназначено для реализации вполне определенных функций. Поэтому программное обеспечение компьютера (системное и прикладное) должно обеспечить соответствующий данному АРМ набор пользовательских услуг. К их числу можно отнести:

• объем предоставляемой информации;

• форма представления информации;

• реализуемые функции (только информационные или с возможностью выдачи управляющих воздействий);

• протяженность и надежность канала связи «источник-потребитель»;

• простота освоения пользователем и т.д.

В периодической прессе последних лет за системным и прикладным программным обеспечением, которое необходимо компьютеру АРМ для получения удаленного доступа к производственной информации, закрепился термин «клиентское приложение». Клиентские приложения различного типа могут предоставлять информацию в любом объеме и приемлемом для пользователя виде.

Клиент-серверная организация SCADA-систем предполагает применение клиентских приложений двух типов: с возможностью передачи управляющих воздействий с клиентского приложения и чисто мониторинговые приложения. Пользователю необходимо лишь определить достаточный набор услуг. Поэтому весьма существенным критерием при организации клиентского узла (АРМ) является его стоимость (аппаратное и программное обеспечение).

В настоящее время существует несколько решений поставленной задачи, базирующихся на применении различных технологий. Но и стоимость предлагаемых решений тоже различна. Отсюда и появились такие понятия, как «бедные/богатые и тонкие/толстые клиенты».

5. Среда разработки программного обеспечения PC WORX 6

5.1. Учебный стенд ILC 130 STARTERKIT

Среда разработки программного обеспечения ***PC WORX 6*** фирмы Phoenix Contact имеет развитую систему документации и специальный учебный стенд ***ILC 130 STARTERKIT***.

Лабораторные стенды ***ILC 130 STARTERKIT*** (рис.5.1) предназначены для освоения методики программирования промышленных контроллеров ***Phoenix Contact*** (Германия). Стенд содержит все необходимые элементы для отладки программы на реальном устройстве: контроллер, имитаторы дискретных и аналоговых сигналов. Лабораторные стенды ***ILC 130 STARTERKIT*** поставлены Волжскому политехническому институту (ВПИ) в рамках участия ВПИ в международной образовательной программе ***EduNet***.

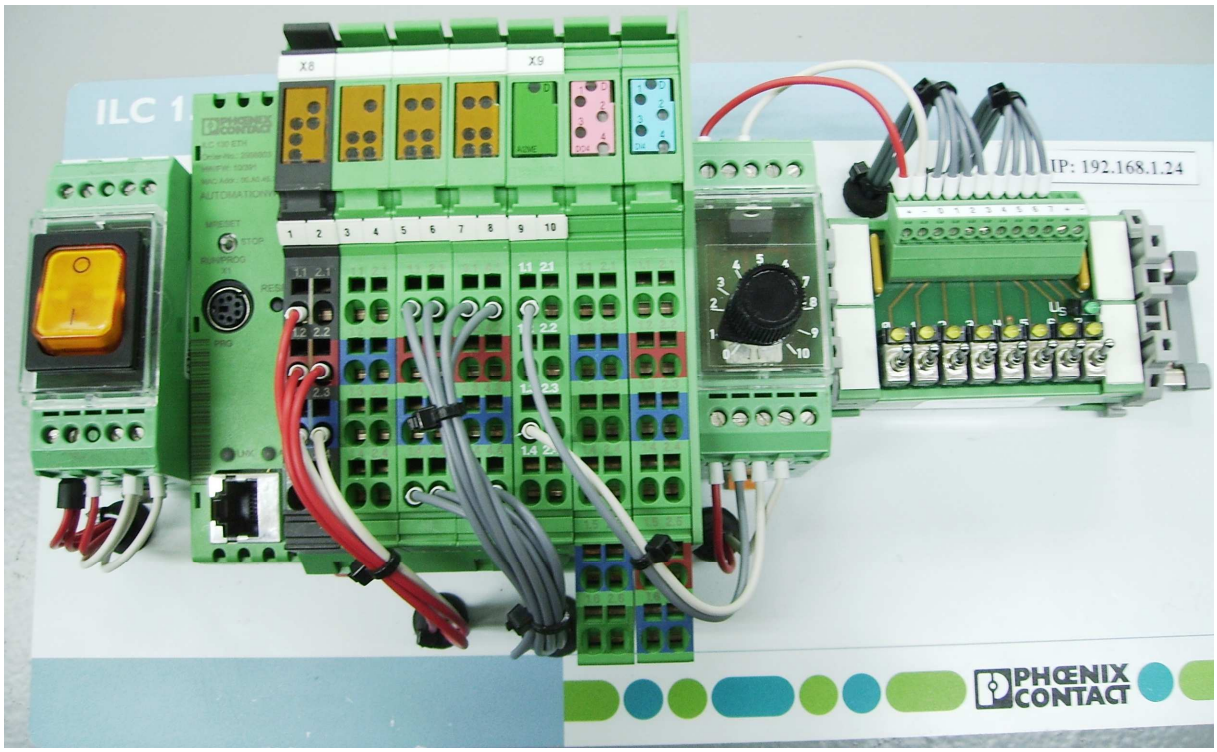


Рис.5.1. Учебный стенд ***ILC 130 STARTERKIT***

Схема соединений контроллера с имитаторами дискретных и аналоговых сигналов приведена на рис.5.2.

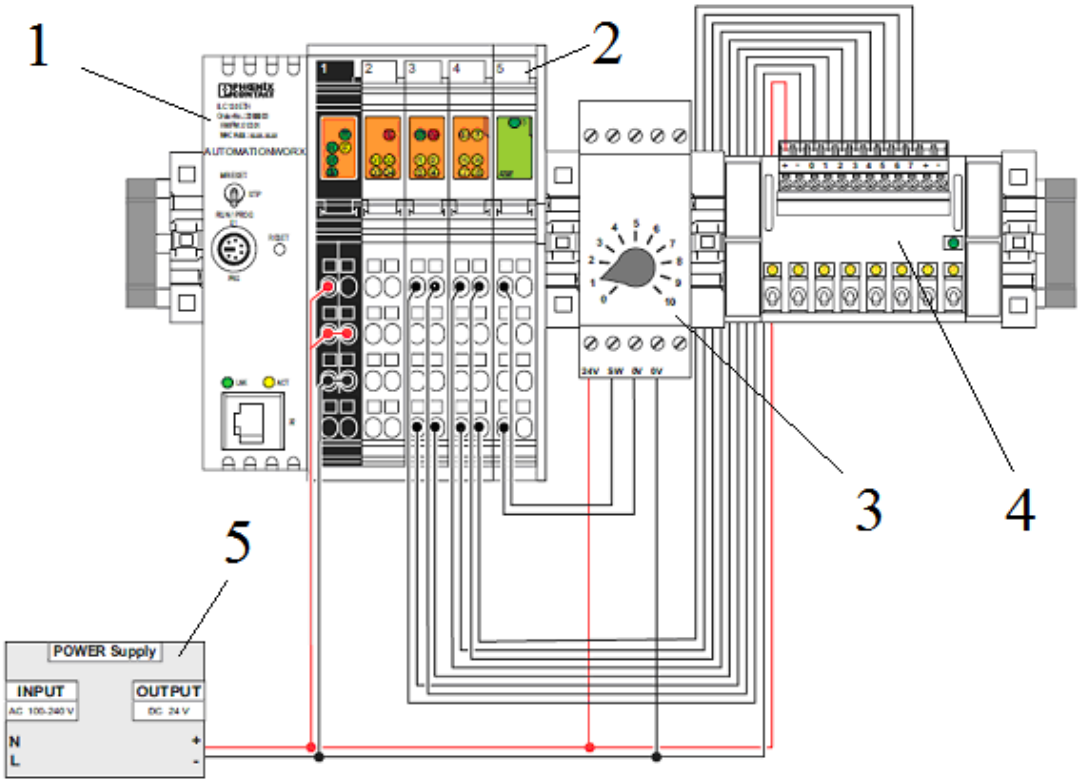


Рис.5.2. Схема соединений учебного стенда ILC 130 STARTERKIT

Состав стенда ***ILC 130 STARTERKIT***:

1. Программируемый логический контроллер ***ILC 130 ETH***;
2. Модуль ввода аналоговых сигналов на 2 канала ***IB IL AI2/SF-ME***;
3. Имитатор аналогового сигнала 0-10 В;
4. Имитатор дискретных сигналов на 8 каналов 24 В;
5. Блок питания на 24 В постоянного тока (1А).

5.2. Контроллер ILC 130 ETH

Контроллер ***ILC 130*** является младшей моделью 100 класса контроллеров от ***Phoenix Contact*** и предназначен для решения малых и средних задач автоматизации. Поддерживает расширение за счет дополнительных модулей и встроенной шины Inline, а также программирование и обмен данными по сети Ethernet.

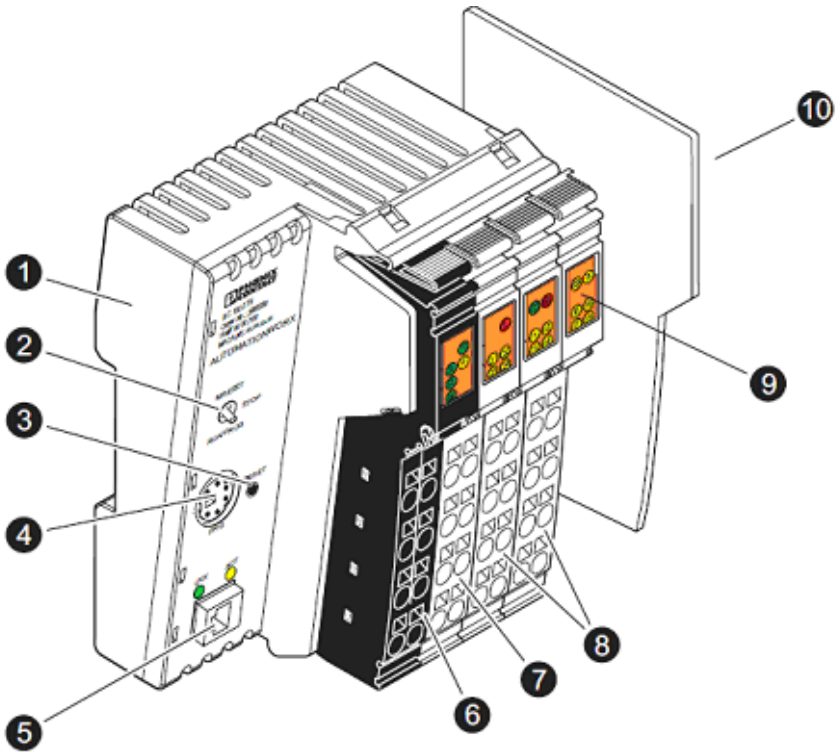


Рис.5.3. Внешний вид контроллера ***ILC 130 ETH***

Элементы контроллера ***ILC 130 ETH***:

1. Процессорный модуль;
2. Переключатель режима работы (старт/стоп/очистка памяти);
3. Кнопка сброса (утоплена);
4. Интерфейс RS-232 (используется разъем PS/2);
5. Ethernet порт;
6. Секция 1 (черная) : Клеммы для подключения питания;
7. Секция 2: Клеммы дискретных выходов;
8. Секции 3 и 4: Клеммы дискретных входов;
9. Индикаторы;
10. Боковая пластина.

Контроллер имеет модульную структуру. Справа к процессорному модулю могут подключаться необходимое количество модулей ввода/вывода в зависимости от требований решаемой задачи. В данном лабораторном стенде к ПЛК подключен только один модуль – модуль аналогового ввода (рис. 5.2, поз. 2).

Технические характеристики контроллера ILC 130 ETH:

Таблица 1

Название	Значение
Система программирования	PC WORX
Память программ	192 КБайт
Память данных	192 КБайт
Энергонезависимая память	8 Кбайт
Количество программируемых задач	8
Память параметрирования	4 Мбайт, встроенная (100 000 циклов перезаписи)
Встроенные дискретные входы	8 шт., допустимый уровень от 0.5 до 30 В (номин. 24 В), ток 7 мА (не более 15мА). Время переключения – 5 мс.
Встроенные дискретные выходы	4 шт., = 24 В (номинальный выходной ток 500 мА)
Интерфейсы обмена данными	Ethernet (для программирования или обмена данными), RS-232 (для диагностики или обмена данными с другими устройствами).
Встроенная шина	Inline, максимум 128 устройств.
Минимальное время цикла	1 мс
Часы реального времени, точность	1 мин в неделю
Степень защиты	IP20

Контроллер имеет встроенные дискретные входы (8 шт.) и дискретные выходы (4 шт.). Нумерация клемм контроллера показана на рис.5.4:

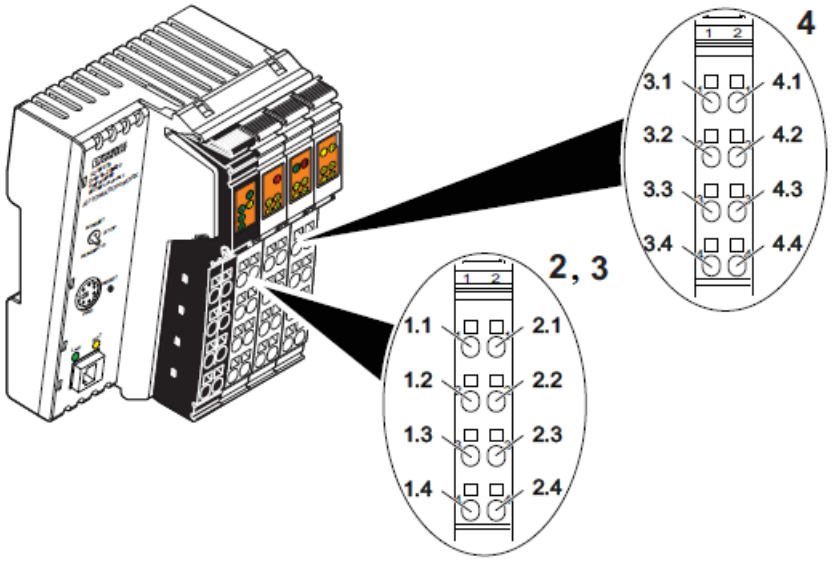
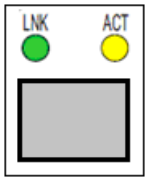
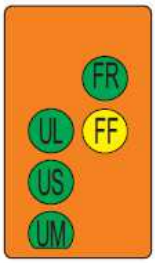


Рис.5.4. Нумерация клемм контроллера (дискретные входы и выходы)

Состояние контроллера отображается с помощью индикаторов, назначение которых указано в таблице 2.

Назначение индикаторов ПЛК:

Таблица 2

Ethernet разъем  (ETH)	LNK	Нет подключения/есть подключение
	ACT	Передача данных не ведется/ПЛК передает или принимает данные
Диагностика состояния контроллера и электропитания 	FR	ВЫКЛ – среда исполнения работает без ошибок; ВКЛ – среда исполнения не готова к работе (сбой микропрограммы); МИГАЕТ – среда исполнения инициализирована, но программа находится в состоянии READY/STOP.
	FF	ВЫКЛ – не обнаружено ошибки среды исполнения IEC 61131; ВКЛ – обнаружена ошибка среды исполнения IEC 61131.
	UL	Питание связи модулей и аналоговых цепей: ВКЛ – электропитание подключено/ ВЫКЛ – электропитание не подключено.
	US	Питание сегментной цепи внутренней шины: ВКЛ – электропитание подключено/ ВЫКЛ – электропитание не подключено.
	UM	Основная цепь питания всех устройств через внутреннюю шину: ВКЛ – электропитание подключено/ ВЫКЛ – электропитание не подключено.
Диагностика INTERBUS 	RDY	ВЫКЛ – Interbus мастер не готов к работе; МИГАЕТ - Interbus мастер в состоянии "готов" или "активен"; ВКЛ - Interbus мастер в состоянии "запущен".
	FAIL	ВЫКЛ – система работает без ошибок; ВКЛ – Обнаружена ошибка: ошибка на шине или ошибка контроллера.
	BSA	ВЫКЛ – все сегменты шины включены; ВКЛ – один или более сегментов шины отключен.
	PF	ВЫКЛ – все устройства работают нормально; ВКЛ – периферийная ошибка на локальной или удаленной шине.
Встроенные дискретные входы и выходы 	I1-I8	ВКЛ – вход активирован; ВЫКЛ – вход не активирован.
	E	ВЫКЛ – (перегрузка выходов/короткое замыкание) не обнаружено; ВКЛ – обнаружено.
	Q1-Q4	ВКЛ – выход активирован; ВЫКЛ – выход не активирован.

Лабораторный стенд имеет встроенные дискретные выходы, которые изначально ни к чему не подключены, однако, их срабатывание можно контролировать по индикаторам Q1 – Q4. К дискретным входам контроллера подключен имитатор дискретных сигналов (8 тумблеров). Их срабатывание также можно контролировать по индикаторам I1-I8.

Модуль IB IL AI2/SF-ME предназначен для ввода в контроллер аналоговых сигналов тока или напряжения и спроектирован для подключения к локальной шине Inline. Основные характеристики модуля приведены в таблице 3, назначение клемм указано в таблице 4.

Технические характеристики модуля:

Таблица 3

Название	Значение
Тип токового сигнала	0..10 мА, -20..+20 мА, 4-20 мА
Тип сигнала по напряжению	0..10 В, -10..+10 В
Способ подключения датчиков	двух- или трехпроводный
Конфигурация каналов	Каждый канал конфигурируется программно, независимо через шину
Форматы представления измеренных данных	4 формата
Разрешение АЦП	12 бит
Время обновления данных (максимальное)	1.5 мс

Назначение клемм модуля представлено в таблице 4:

Таблица 4

Номера клемм	Сигнал	Назначение
1.1	+U1	Положит. вход по напряжению канала 1
2.1	+U2	Положит. вход по напряжению канала 2
1.2	+I1	Положит. вход по току канала 1
2.2	+I2	Положит. вход по току канала 2
1.3	- (U1 или I1)	Минусовый вход канала 1
2.3	- (U2 или I2)	Минусовый вход канала 2
1.4, 2.4	Экран	Подключение экрана

По умолчанию значение передается в формате IB IL, что означает двухбайтовый регистр:

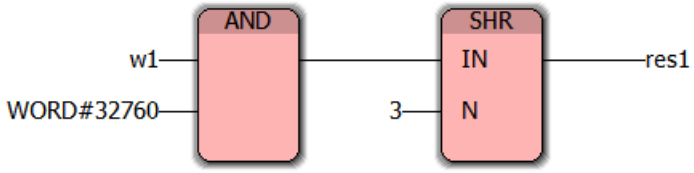
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SB	AV												x	x	x

где SB – бит знака, AV – аналоговое значение, X – незначащий бит (может быть 0 или 1).

Поэтому для извлечения из этого регистра измеренного аналогового значения:

- содержимое регистра нужно побитно умножить на (32760 = 01111111 11111000), чтобы обнулить незначащие биты и знаковый бит;
- полученное число циклически сдвинуть вправо на 3 разряда.

Эти действия должен делать сам программист в своей программе. Пример извлечения измеренного значения на языке FBD:



Исходная переменная w1 (WORD), результат помещается в res1 (WORD). Рекомендуется сделать собственный функциональный блок (ФБ) и далее использовать его везде, где необходимо получить величины, измеряемые блоком аналоговых входных сигналов.

5.3. Создание программного проекта в среде PC WORX 6

PC Worx представляет собой универсальное программное обеспечение для программирования всех классов контроллеров Phoenix Contact. Оно объединяет в себе языки программирования стандарта МЭК 61131-3, конфигурирование полевых шин и диагностику оборудования.

1) Создание проекта начинается с выбора пункта меню: **File->New Project.** Далее откроется окно выбора модели контроллера, название типа которого необходимо указать в этом окне.

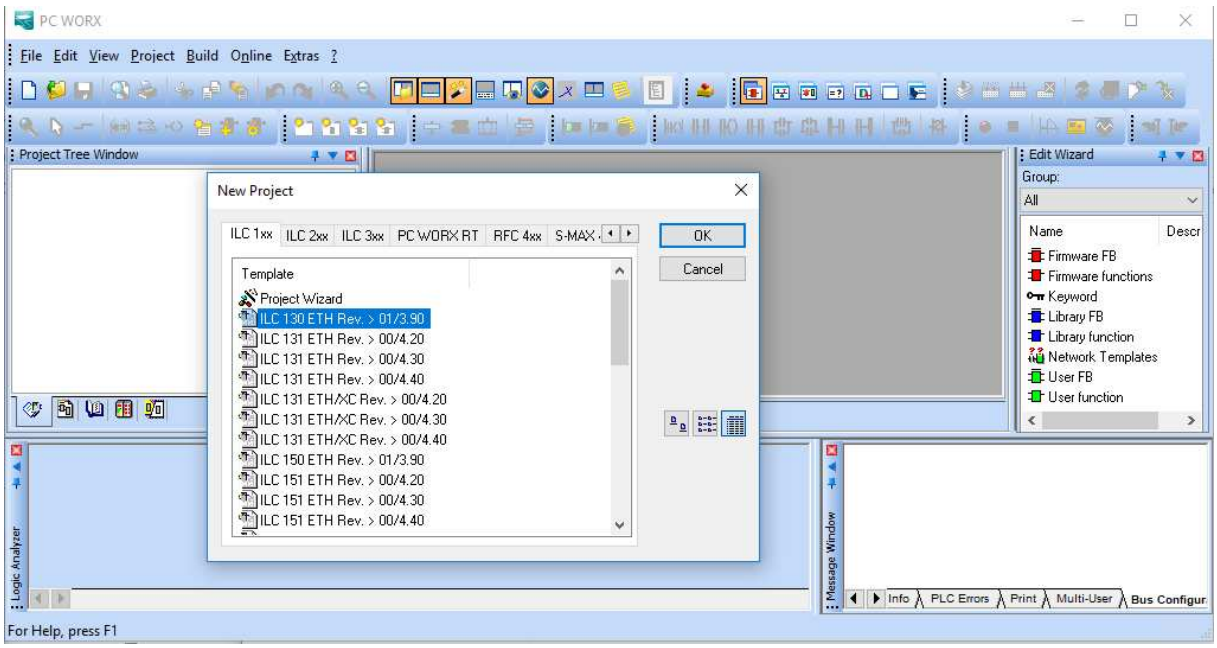


Рис.5.5. Окно выбора модели контроллера Phoenix Contact

Вкладки сверху обозначают различные классы контроллеров (ILC1xx, ILC2xx и т.д.), а их содержимое – конкретные модели внутри класса. Для учебного следует выбрать контроллер **ILC 130 ETH Rev. > 01/3.90.**

2) После того как проект создан, необходимо установить связь с ПЛК, для этого необходимо соединить оба устройства (ПК и ПЛК) кабелем Ethernet:

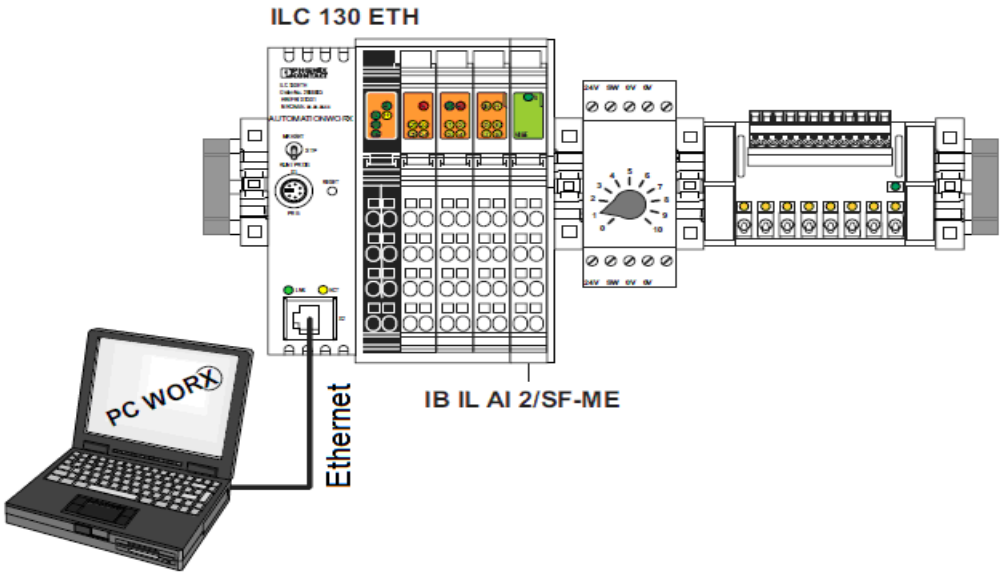


Рис.5.6. – Соединение ПК и ПЛК кабелем Ethernet

Необходимо убедиться, что IP-адрес компьютера находится в одной подсети с IP-адресом контроллера. Это означает, что IP адреса должны (при маске подсети 255.255.255.0) отличаться только последней цифрой – номером хоста, например:

Компьютер: 192.168.3.1, контроллер: 192.168.3.10.

Проверить и изменить IP-адрес компьютера можно в сетевых настройках. (В Windows 7: Панель управления->Сеть и интернет->Сетевые подключения->Свойства Подключения по локальной сети->Протокол интернета версии 4 (TCP/IP v.4)->Свойства).

Если контроллер программируется впервые, то у него отсутствует IP-адрес. В этом случае используется только первый метод назначения адреса – через **BootP** протокол. Если контроллер уже имеет свой IP-адрес, его необходимо указать в проекте PC Worx и проверить наличие связи. Для этого переключиться в режим конфигурации шины **Bus Configuration Workspace**  . Выделить строку с именем контроллера **ILC 130 ETH 192.168.0.2** слева в дереве объектов (**Bus Structure**). Перейти на вкладку **Extended Settings** на правой панели **Device Details** свойств контроллера и в строке **IP Address** ввести адрес контроллера. Маску подсети оставить равной **255.255.255.0**.

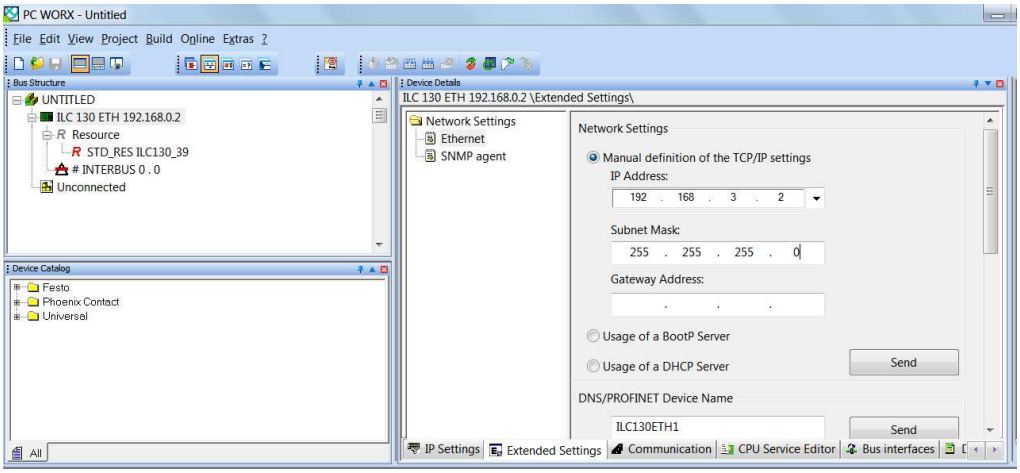


Рис. 5.7. Проверка правильности IP-адреса контроллера и маски подсети

Далее перейти на вкладку **Communication** и проверить правильность IP адреса и маски подсети (рис.5.8). Нажать кнопку **Test** для проверки связи с контроллером. В случае успешного соединения полоска над кнопкой **Test** станет зеленой:

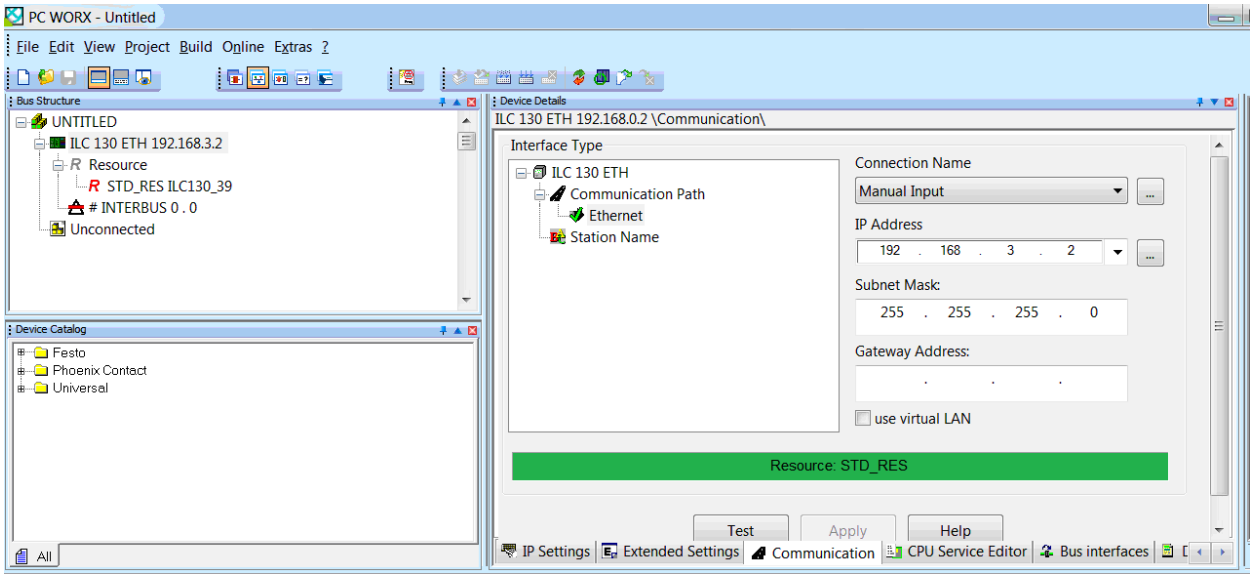


Рис.5.8. Проверка наличия связи с контроллером

После того, как связь с контроллером установлена, можно переходить к конфигурированию шины, то есть, подключению всех модулей ввода-вывода, установленных в системе.

Конфигурирование шины может осуществляться в ручном (offline) или автоматическом режиме (online). Offline конфигурирование может использоваться в тех случаях, когда отсутствует физическое подключение к контроллеру.

Конфигурирование в ручном (offline) режиме:

- Переключиться в режим конфигурации шины Bus Configuration Workspace  .
- На панели Bus structure (слева вверху) выделить строку INTERBUS 0.0.

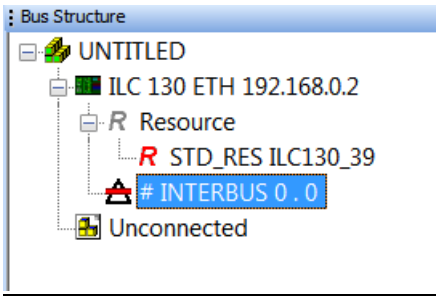


Рис.5.9. Панель Bus structure

- На панели Device Catalog найти нужный модуль (для этого использовать его полное название, указанное на коробке).

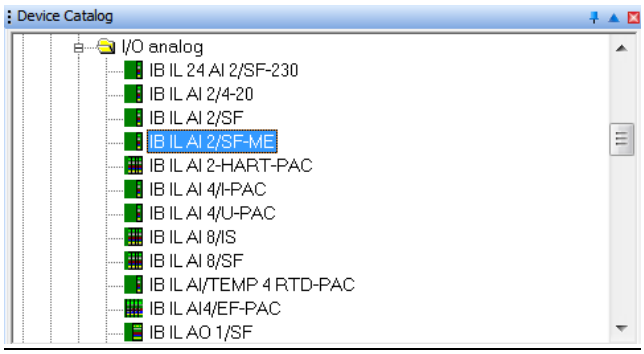


Рис.5.10. Панель Device Catalog

- Нужный нам модуль следует искать по адресу: Phoenix Contact -> IL -> I/O analog -> IB IL AI2/SF-ME.
- Двойным нажатием на имени модуля добавить его на шину контроллера. Имя модуля должно появиться в ветке INTERBUS 0.0. Удаление модуля возможно через его контекстное меню.

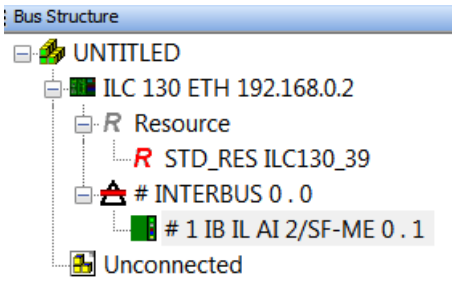


Рис.5.11. Панель Bus structure с подключенным модулем аналогового входа

- Аналогичным образом добавить остальные модули.

В приведенных выше рисунках показано подключение модуля ввода аналоговых сигналов в режиме offline. При наличии в лабораторном стенде дополнительных модулей (дискретного ввода и дискретного вывода)

выполнить также их подключение. Следует использовать следующие пути при поиске по каталогу устройств:

Phoenix Contact -> IL -> I/O digital -> IB IL 24 DO 4

Phoenix Contact -> IL -> I/O digital -> IB IL 24 DI 4-PAC

В результате дерево устройств должно выглядеть так, как показано на рис.5.12.

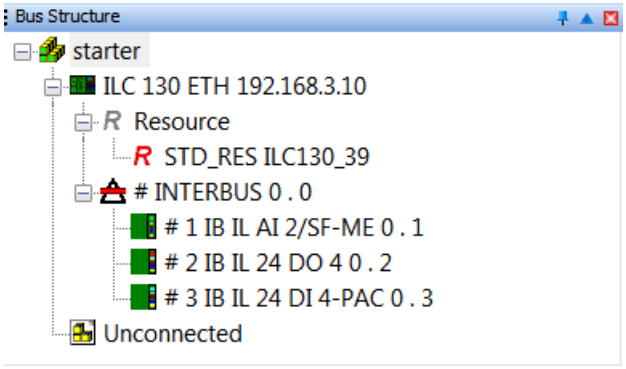


Рис.5.12. Панель Bus structure с подключенными модулями

5.4. Создание и загрузка программы в контроллер средствами среды PC WORX 6

Рассмотрим создание простой программы с использованием языка функциональных блоков FBD. Программа будет выполнять функцию логического "И" с сигналами первых двух дискретных входов ПЛК. Результат логической операции будет управлять первым дискретным выходом ПЛК.

Создание программы осуществляется в режиме IEC Programming Workspace, для активации которого нужно выбрать соответствующую кнопку в строке меню  . В окне дерева проекта слева (Project Tree Window) существует главная программа проекта – Main. Она состоит из трех файлов: MainT – файл комментариев; MainV – таблица переменных; Main – непосредственно программа, рис.5.13.

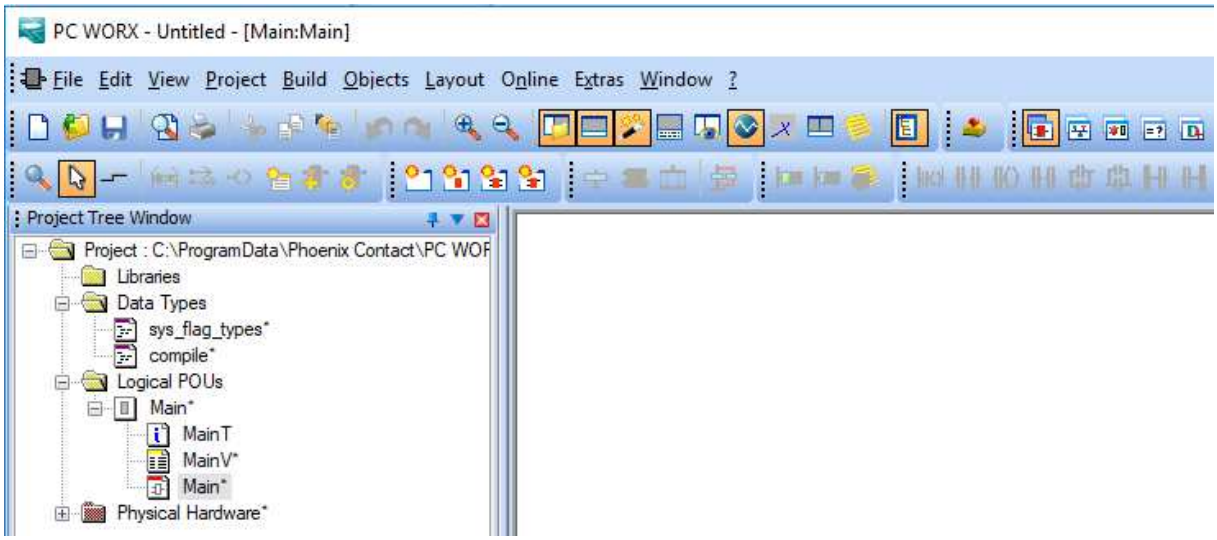


Рис.5.13. Дерево проекта и окно (пустое) программы Main

Примечание: по умолчанию для программы Main выбирается язык FBD. Если требуется изменить язык для программы Main, то следует удалить её (нажать правой кнопкой на строке Main и выбрать Delete), а потом создать новую программу (правой кнопкой на Logical POUs->Insert->Program, выбрать язык и имя программы).

Нажать двойным щелчком на файле программы Main (нижняя строка). Откроется рабочее поле программы. Установить крестообразный курсор на произвольном месте рабочего поля. Вызвать функцию логического "И". Это можно сделать двумя способами:

- начать набирать на клавиатуре слово AND, после чего функция будет автоматически предложена программисту;
- использовать помощник (Edit Wizard), расположенный справа от рабочего поля программы. Найти там нужную функцию и перетащить ее мышью на рабочее поле программы.

Результат ввода в программу Main функционального блока логического AND ("И") показан на рис.5.14.

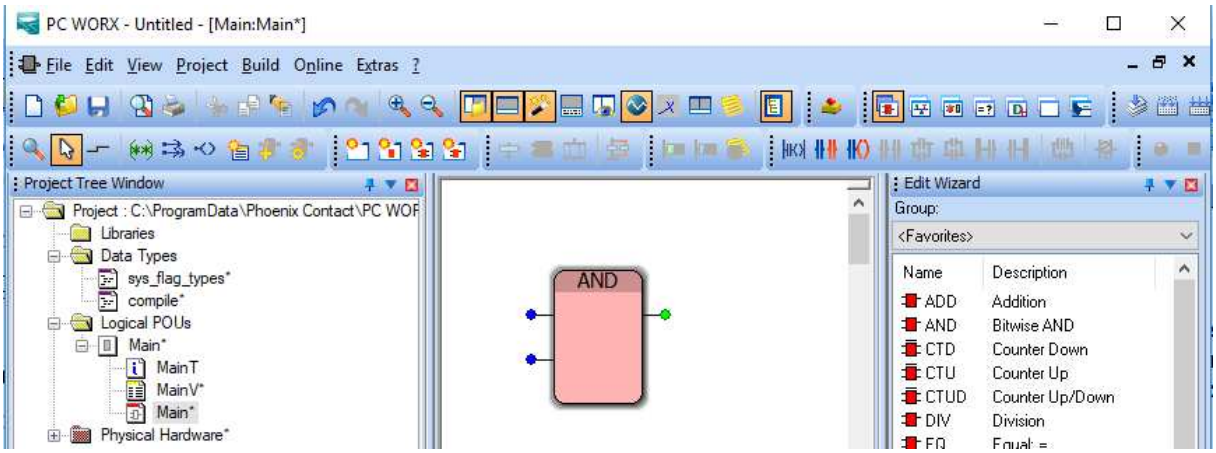


Рис.5.14. Программа Main с функциональным блоком логического AND

Привязка дискретных входов и выходов ПЛК к соответствующим ножкам блока AND выплняется двойным щелчком на первом входе блока AND, чтобы появилось меню Varable Properties. Выставить галочку Global, указать группу System Variables, и среди системных переменных выбрать ONBOARD_INPUT_BIT0, как показано на рис.5.15.:

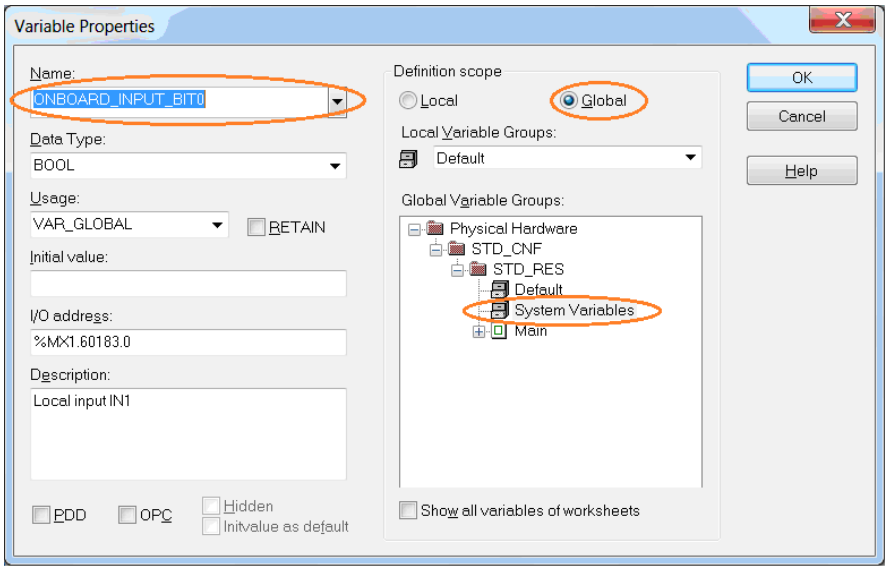


Рис.5.15. Создание глобальной переменной для первого дискретного входа

После нажатия кнопки ОК первый дискретный вход будет привязан к входу логического блока AND, как показано на рис.5.16.

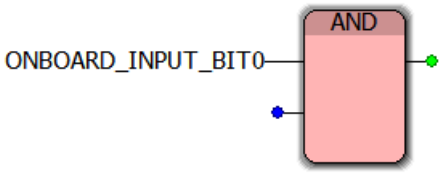


Рис.5.16. Привязка первого дискретного входа к входу блока AND

Таким же образом следует привязать физический второй вход и первый выход контроллера:

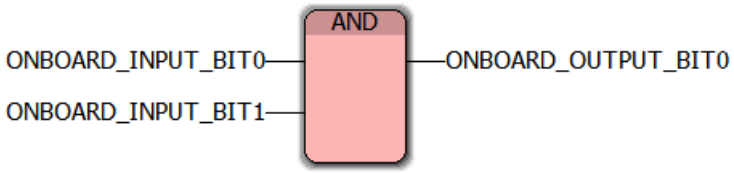


Рис.5.16. Входы и выход блока AND привязаны к физическим линиям контроллера

Компиляция программы осуществляется нажатием Build->Compile Worksheet. Сборка проекта осуществляется нажатим Build->Rebuild Project. При отсутствии ошибок (вкладка Errors внизу) проект может быть загружен в контроллер. Для этого выбрать кнопку Project Control Dialog . Откроется панель управления состоянием контроллера, рис.5.17.

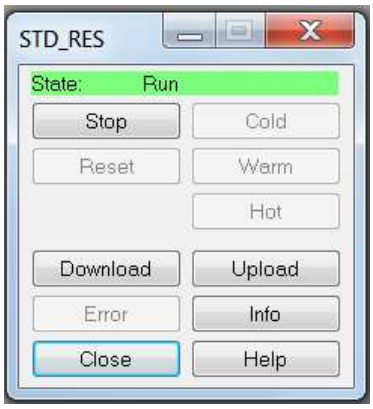


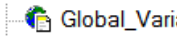
Рис.5.17. Меню управления состоянием контроллера

Далее на панели нажать кнопку Stop – остановка выполнения текущей программы ПЛК. И нажать кнопку Download. Если в программе нет ошибок – она загрузится в контроллер. ПЛК перейдет в режим ожидания.

Запустить программу можно кнопкой Cold – рестарт программы с очисткой и инициализацией всех переменных.

Работоспособность всех приведенных в этом разделе операций по созданию программы можно на учебном стенде.

5.5. Контроль состояния переменных программы средствами среды PC WORX 6

В процессе выполнения программы в PC WORX имеется возможность просмотра и изменения величин переменных в реальном времени. Для этого нажать на иконке Debug ON/OFF  . При этом рядом с входами и выходами ФБ будут отображаться их значения. Также значения переменных можно наблюдать в таблице переменных функциональных блоков, что показано на рис.5.18 (для глобальных системных переменных ПЛК 

).

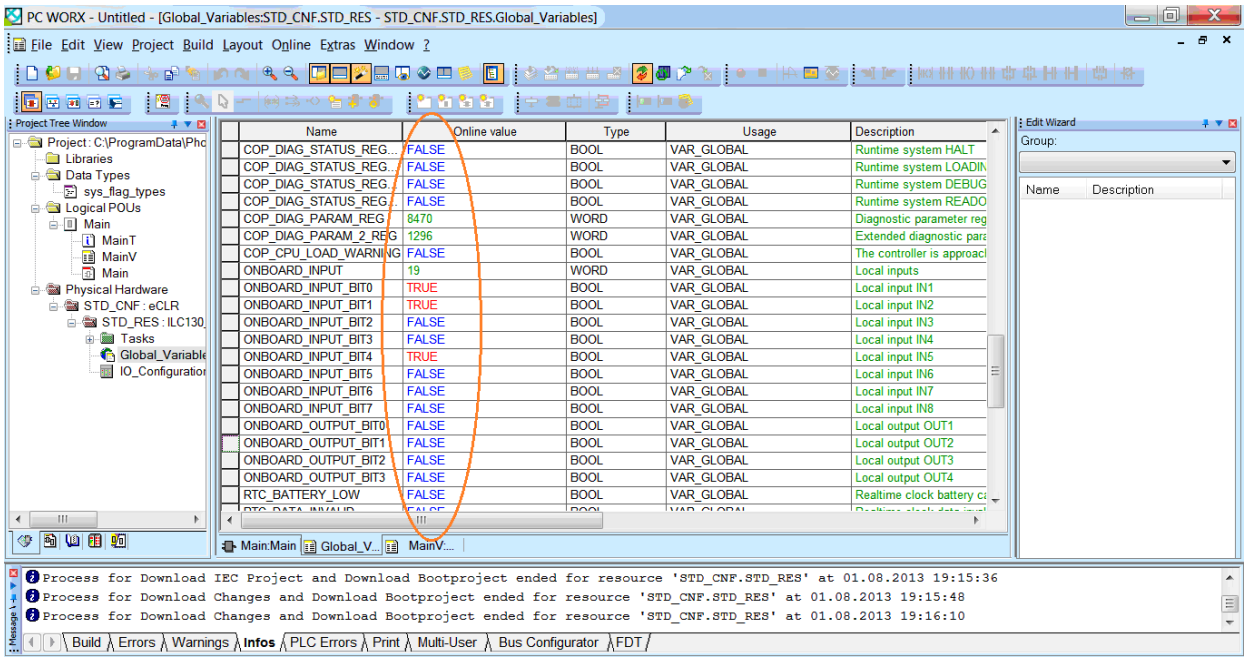


Рис.5.18. Отслеживание состояния переменных в реальном времени

Приведенная выше программа использует встроенные системные переменные контроллера для управления логическими входами и выходами. Если кроме ПЛК установлены дополнительные модули (на учебном стенде это модуль аналогового ввода IB IL AI2/SF-ME), то для использования данных, полученных с модуля ввода, в своей программе необходимо создать переменную, которую следует привязать к соответствующему физическому входу или выходу модуля. Данная операция выполняется в режиме Process Data Workspace  .

После перехода в этот режим откроется окно, поделенное на 4 функциональные зоны, которые показаны на рис.5.19.

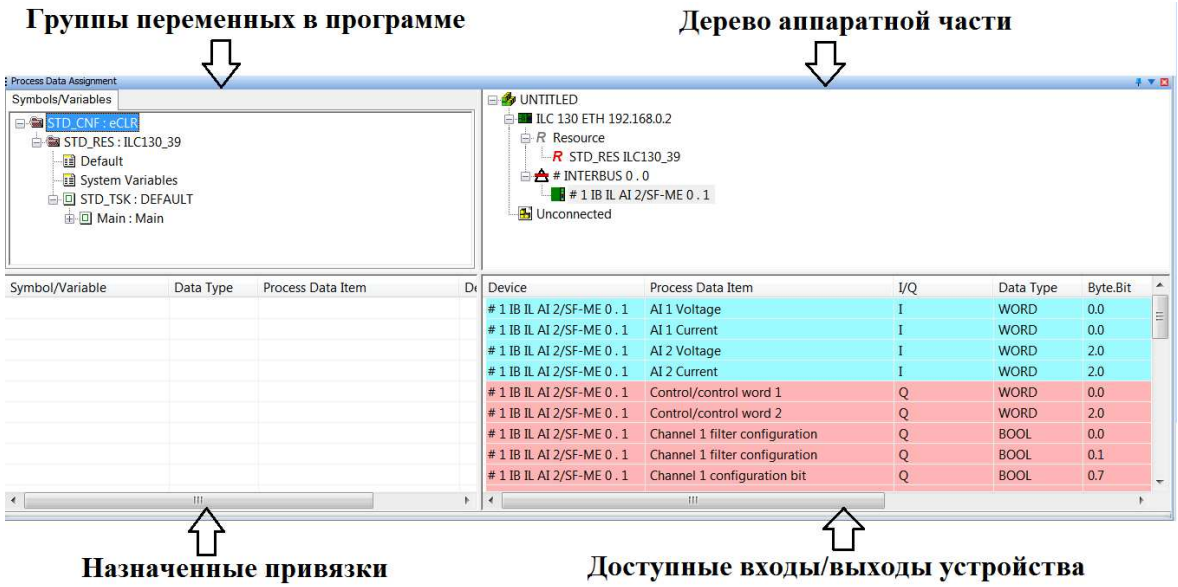


Рис.5.19. Окно режима Process Data Workspace

Создадим переменную для первого аналогового входа 0-10В модуля аналогового ввода IB IL AI2/SF-ME. На панели групп переменных (левой верхней) выберем строку STD_RES:ILC130_39. На панели дерева аппаратной части (справа вверху) выберем строку IB IL AI2/SF-ME. При этом на нижней правой панели отобразятся все доступные входы устройства, а также его дополнительные настроечные регистры. Далее выбрать строку AI 1 Voltage и мышью перетащить её на левую нижнюю

панель привязок на первую позицию. Автоматически создастся переменная, привязанная к первому аналоговому входу, как показано на рис.5.20.

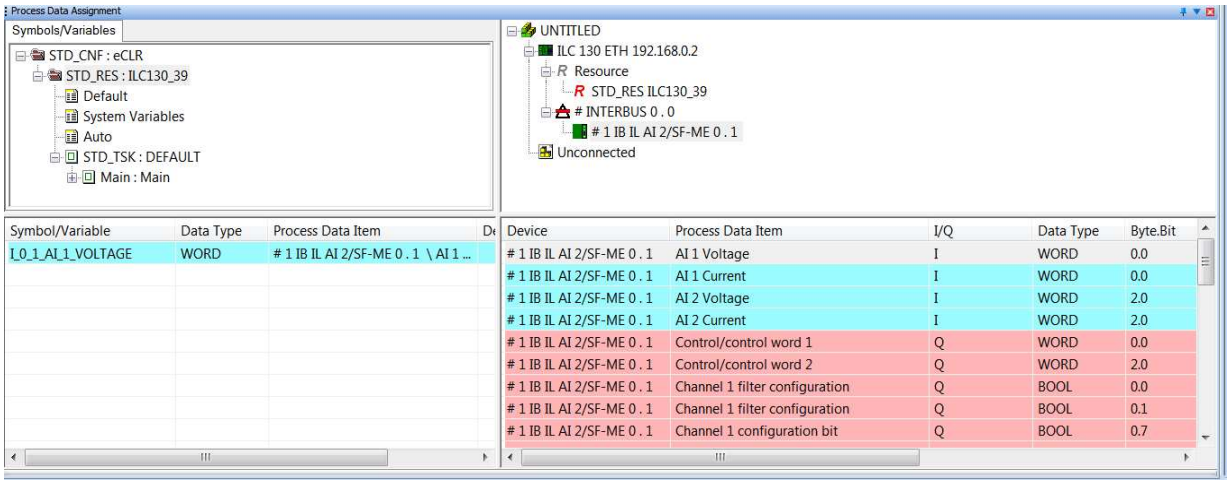


Рис.5.20. Автоматическое создание переменной для входа аналогового модуля

PC Worx автоматически присвоил имя этой переменной: ***I_0_1_AI_1_VOLTAGE***. Тип переменной определяется конкретным модулем и в данном случае является положительным целочисленным двухбайтовым числом – WORD (0..65536). После переключения в режим программирования IEC Programming Workspace  , созданную переменную можно обнаружить в Global_Variables в группе Auto, как показано на рис.5.21.

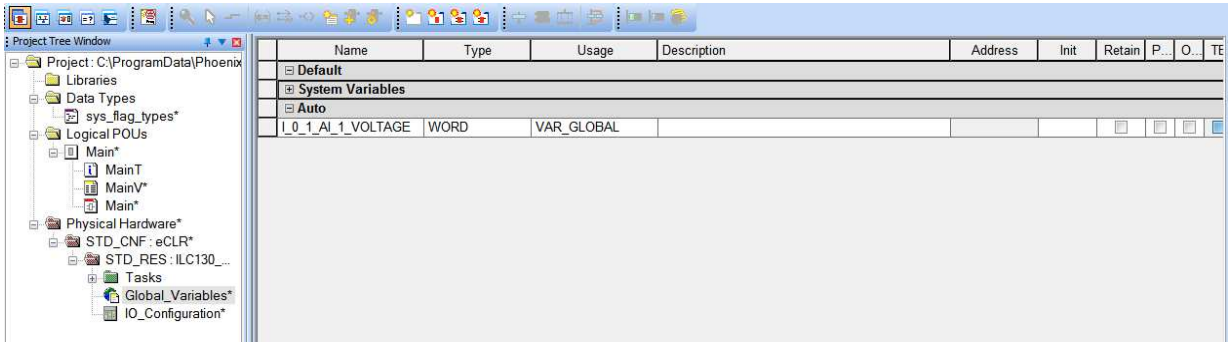


Рис.5.21. Таблица глобальных переменных

Полученную новую переменную для сигнала первого аналогового входа можно использовать в программе.

В соответствии с документацией на модуль ввода измеряемое значение AV занимает 12 бит в 16 битовой переменной:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SB	AV												x	x	x

Следующая FBD программа на рис.5.22 позволяет преобразовать получаемое число к диапазону 0..100.

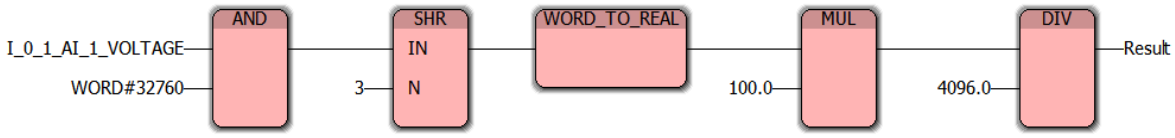


Рис.5.22. Процедура нормирования входного аналогового сигнала

Сначала происходит побитовое умножение (AND) на 32760 и сдвиг содержимого переменной I_0_1_AI_1_VOLTAGE на 3 разряда вправо (SHR) (см. п. 1.3). Затем идет преобразованием типов (WORD_TO_REAL) для обеспечения точности. Потом полученное число умножается на коэффициент 100/4096. Результат записывается в переменную Result (тип REAL).

Стандартные модули языка FBD копируются из окна Edit Wizard. Полный набор блоков доступен в группе <all FUs and FBs>, как показано на рис.5.23.

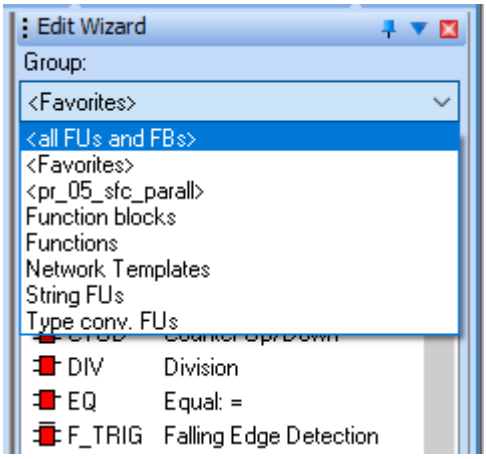


Рис.5.23. Группы блоков языка FBD

Программа на рис.5.24 включает дискретный выход Q2, если значение переменной AnScale превысило 75. Если значение Result ниже 25, то включается выход Q4. Если значение лежит в интервале (25..75), то оба выхода выключены.

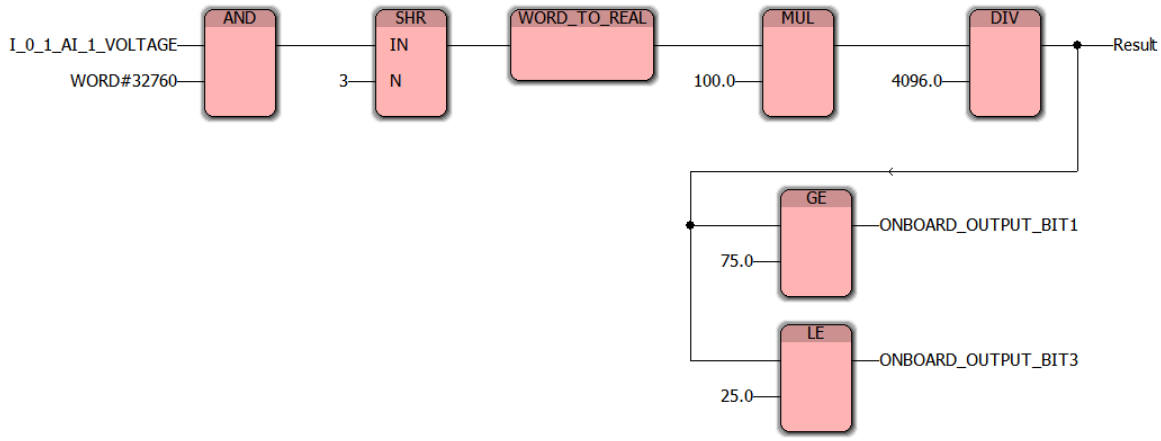


Рис.5.24. Пример программы на языке FBD для работы с аналоговым сигналом

6. Отладка программ средствами среды PC WORX6 в режиме симуляции

Программная среда PC Worx имеет возможность отладки программы контроллера в режиме эмуляции без наличия реального физического контроллера. Это удобно, когда программу требуется разработать и отладить заранее или проверка работы программы на реальном контроллере невозможна. Это также позволяет экономить ресурсы контроллера, так как число циклов перезаписи энергонезависимой памяти ПЛК ограничено.

Режим эмуляции возможен только в том случае, когда проект PC Worx создан для контроллера **4xx** класса, например **RFC 470S PN-3TX,** что показано на рис.6.1.

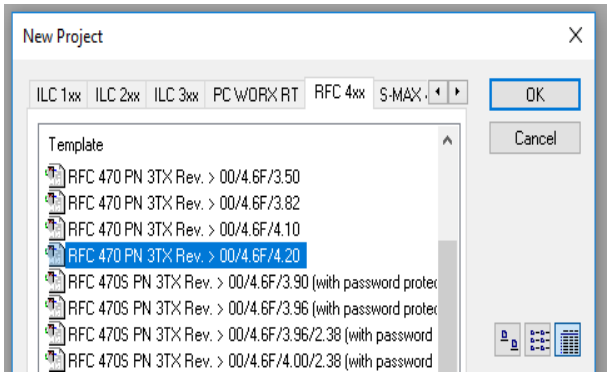


Рис.6.1. Выбор контроллера при создании проекта

Если в дальнейшем требуется перенести разработанные алгоритмы на контроллер другой модификации, например ILC 130 ETH, то возможен такой вариант:

- программа разрабатывается и отлаживается для контроллера ***RFC 470S PN-3TX*** в режиме эмуляции;
- все функциональные модули программы экспортируются в файлы, а затем импортируются в новый проект, созданный для контроллера ***ILC 130 ETH***.

Разработка проекта начинается с выбора контроллера (рис.6.1). После создания проекта требуется активировать режим **эмуляции**. Для этого:

- перейти в режим ***Bus Configuration Workspace***  ;

- в панели ***Bus Structure*** (Структура шины) выбрать контроллер ***RFC 470S PN-3TX***;

- на правой панели свойств контроллера во вкладке ***Communication*** выбрать в списке режим ***Simulation*** и нажать кнопку ***Apply*****,** как показано на рис.6.2. При переходе в режим симуляции появляется галочка в строке **Simulation**, а полоска ***Apply successful*** кратковременно становится зеленой.

При запуске программы в режиме симуляции загрузка будет осуществляться в виртуальный контроллер.

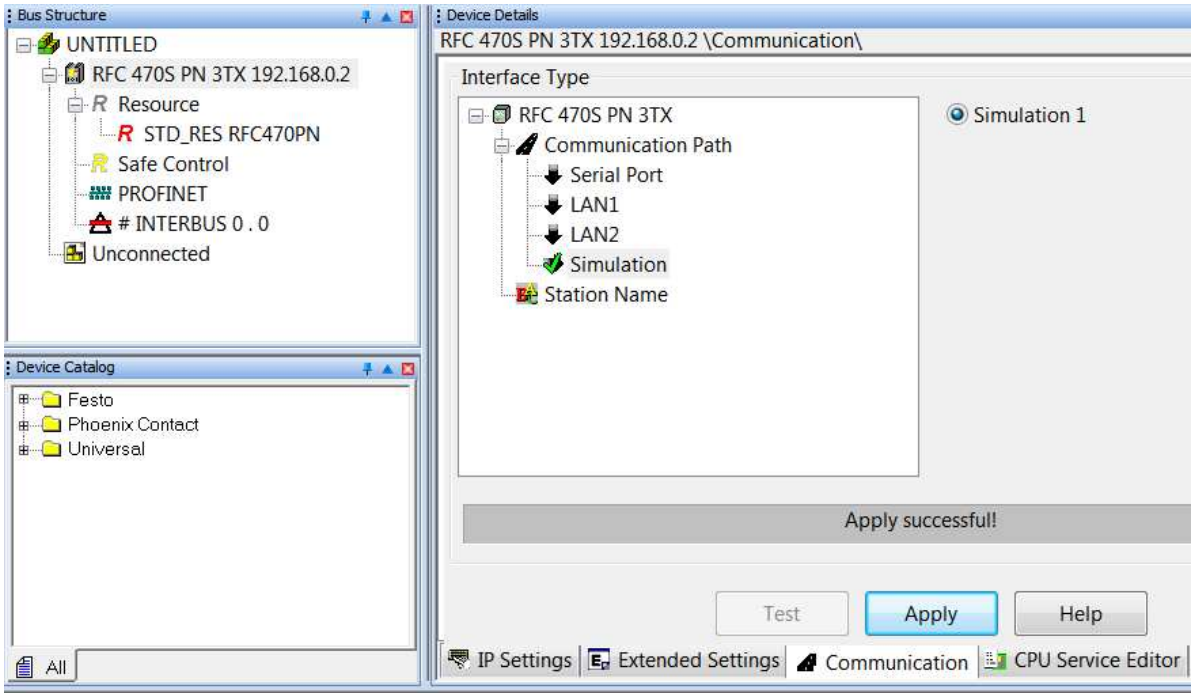


Рис.6.2. Включение режима эмуляции контроллера (Simulation)

Следующим этапом подготовки симуляции является этап создания программы, путем переключения в режим ***IEC Programming Workspace***  . Дальнейшее изложение ведется на примере создания простейшей программы генерации прямоугольных импульсов на основе двух таймеров задержки на языке FBD, показанной на рис.6.3.

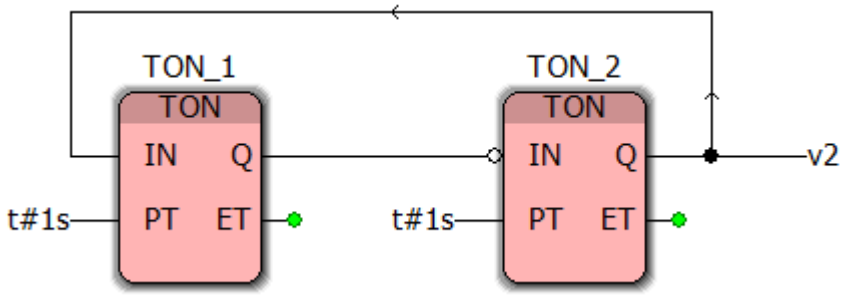


Рис.6.3. Пример программы для симуляции

Таймеры копируются из библиотеки ***Edit Wizard***. Процесс копирования сопровождается появлением окна ***Variable Properties*** рис.6.4, в которое вносить изменения нецелесообразно.

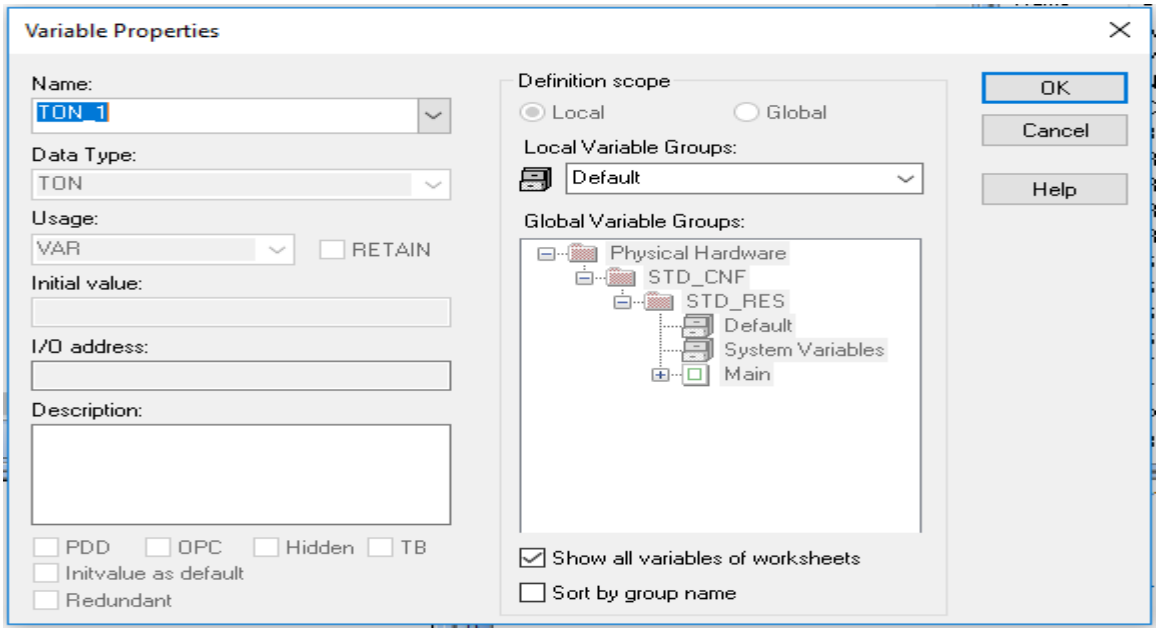


Рис.6.4. Окно свойств таймера

Второй таймер должен иметь инверсный вход, обозначаемый кружочком рис.6.5. Щелчок правой кнопкой мышки на входе таймера позволяет вызвать контекстное меню, показанное на рис.6.6, в котором следует выбрать строку ***Object Propertis***.

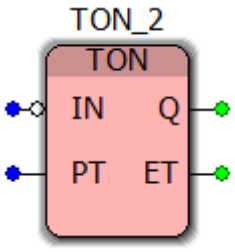


Рис.6.5. Инверсный вход таймера

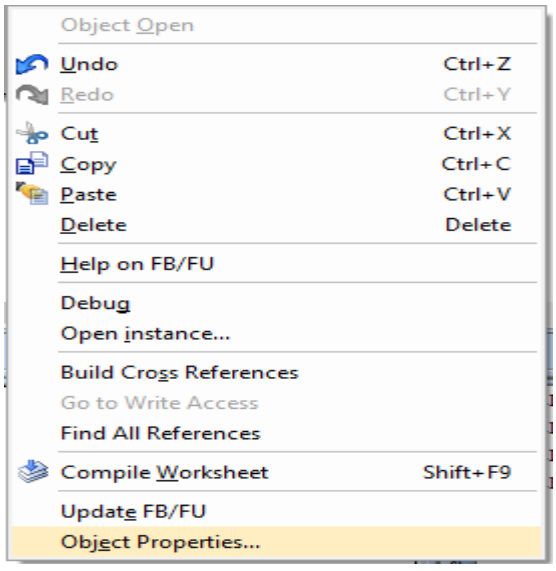


Рис.6.6. Контекстное меню входа таймера

В появившемся окне следует выбрать строку ***IN***, в которой следует установить галочку в столбике ***Negted***, как показано на рис.6.7.

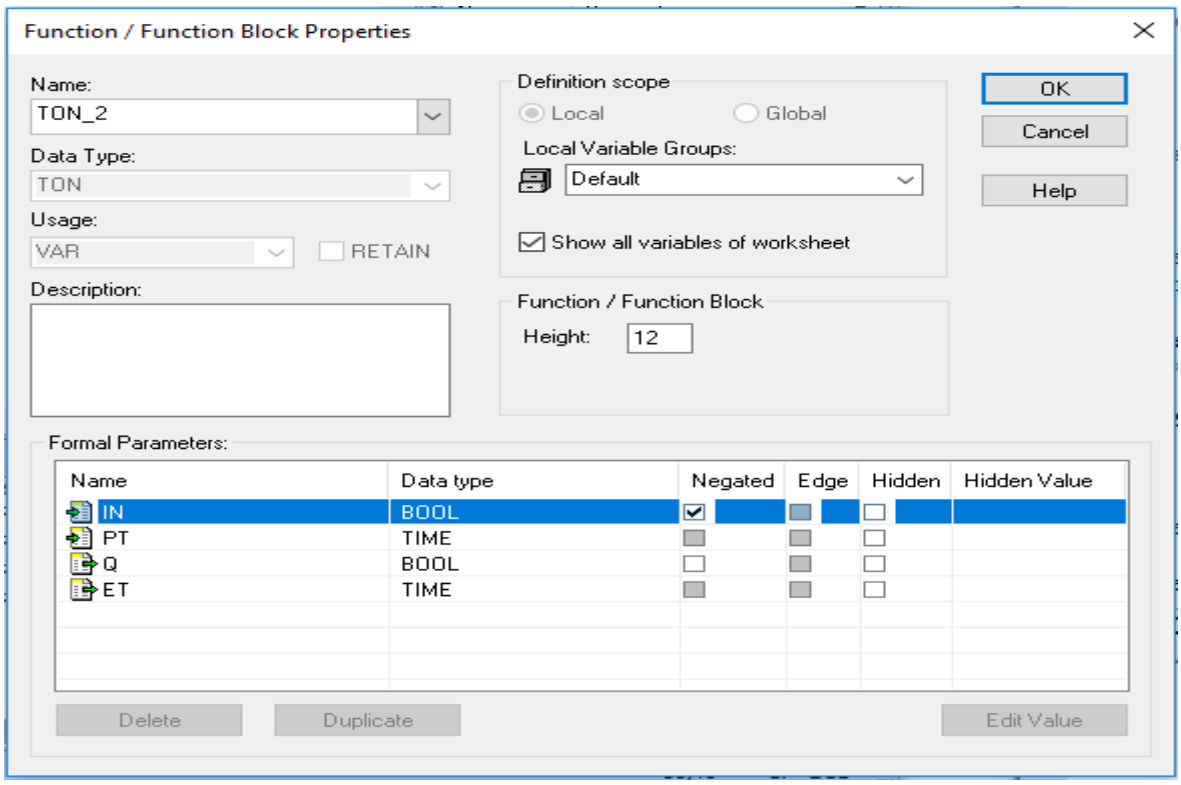


Рис.6.7. Окно свойств входов и выходов таймера

Указание величины задержки таймера на входе ***PT*** осуществляется в окне, которое вызывается двойным щелчком мышки на этом входе и показано на рис.6.8.

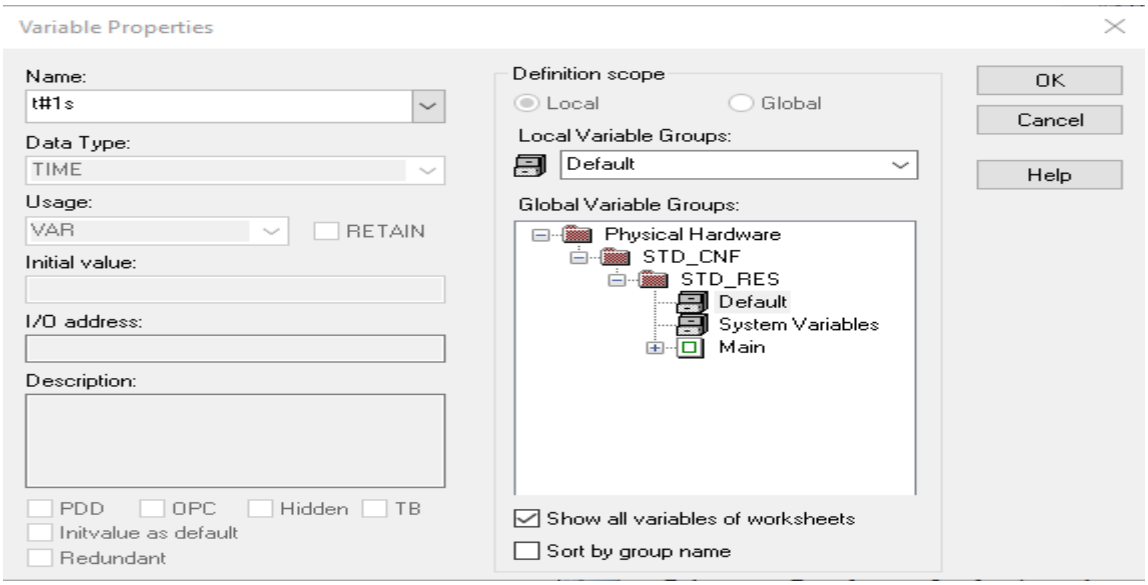


Рис.6.8. Окно для ввода задержки таймера

Нажатие кнопки ***OK*** приводит к появлению на входе ***PT*** таймера указанной величины задержки, как показано на рис.6.9.

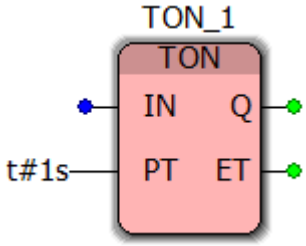


Рис.6.9. Таймер с указанной на входе величиной задержки

Переменные в программах на языке ***FBD*** формируются путем вызова контекстного меню, показанного на рис.6.10, щелчком правой кнопки мышки в свободном месте диаграммы.

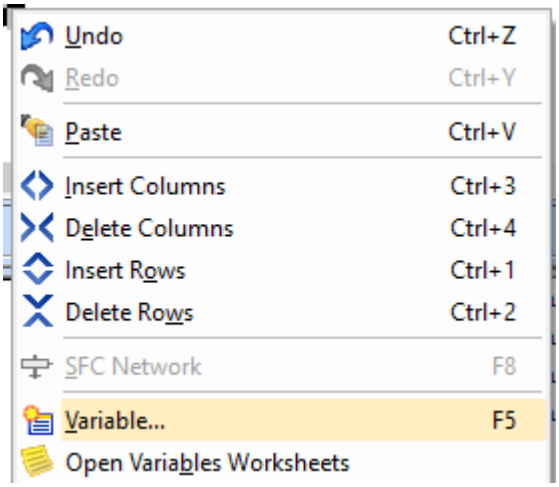


Рис.6.10. Меню формирования переменной

Нажатие кнопки ***Variable*** вызывает меню, показанное ранее на рис.6.8, в котором необходимо ввести имя новой переменной, а не величину задержки. Новая переменная ***v2*** показана на рис.6.11.

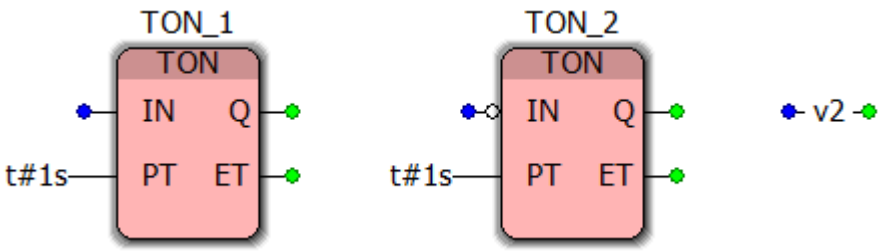


Рис.6.11. Переменная ***v2*** на диаграмме

Соединение элементов программы линиями позволяет получить диаграмму программы на языке ***FBD***, которая была приведена на рис.6.3. Результирующий список переменных программы открывается двойным щелчком мышки в окне ***ProjectTreeWindow*** и показан на рис.6.12.

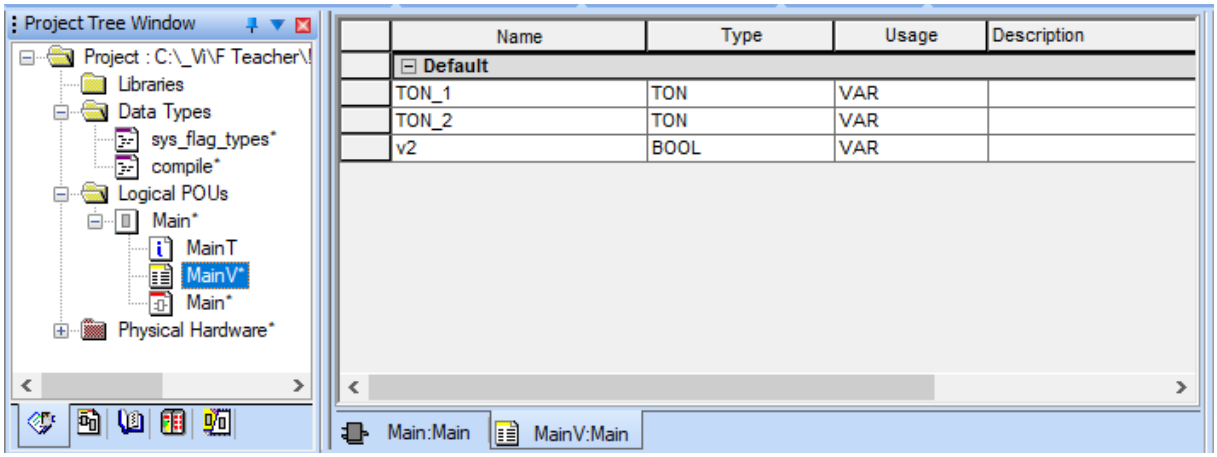


Рис.6.12. Список переменных программы

Примечание: если арифметическая операция выполняется с вещественными типами данных (REAL), то числовые константы записываются с десятичной точкой, например: ***3.0***

Выполнить компиляцию программы (***Build -> Compile Worksheet***  и затем ***Build -> Rebuild Project*** ), при этом в окне ***MessageWindow*** должно появиться сообщение об отсутствии ошибок в процессе компиляции:

«0 Error(s)».

В случае наличия ошибок необходимо проверить правильность программы и её конфигурации, в том числе активность режима симуляции.

Вызов диалога управления процессов проекта ***STD_RES***, показанного на рис.6.13, осуществляется нажатием кнопки ***Project Control Dialog***  . После нажатия кнопки ***Download*** в окне ***STD_RES*** откроется меню загрузки, показанное на рис.6.14, в котором нужно установить галочку в

позиции ***Ensure real-time*** и нажать левую верхнюю кнопку ***Download***. Процесс симуляции запускается кнопкой 

.

Рис.6.13. Окно управления

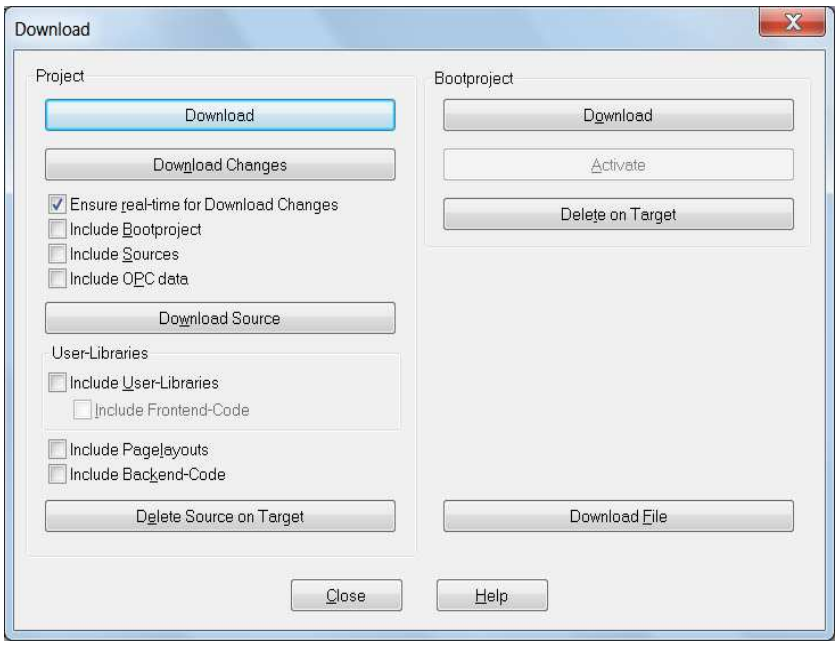


Рис.6.14. Окно загрузки проекта в контроллер

Назначение основных пунктов:

- ***Ensure real-time for Download Changes*** – проверка онлайн изменений в программе;
- ***Include Bootproject*** – загрузить программу в энергонезависимую память;
- ***Include Sources*** – загрузить в контроллер также файлы проекта;
- **Include OPC data** – загрузить в контроллер OPC данные.

После того, как программа будет загружена, нажать кнопку **Debug on/off**  . Рядом с переменными программы будут показываться их текущие значения, как показано на рис.6.15.

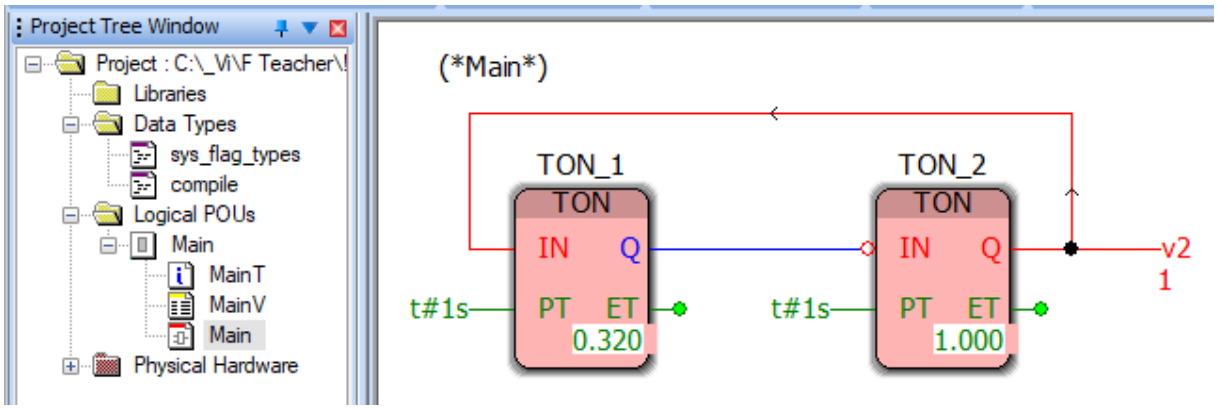


Рис.6.15. Отображение переменных в режиме отладки

В режиме отладки имеется возможность оперативного изменения значения переменной ***v2*** в реальном времени. Для этого нажать на ней двойным щелчком. Появится меню изменения значения переменной. В поле ***Value*** вписать: 100.0. Заодно изменим формат представления вещественных чисел. Значения отображаются в ***IEEE*** формате. Чтобы они отображались в обычном формате, необходимо снять галочку в пункте ***IEEE format*** и выставить другие опции, как показано на рис.6.16.

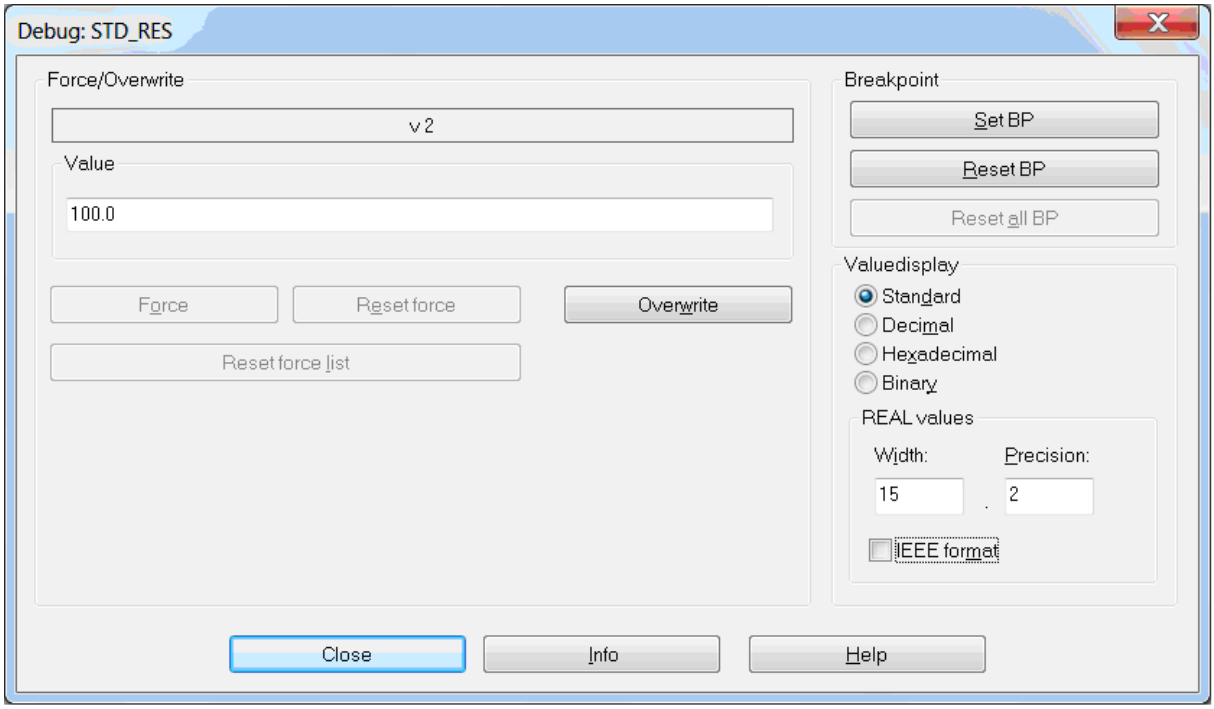


Рис.6.16. Оперативное изменение значения переменной

Последующее нажатие кнопок ***Overwrite*** и ***Cold*** приведет к обновлению значения переменной ***v2***.

7. Трассировка переменных программы средствами среды PC WORX6

В PC WORX можно осуществлять трассировку переменных программы, то есть строить графики зависимости переменных от времени. Имеется также возможность сохранить результаты трассировки в текстовый файл для последующего анализа.

Чтобы воспользоваться графической трассировкой следует:

- выбрать в главном меню PC Worx пункт **View -> Logic Analyzer**;
- запустить программу (кнопка ***Cold***) и включить режим отладки ***Debug on/off***  ;
- мышью перетащить переменные, которые необходимо отображать на графике, из рабочего поля программы в окно ***Logic Analyzer***. В примере предыдущего раздела это переменная ***v2***. В окне ***Logic Analyzer*** автоматически появятся оси графиков для каждой переменной;
- нажать на кнопку ***StartRecordingValues***  .

После выполнения указанных выше действий появится график зависимости переменной ***v2*** от номера выборки, приведенный на рис.7.1. Длительность регистрации составит 200 выборок. Поскольку длительность импульса составляет 1 секунду, то период выборок равен 20 мс.

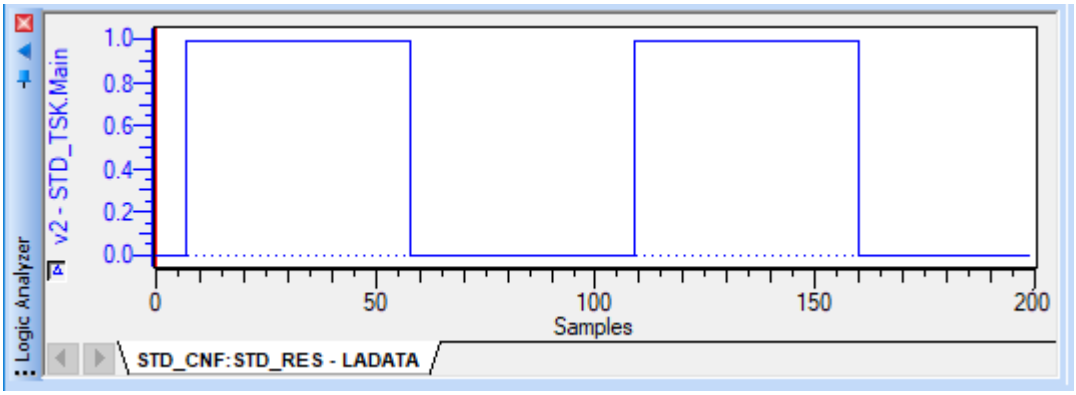


Рис.7.1. График зависимости переменной **v2** от номера выборки

Длительность регистрации можно установить путем вызова окна ***Trigger configuration***, показанного на рис.7.2, и выставления галочки в

поле опции ***Continuous recording***, при этом время будет отображаться в секундах. Начало записи инициируется кнопкой ***StartRecordingValues***  , останавливает процесс записи кнопкой ***StopRecordingValues***  . Список графиков вызывается кнопкой ***ConnectedVariables***  . В этом списке каждая переменная имеет поле с галочкой, позволяющей отключить график для конкретной переменной.

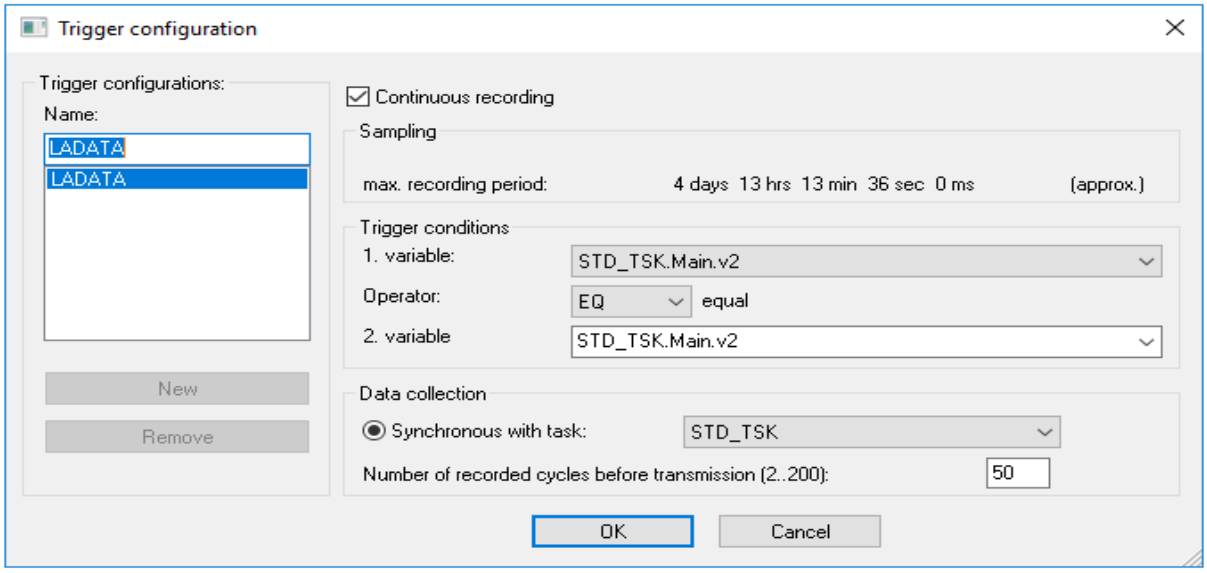


Рис.7.2. Окно ***Trigger configuration***

Результат формирования графика переменной v1 после конфигурации приведен на рис.7.3.

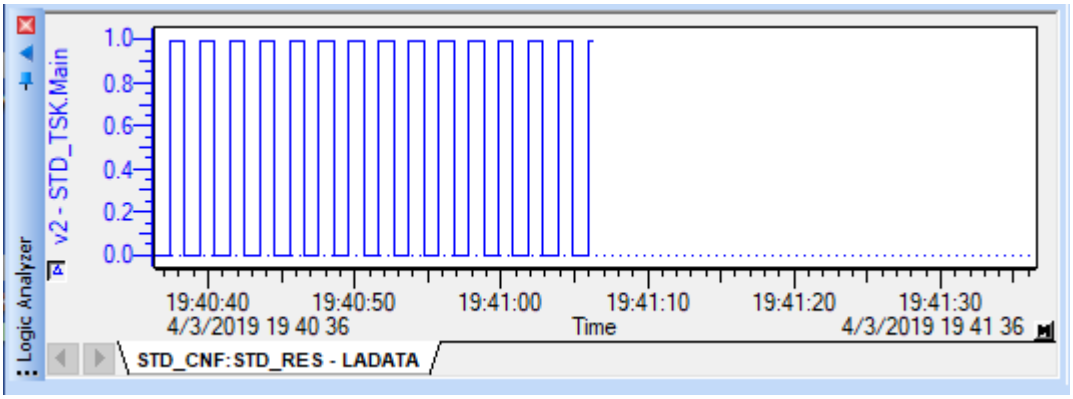


Рис.7.3. График зависимости переменной v1 с управляемой длительностью регистрации

Можно переместить, растянуть или сжать ось значений каждой переменной, чтобы увеличить или уменьшить масштаб в окне «***Logic***

Analyzer». Щелчок на оси времени левой кнопкой мышки позволяет смещать левую границу поля графика, при этом правая граница остается неизменной. Щелчок на оси времени правой кнопкой мышки позволяет смещать правую границу поля графика, при этом левая граница остается неизменной. Смещение границ поля графика позволяет сжимать или растягивать исследуемый участок графика.

Среда ***PC Worx*** предоставляет возможность сохранять результаты симуляции в текстовый файл, что может быть полезно для последующей математической обработки или для анализа и презентации результатов работы программы.

Для этого следует зайти в меню **Online->Logic Analyzer->Export Data**… и указать путь сохранения файла и его имя, например: ***data.csv***. Сохраняемый файл можно открыть в табличном редакторе Excel. Файл имеет открытый формат, его можно прочитать любой программой (MathCAD) или языком программирования (Pascal, C++).

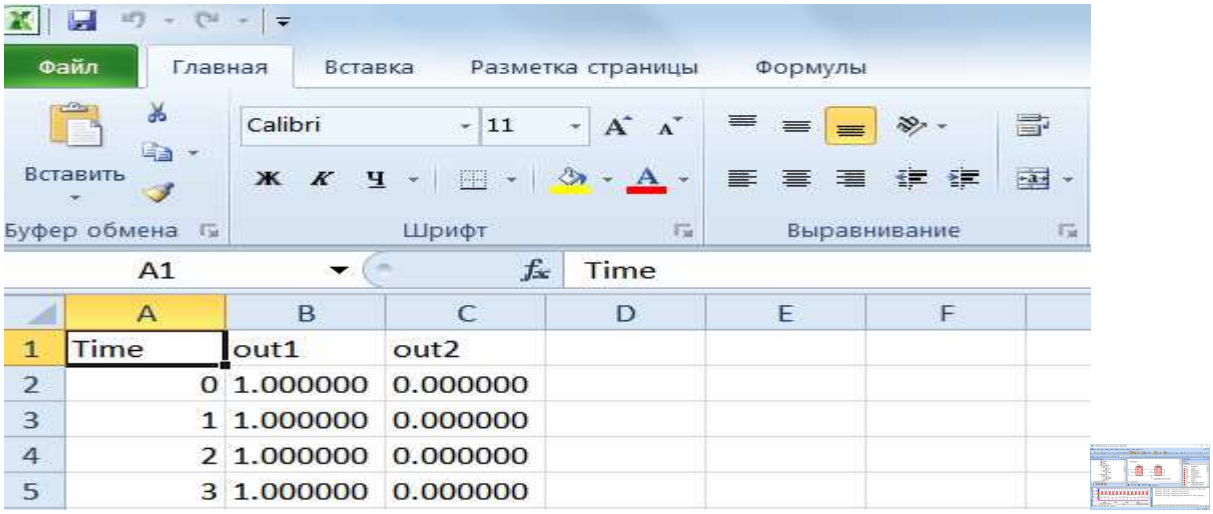


Рис.7.4. – Открытие файла данных в Excel

Первая колонка чисел в файле представляет собой время в десятых долях секунды, а следующие колонки – значения переменных.

8. Создание функциональных блоков средствами среды PC WORX6

Встроенная библиотека ***PC Worx*** имеет большое количество функциональных модулей: функций и функциональных блоков.

Функция – это подпрограмма, которая возвращает одно значение. То есть функция имеет некоторое количество входов и один выход. Примеры функций в ***PC Worx***: sin (синус числа), cos (косинус), asin (арксинус), acos (арккосинус), abs (модуль числа), add (сложение), mul (умножение) и т.д.

Функциональный блок (ФБ) – это подпрограмма, которая производит некоторые действия над входными переменными и выдает результат в выходные переменные. Функциональный блок обычно имеет несколько входов и несколько выходов (хотя может быть ни одного). Примеры функциональных блоков в PC Worx: TON (таймер с задержкой на включение), CTU (счетчик суммирующий), RS (rs-триггер), FILE_READ (чтение из файла).

Функциональный блок, в отличие от функции, имеет «время жизни» более одного цикла контроллера.

Третий тип функционального модуля в проекте – это программа (PRG). Программа выполняет некоторые действия и не имеет входов и выходов. Программам могут назначаться приоритеты планировщиком ОС контроллера. То есть несколько программ могут выполняться параллельно.

Среда ***PC Worx*** имеет возможность создания собственных функциональных модулей. В виде функционального модуля может быть оформлена отдельная функционально законченная часть программы.

В дальнейшем программист может пользоваться этим модулем, вызывая его сколько угодно раз. Наиболее нужные и часто используемые модули могут быть сохранены в виде собственных библиотек.

Создадим собственный функциональный блок, реализующий мультивибратор – генератор серии прямоугольных импульсов:

Для создания нового блока нажать правой кнопкой мыши на ***Logical POUs*** в дереве проекта. Выбрать ***Insert.. -> Function Block***. Выбрать язык ***FBD*** и имя блока ***mVibr***:, как показано на рис.8.1 и рис.8.2.

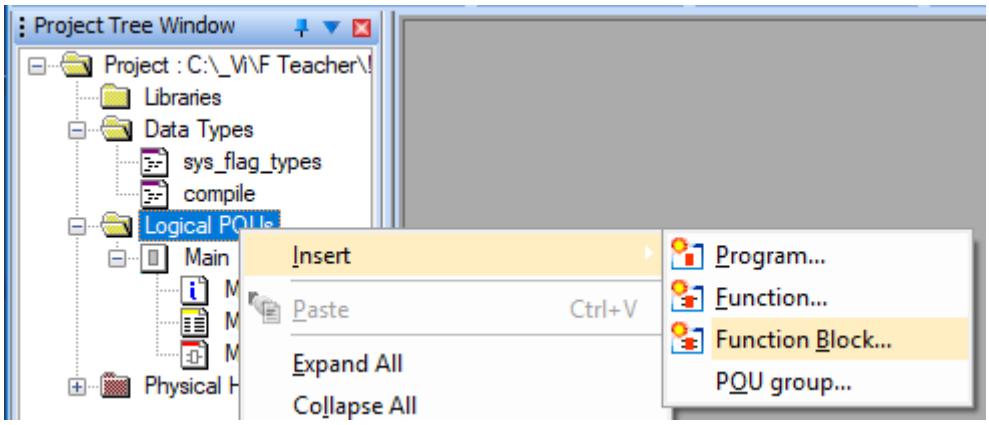


Рис.8.1. Создание функционального блока в дереве проекта

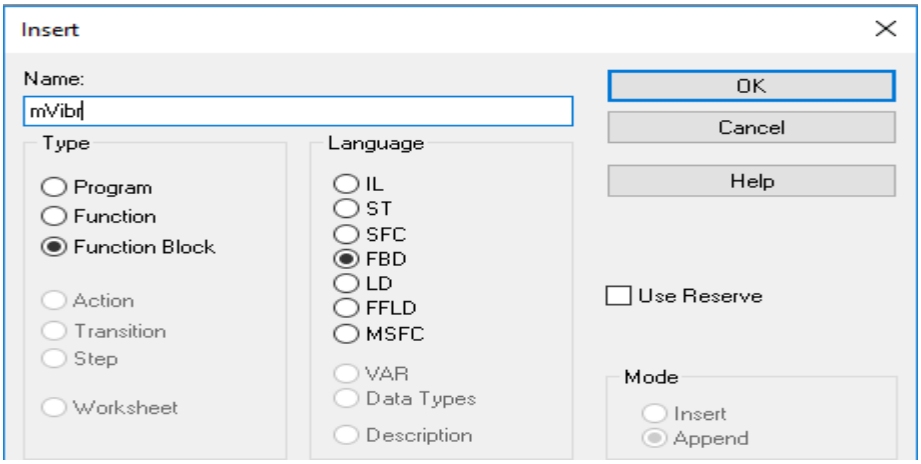


Рис.8.2. Выбор языка программирования функционального блока

В итоге, в дерево проекта, в группу проекта добавиться строка с названием ***mVibr***, как показано на рис.8.3. Раскрытие этой строки показывает, что блок включает 3 файла: комментарии, переменные и программу.

Создадим функциональный блок с двумя входами vA, vB и одним выходом vC. Входы vA и vB позволяют задавать длительность генерируемых импульсов и паузы между ними. В состав включим два таймера с задержкой на включение (TON_1, TON_2). Таймер TON_1 отвечает за длительность высокого уровня импульса, а таймер TON_2 – за длительность низкого уровня. Обратная связь между входом первого

таймера и выходом второго таймера совместно с инверсией сигнала на входе второго таймера создает условия возбуждения импульсов в схеме, приведенной на рис.8.3.

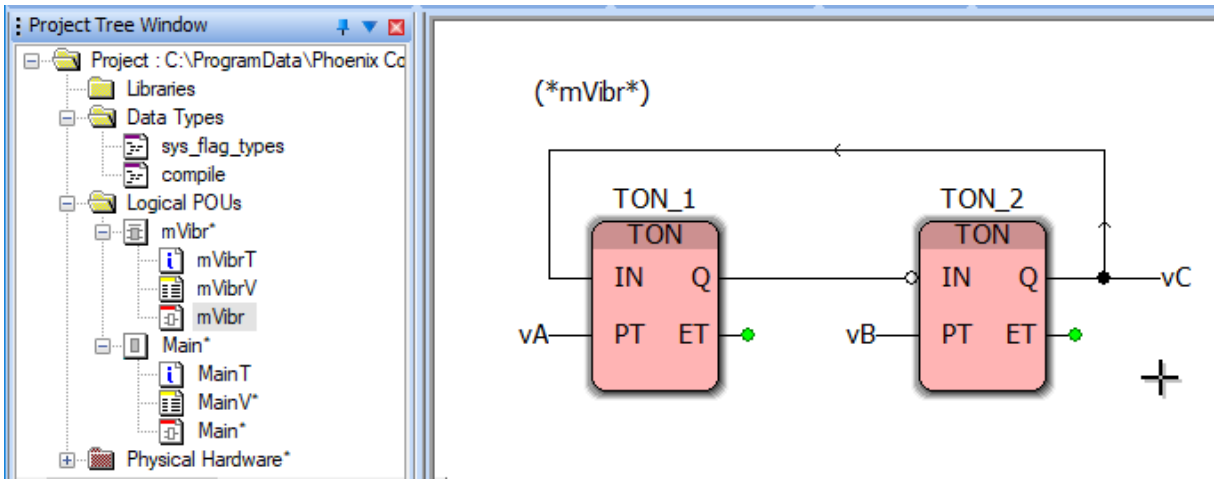


Рис.8.3. Программа внутри создаваемого функционального блока

Переменным vA, vB и vC при активации следует задать тип и характеристику связи с программой, как показано на рис.8.4.

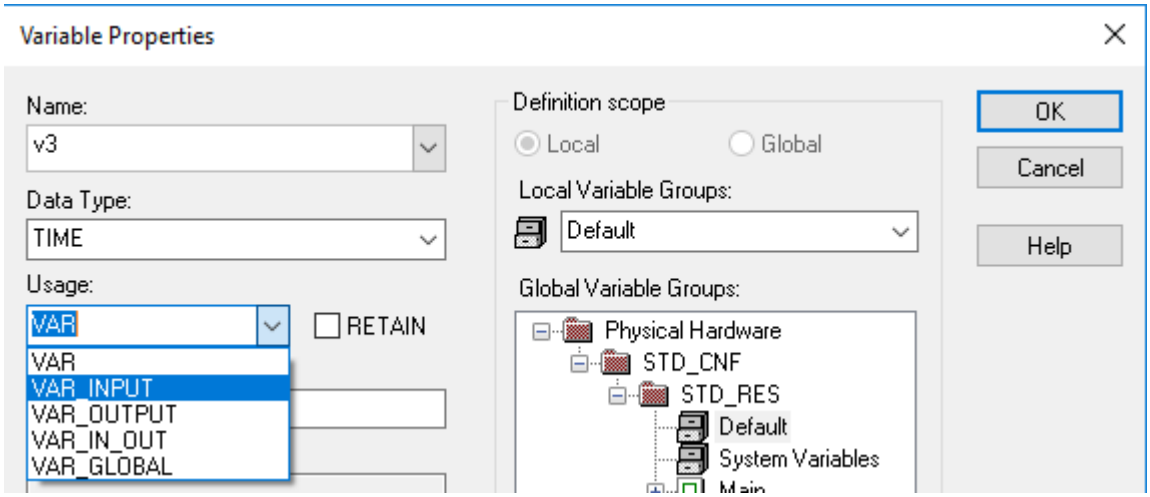


Рис.8.4. Задание для переменной типа и характеристики связи с программой

Контроль и при необходимости коррекция свойств переменных блока осуществляется через список переменных, показанный на рис.8.5. Щелчок на нужном поле списка вызывает список допустимых свойств переменной для коррекции. Следует иметь в виду, что среда программирования автоматически не удаляет введенные ранее ошибочные переменные, поэтому удаление таких переменных нужно выполнять вручную, иначе при отладке возникнет трудно диагностируемая ошибка.

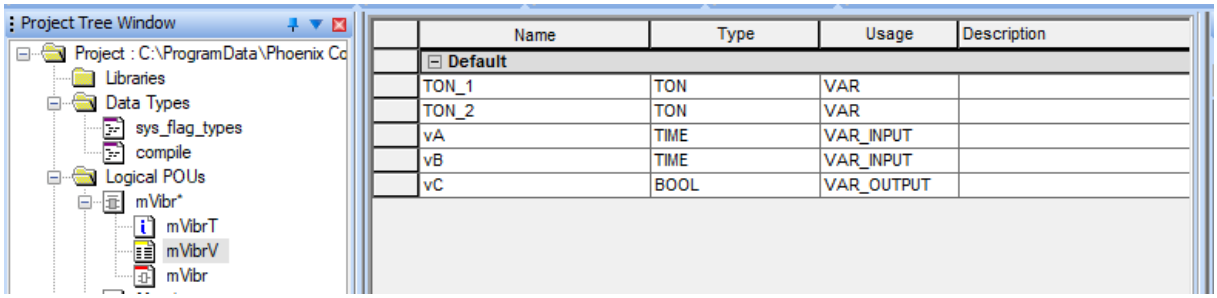


Рис.8.5. Список переменных создаваемого функционального блока

Компиляция функционального блока осуществляется командой ***Build->Compile WorkSheet***  **.** Если ошибок в синтаксисе не найдено, то система зарегистрирует новый функциональный блок в группе с именем проекта в ***Edit Wizard***, как показано на рис.8.6.

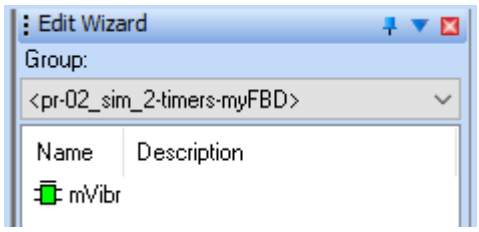


Рис.8.6. Новый функциональный блок в окне шаблонов функций и блоков

Переход в главную программу ***Main*** и перетаскивание нового функционального блока в поле главной программы позволяет использовать его в программе. Новый блок будет находиться в группе с именем проекта, если проект еще ни разу не сохранялся, то группа шаблонов будет называться ***Untitled***.

Результат установки нового функционального блока в главной программе ***Main*** показан на рис.8.7. Необходимо назначить переменные для каждого входа/выхода нового блока в соответствии с его типом данных. Пример назначения переменных новому блоку показан на рис.8.7. Дальнейшие действия по отладки программы описаны в предыдущих разделах.

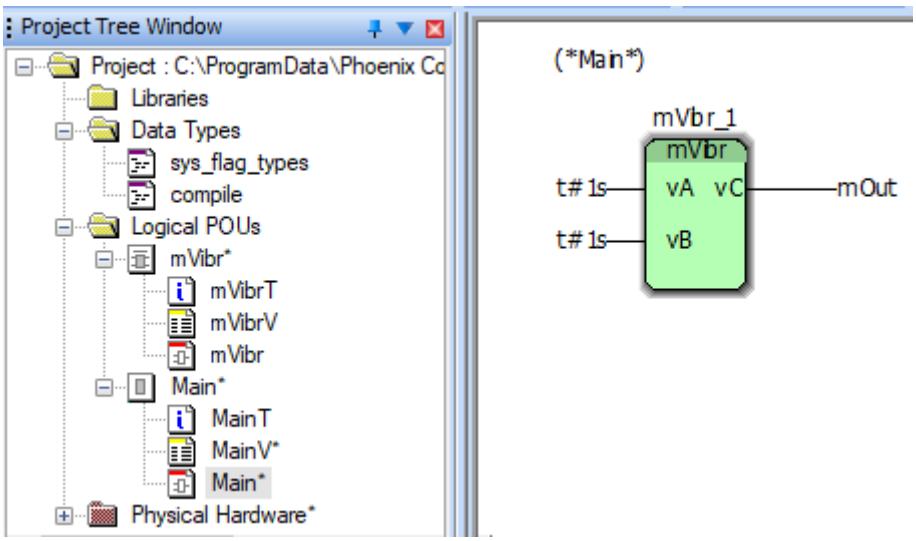


Рис.8.7. Новый функциональный блок в окне главной программы

9. Создание программ на языке SFC средствами среды PC WORX6

SFC (Sequential Function Chart) – это графический язык, который позволяет описать хронологическую последовательность различных действий в программе. Для этого действия ***actions*** связываются с шагами ***steps***, а последовательность работы определяется условиями переходов ***transitions*** между шагами. Действия и переходы представляют собой программные модули, созданные на одном из стандартных языков. Простейшая диаграмма показана на рис.9.1.

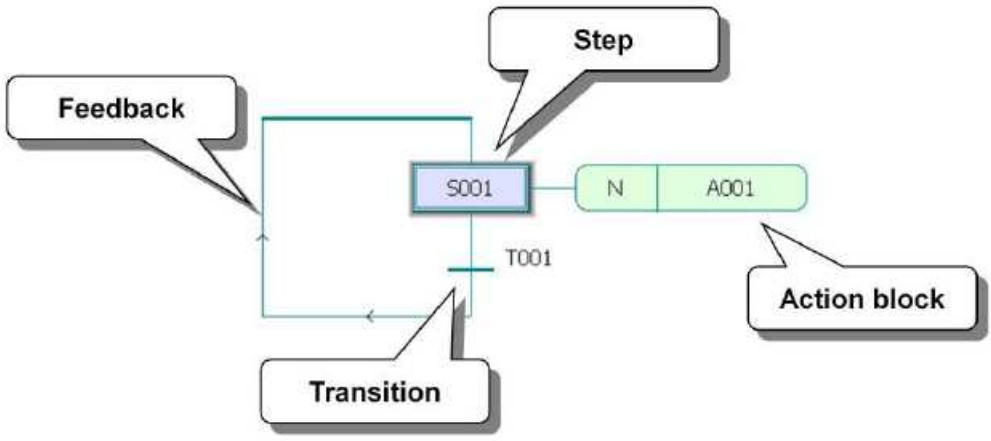


Рис.9.1. Основные элементы языка SFC

Создание программы на языке ***SFC*** начинается так же, как указано в разделе 5, однако, по умолчанию программа ***Main*** создается на языке ***FBD***. Для перевода программы ***Main*** на язык ***SFC*** нужно удалить старую программу на языке ***FBD*** и вставить новую – на языке ***SFC***. Указанные операции осуществляются после нажатия кнопки  - ***IEC Programming WorkSpace***, то есть в режиме программирования. ***SFC*** диаграмма создается сразу для первых двух последовательных шагов программы.

Удаление программы ***Main*** на языке ***FBD*** осуществляется путем выделения строки в окне проекта, как показано на рис.9.2 и нажатия кнопки ***Cut*** в контекстном меню.

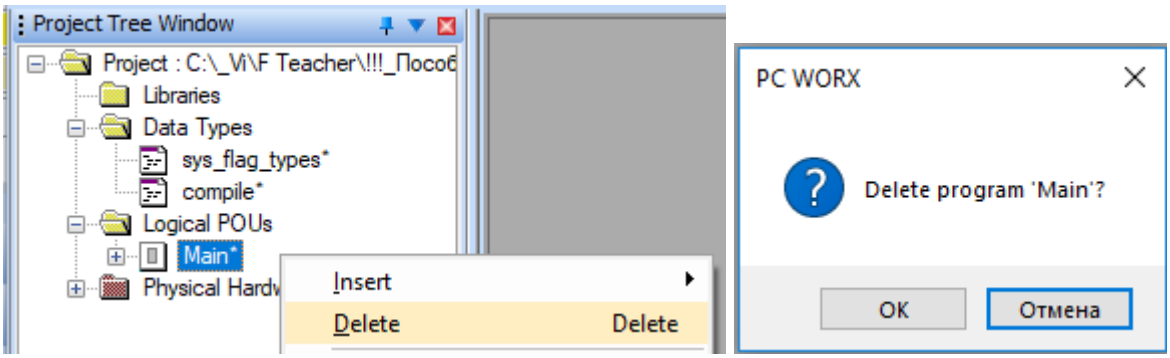


Рис.9.2. Удаление программы ***Main*** на языке ***FBD***

После удаления программы ***Main*** на языке ***FBD*** вставляется программа на языке ***SFC,*** как показано на рис.9.3 и рис.9.4.

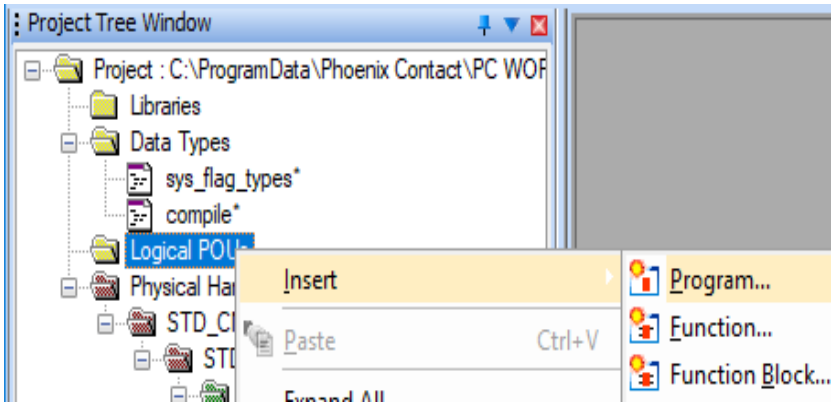


Рис.9.3. Вставка программы ***Main*** на языке ***SFC***

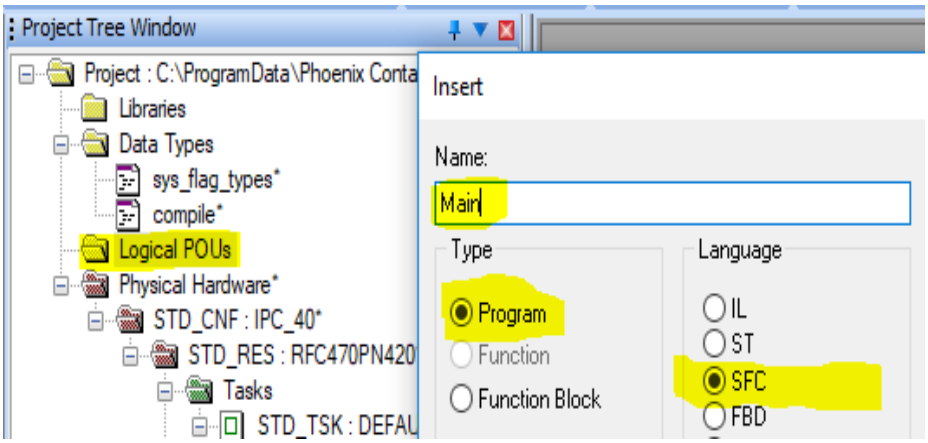


Рис.9.4. Указание языка ***SFC*** для программы ***Main***

После создания программы на языке ***SFC*** в окне проекта должны появиться две новые папки для переходов ***Transitions*** и действий ***Actions***, как показано на рис.9.5.

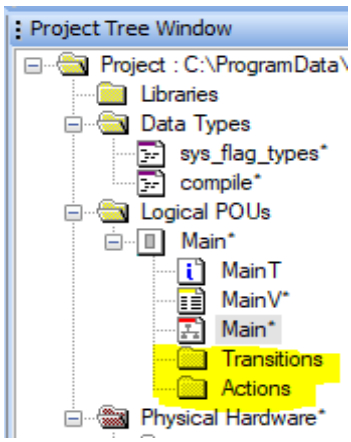


Рис.9.5. Структура программы на языке ***SFC*** в окне проекта

Переход в окно программы ***Main***, щелчок внутри окна и нажатие на кнопку меню  приводит к появленью простейшей диаграммы ***SFC***.

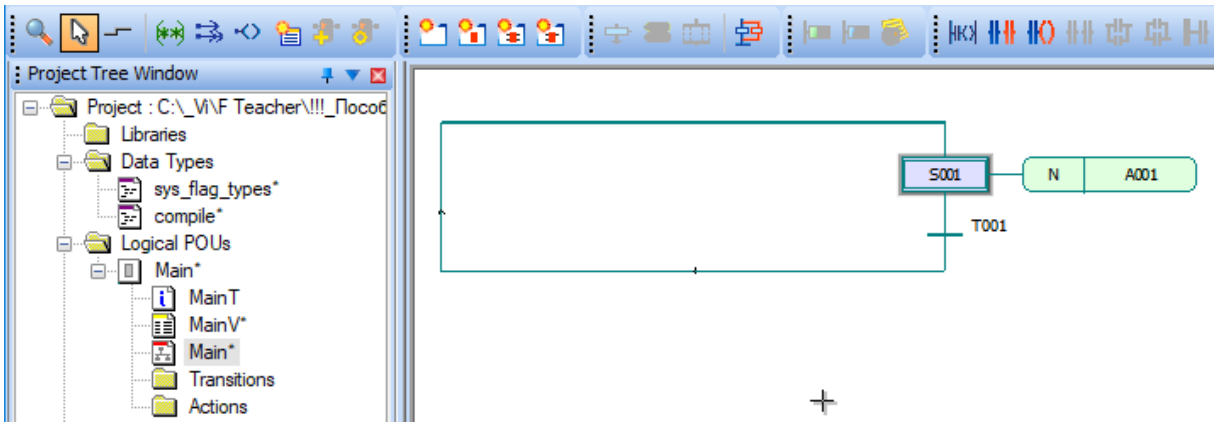


Рис.9.6. Простейшая диаграмма ***SFC***

Простейшая диаграмма ***SFC*** является замкнутой, что отображает факт наличия контроллера замкнутого рабочего цикла, то есть повторов обращения к программе с заданным периодом. Работоспособной программа станет после написания программ для действия ***A001*** и перехода ***T001***.

Диаграммы ***SFC*** создаются с помощью инструментов управляющей панели языка ***SFC***, на которой находятся следующие кнопки:

 - добавляет новое действие к выбранному шагу;

 - добавляет альтернативную или параллельную SFC ветвь.

 - добавляет шаг, действие и переход после выделенного перехода.

Альтернативная ветвь диаграммы формируется путем выделения фрагмента диаграммы, который начинается и заканчивается ***переходом***, с последующим нажатием на кнопку  .

Параллельная ветвь диаграммы формируется путем выделения фрагмента диаграммы, который начинается и заканчивается ***шагом***, с последующим нажатием на кнопку  .

9.1. Формирование действий на языке SFC средствами среды PC WORX6

Действие (Action) может быть двух типов:

- ***variable*** (привязанное напрямую к логической переменной, отображается розовым);
- ***detail*** (содержит программный код, отображается зеленым).

Дополнительно может использоваться модификатор действия (на рис.9.6 модификатор действия = N). Возможные значения модификатора перечислены в таблице 9.1. Модификаторы действий предназначены для

задания кратности повторения, привязки к переменным и времени, задержки пока активен соответствующий действию шаг.

Таблица 9.1.

Модификатор (qualifier)	Значение
N	Код внутри действия выполняется или переменная, привязанная к действию, равна TRUE всё время, *пока шаг активен*.
R	Код внутри действия больше не выполняется или переменная, привязанная к действию, равна FALSE, *пока не встретится такое же действие с модификатором S*.
S	Код внутри действия выполняется или переменная, привязанная к действию, равна TRUE, *пока не встретится такое же действие с модификатором R*.
L	Код внутри действия выполняется или переменная, привязанная к действию, равна TRUE, *но не дольше заданного времени*.
D	Код внутри действия начинает выполняться или переменная, привязанная к действию, становится TRUE *через заданное время* после входа программы в шаг.
P	Код внутри действия выполняется или переменная, привязанная к действию, равна TRUE *в течение одного цикла контроллера* при входе в шаг.

Также возможны три сочетания: SD, DS, SL.

Внутри кода действий программист может использовать вспомогательную переменную – **флаг активности шага**. Обращение к ней происходит так:

ИмяШага.x

Переменная имеет логический тип (***BOOL***). Пока шаг активен, переменная равна TRUE, иначе равна ***FALSE***. Эта переменная «видна» в любом шаге или переходе и может использоваться для завершения шага по алгоритму, находящемуся внутри одного из его действий.

По умолчанию действие (***Action***) создается с типом ***Detail***. Чтобы начать редактировать его код достаточно просто нажать двойным щелчком

на действии и выбрать язык программирования. Изменение типа действия и его модификатора осуществляется из контекстного меню действия (клик правой кнопкой мыши, затем выбор пункта ***Object Properties..***), как показано на рис.9.7.

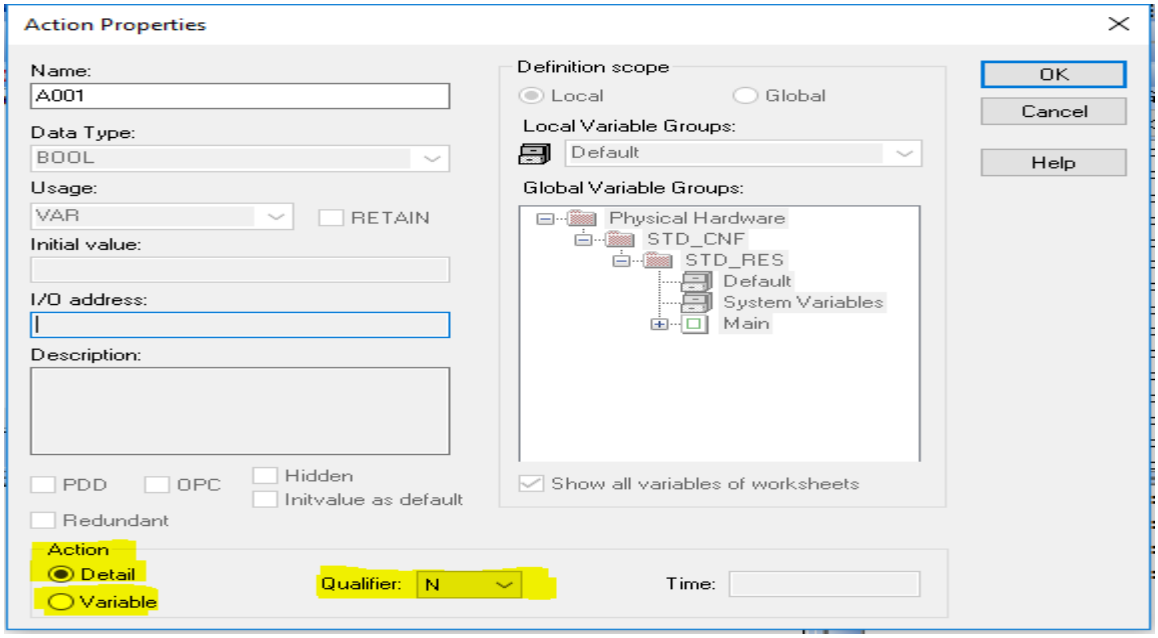


Рис.9.7. Выбор модификатора действия

Если действие должно быть ***Variable*** типа, то в поле имени сразу пишется имя логической переменной, а внизу выставляется флажок ***Variable*** (рис.9.7).

Если тип должен быть ***Detail***, то выбирается флажок ***Detail*** и нажимается ***OK***. Если затем нажать двойным щелчком на действии, то откроется окно выбора языка программирования, а далее окно редактирования внутреннего кода.

К одному шагу может быть привязано два и более действия, как показано на рис.9.8. Это можно сделать командой ***"Action Block.."*** из контекстного меню шага.

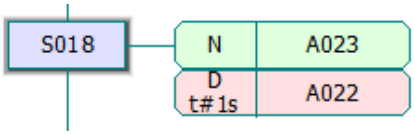


Рис.9.8. Шаг с двумя действиями

9.2. Формирование переходов на языке SFC средствами среды PC WORX6

Назначение переходов состоит в определении условия завершения входного шага и запуске выходного шага. Переходы бывают двух видов:

- ***Detail*** – переход содержит программный код, в котором должна присутствовать логическая переменная с именем имени перехода;
- ***Direct connection*** – переход представляет собой к логическую переменную.

Пример обозначения переходов на диаграмме

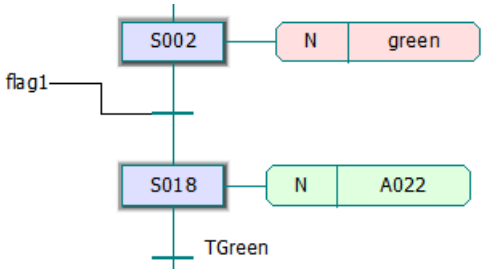


Рис.9.9. Переход ***flag*** типа ***direct connection*** и переход ***TGreen*** типа ***detail***

Верхний переход на рис.9.9 имеет тип ***direct connection*** и привязан к переменной ***flag1***. То есть, когда переменная ***flag1*** будет равна ***TRUE***, то завершится шаг ***S002*** и произойдет переход к шагу ***S018***. Нижний переход ***TGreen*** имеет тип ***detail***. Код внутри него должен содержать логическую переменную с именем ***TGreen***. Когда эта переменная будет равна ***TRUE***, произойдет переход к следующему шагу. Выбор типа перехода (***detail*** или ***direct connection***) осуществляется из контекстного меню перехода, вызываемого правой кнопкой мышки, с последующим выбором строки ***Object Properties->…***, как показано на рис.9.10.

Рис.9.10. Переключение типа перехода

Если переход имеет тип ***detail*** то достаточно указать его двойным щелчком мыши для выбора языка программирования и дальнейшего редактирования внутреннего кода. Если переход имеет тип ***direct connection***, то к нему следует привязать переменную. Вставить переменную можно с помощью кнопки меню  . Далее ее нужно соединить с переходом.

9.3. Формирование ветвлений на языке SFC средствами среды PC WORX6

Среда программирования позволяет создавать альтернативные и параллельные ветви в программе.

Параллельные ветви SFC

Параллельные ветви применяются для формирования группы шагов, действия которых выполняются параллельно. Начинаются и завершаются все шаги параллельной группы одновременно по условию, которое отслеживается общим переходом. Параллельная ветвь диаграммы формируется путем выделения фрагмента диаграммы, который начинается и заканчивается ***шагом***, с последующим нажатием на кнопку  .

Пример добавления параллельной ветви на уровне шага ***S002*** приведен на рис.9.11. Выделение шага ***S002*** и последующее нажатие на кнопку меню  приводит к появлению параллельной ветви с шагом ***S003*** и действием ***A003***. Новых переходов при этом не возникает. Начала и концы параллельных ветвей обозначены в виде парных параллельных прямых.

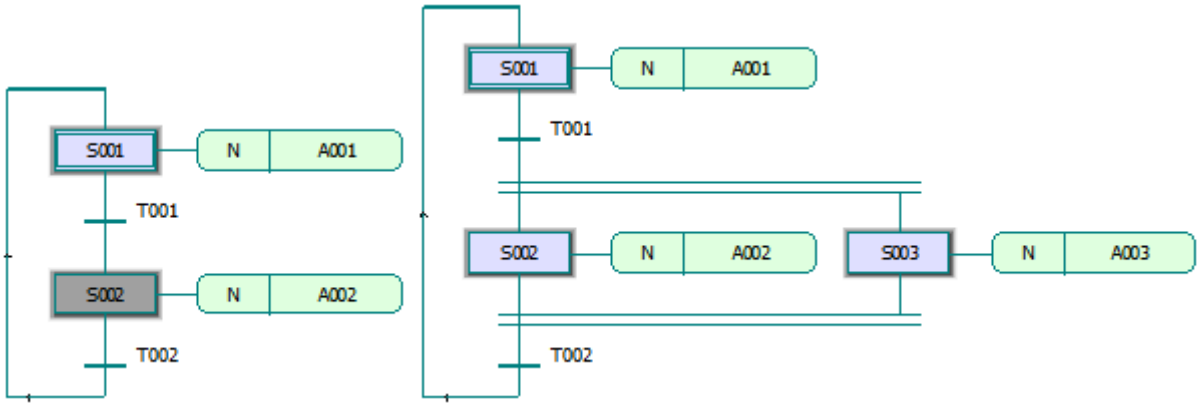


Рис.9.11. Формирование параллельной ветви

Параллельные ветви начинают функционировать по команде, формируемой переходом ***T001***, а завершают работу по команде перехода ***T002***, который является последним в рабочем цикле контроллера.

Альтернативные ветви SFC

Альтернативные ветви располагаются после перехода, который кроме условия завершения входной ветви может содержать алгоритм выбора одной ветви из совокупности альтернативных ветвей. Альтернативная ветвь диаграммы формируется путем выделения фрагмента диаграммы, который начинается и заканчивается ***переходом***, с последующим нажатием на кнопку  .

Пример добавления альтернативной ветви на уровне шага ***S002*** приведен на рис.9.12. Выделение переходов ***T001*** и ***T002*** мышкой с удержанием кнопки ***Ctrl*** и нажатие кнопки добавление ветви  позволило сформировать альтернативную ветвь с новым переходом ***T004***. Переходы ***T001*** и ***T004*** в совокупности составляют общее условие для завершения шага ***S001*** и условие выбора начала исполнения одной из альтернативных ветвей. В случае совпадения условий входных переходов совокупности альтернативных ветвей, приоритет исполнения определяется по их положению на диаграмме слева направо. Следует отметить, что к первому шагу невозможно добавить альтернативную и параллельную ветвь.

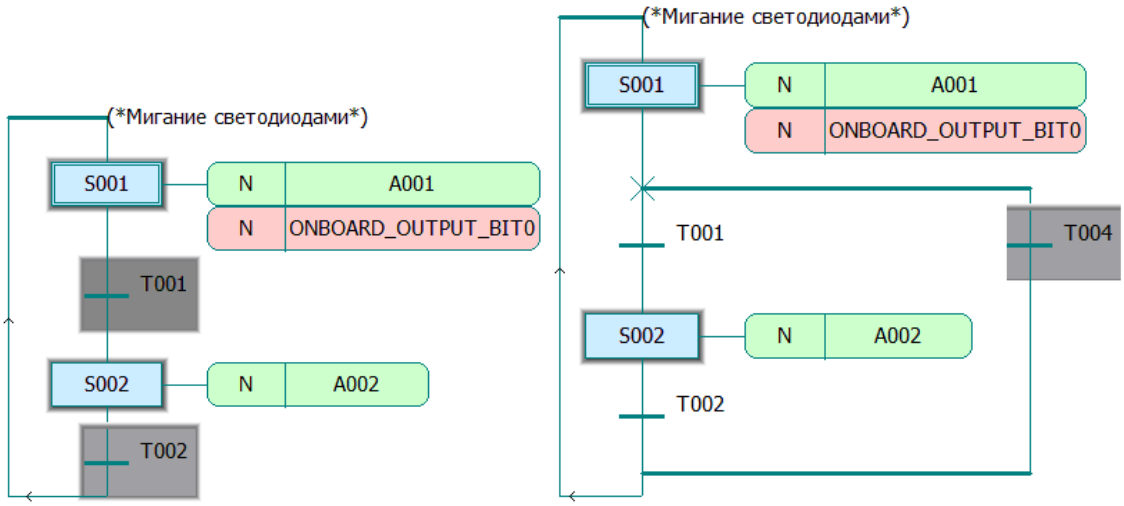


Рис.9.12. Формирование альтернативной ветви

Полученная альтернативная ветвь может быть дополнена шагом и действием. Для этого нужно выделить новый переход ***T004*** , как показано на рис.9.1, и нажать на кнопку вставки шага  . Появятся новый шаг ***S003*** с действием ***A004*** и новый переход ***T005***, определяющий условие завершения этого шага, что показано на рис.9.13.

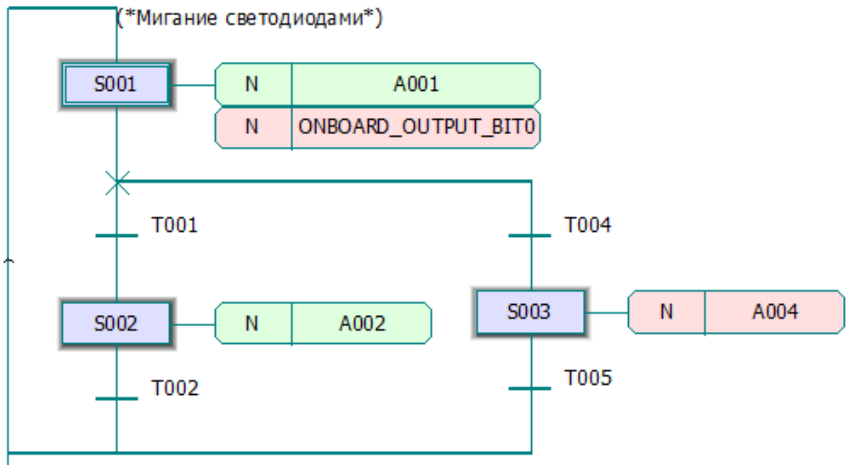


Рис.9.13. Вставка в альтернативную ветвь шага

Пример программы с альтернативными ветвями

Рассматривается пример программы двумя альтернативными ветвями, главная программа написана на языке ***SFC***, а действия и переходы на языке ***FBD***. Общая структура программы показана на рис.9.14, а список переменных на рис.9.15.

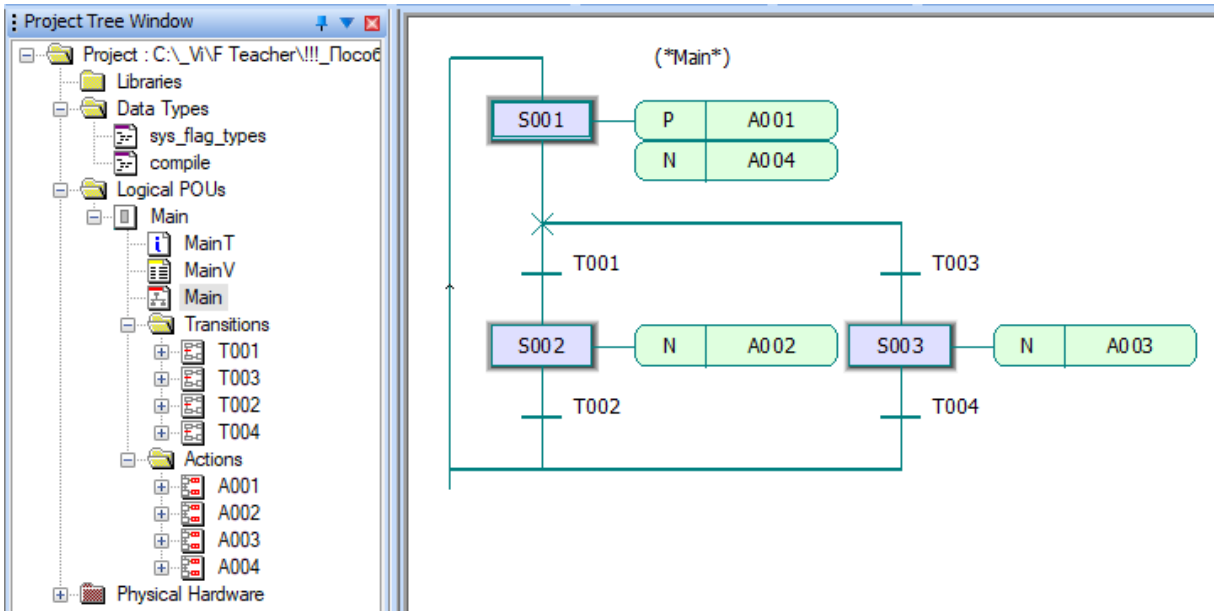


Рис.9.14. Окно проекта и программы ***Main***

	Name	Type	Usage
	Default		
	flag_c	INT	VAR
	TON_1	TON	VAR
	TON_2	TON	VAR
	TON_3	TON	VAR
	TON_4	TON	VAR
	out_2	BOOL	VAR
	TON_5	TON	VAR
	TON_6	TON	VAR
	out_3	BOOL	VAR
	TON_7	TON	VAR
	TON_8	TON	VAR
	TON_9	TON	VAR
	TON_10	TON	VAR

Рис. 9.15. Таблица переменных программы

Переключение альтернативных ветвей в данном примере осуществляется по циклическому счетчику, который реализован в действии ***A001***. Переходы ***T001*** и ***T003*** анализируют текущее значение ***flag_c*** счетчика, что позволяет срабатывать одному из них.

Программа действия ***A001*** представлена на рис.9.16 и состоит из блоков сравнения ***LE***, сложения ***ADD*** и оператора выборки ***SEL_INT***. Программа имеет модификатор ***P***, предписывающий выполнять её один раз в течение рабочего цикла, но поскольку программа повторяется при завершении рабочего цикла, то значение переменной ***flag_c*** в итоге увеличивается на 2, что подтверждает рис.9.17.

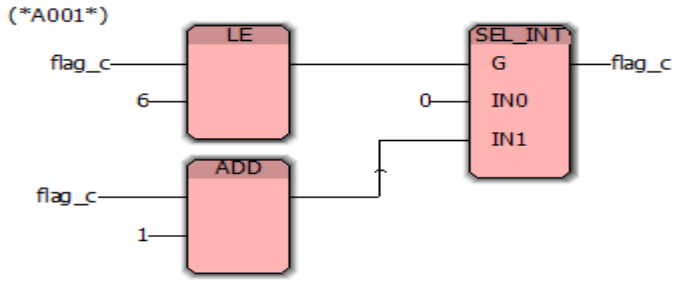


Рис.9.16. Программа действия ***A001***

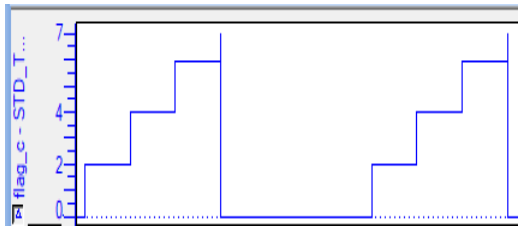


Рис.9.17. Циклическое изменение ***flag_c***

Программы действий ***A002*** и ***A003***, показанные на рис.9.18, структурно одинаковы и представляют собой генераторы периодических импульсов на таймерах задержки ***TON***. Эти генераторы принадлежат разным альтернативным ветвям и работают по очереди.

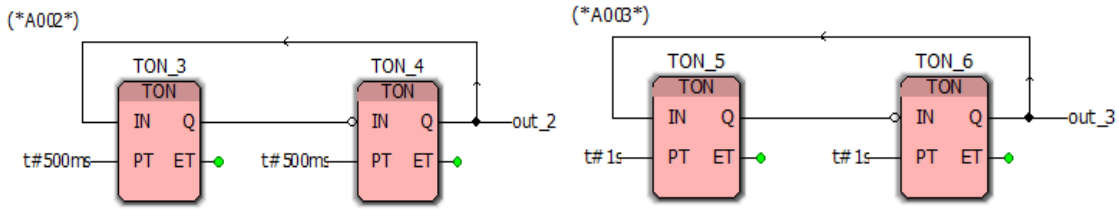


Рис.9.18. Программы действий ***A002*** и ***A003***

Действие ***A004*** генерирует периодические импульсы, является вторым действием первого шага и показано на рис.9.19.

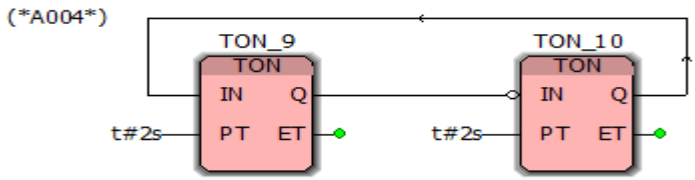


Рис.9.19. Программа действия ***A004***

Переходы ***T001*** и ***T003*** имеют одинаковую структуру. Они осуществляют управление событиями первого шага по входу ***S001.x*** и

решают задачу альтернативного ветвления в соответствии с текущим значением переменной ***flag_c***.

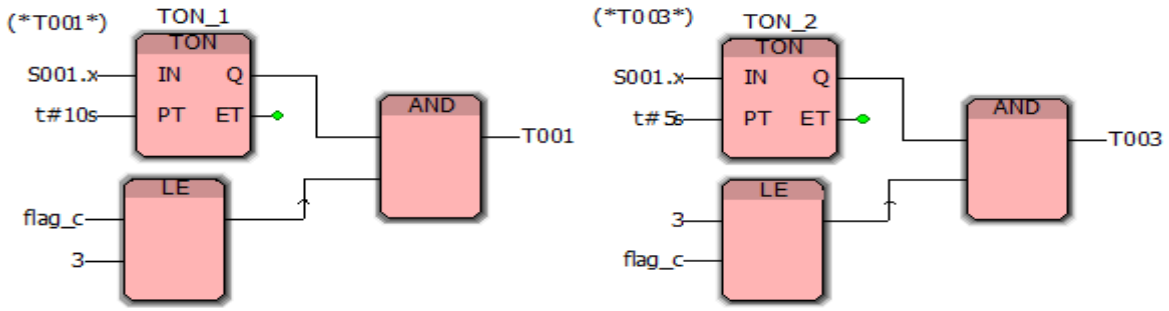


Рис.9.20. Программы переходов ***T001*** и ***T003***

Переходы ***T002*** и ***T004*** имеют одинаковую структуру. Они осуществляют управление длительностью функционирования второго и третьего шагов по входам ***S002.x*** и ***S003.x***.

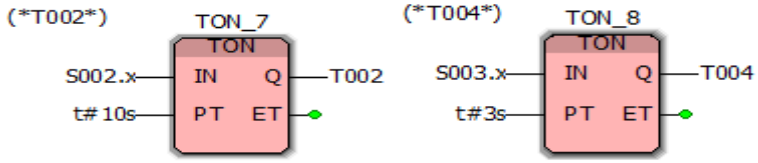


Рис.9.21. Программы переходов ***T002*** и ***T004***

Описанный пример программы с альтернативными ветвями был откомпилирован в режиме симуляции. Зависимости параметров от времени были визуализированы средствами графического анализатора и приведены на рис.9.22.

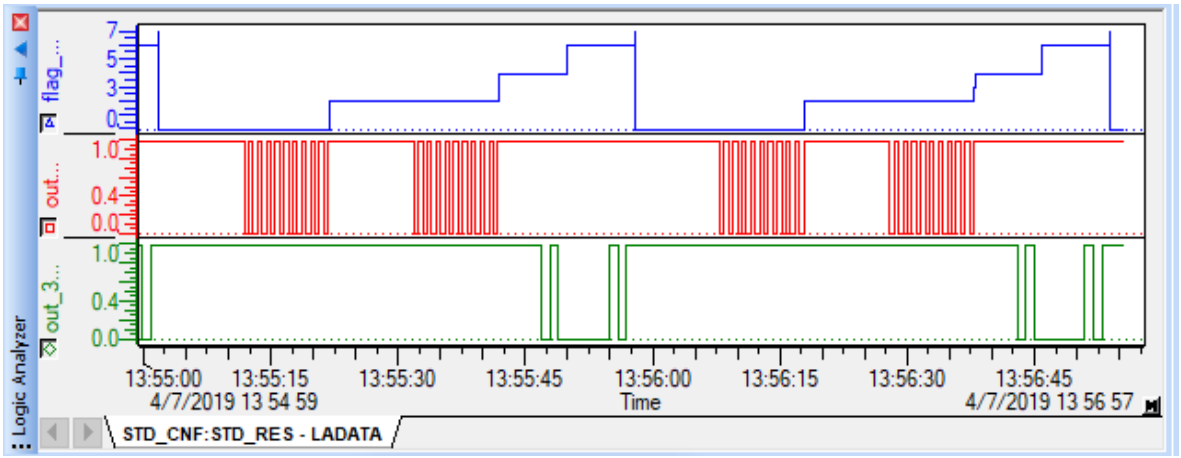


Рис.9.22. Диаграммы выходных сигналов

Приведенные на рис.9.22 графики иллюстрируют циклический процесс переключения альтернативных ветвей.

Литература

1. Петренко Ю.Н., Новиков С.О., Гончаров А.А. Программное управление технологическими комплексами в энергетике. – Минск:Высш.Шк., 2013. – 405 с.
2. Андреев Е.Б. Современные технологии автоматизации. Учебное пособие для вузов. - М.: ООО «Недра-Бизнесцентр», 2009. - 240 с.
3. Андреев Е.Б., Куцевич Н.А., Синенко О.В. SCADA-системы: взгляд изнутри. М.: Издательство «РТСофт», 2004. - 176 с.
4. Дорф Р., Бишоп Р. Современные системы управления. М.: Лаборатория Базовых Знаний, 2002. – 871с.
5. Автоматизация и управление в технологических комплексах / А. М. Русецкий [и др.]; под общ. ред. А. М. Русецкого. – Минск: Беларуская навука, 2014. – 375 с.
6. Петров И.В. Программируемые контроллеры. Стандартные языки и приемы прикладного программирования. М.: СОЛОН-Пресс, 2004. – 256 с.
7. Советов Б.Я. Теоретические основы автоматизированного управления. М.: Высш. шк., 2006. – 463 с.
8. Гейдаманин Н.А. Автоматизированные информационные системы. М.: Гелиос, 2002. – 243 с.
9. Пьявченко Т.А. Автоматизированные информационно-управляющие системы с применением SCADA-системы TRACE MODE. СПб.: «Лань», 2015. – 336 с.
10. Пономарев О.П. Наладка и эксплуатация средств автоматизации. SCADA-системы. Калининград: КВШУ, 2006. – 78 с.
11. Андреев Е.Б., Куцевич Н.А., Синенко О.В. SCADA-системы: взгляд изнутри. М.: «РТСофт», 2004. – 181 с.
12. Лопатин А.Г., Киреев П.А. Методика разработки систем управления на базе SCADA системы Trace Mode. Новомосковск: РХТУ, 2007. – 112 с.
13. Елизаров И.А. и др. Интегрированные системы проектирования и управления: SCADA-системы. Тамбов: ТГТУ, 2015. – 160 с.
14. Чикуров Н.Г. Алгоритмическое и программное обеспечение компьютерных систем управления. Уфа: УГАТУ, 2008. – 227 с.
15. Федорович О.Е. Системы промышленной автоматизации на основе технологии SCADA. Харьков: ХАИ. 2007. – 126 с.
16. Таненбаум Э., Бос Х. Современные операционные системы. СПб.: Питер, 2015. – 1120 с.
17. PC WORX 6 IEC 61131 – Programming. Phoenix Contact GmbH & Co. KG, 2010 – 442 p.

Электронное учебное издание

Виктор Иванович **Капля,**
Андрей Георгиевич **Бурцев,**
Степан Игоревич **Ефремкин**

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ СИСТЕМ УПРАВЛЕНИЯ

Учебное пособие

Электронное издание сетевого распространения

Редактор Матвеева Н.И.

Темплан 2019 г. Поз. № 30.
Подписано к использованию 06.05.2019. Формат 60x84 1/16.
Гарнитура Times. Усл. печ. л. 5,0.

Волгоградский государственный технический университет.
400005, г. Волгоград, пр. Ленина, 28, корп. 1.

ВПИ (филиал) ВолгГТУ.
404121, г. Волжский, ул. Энгельса, 42а.