ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

УПРАВЛЕНИЕ ДИСТАНЦИОННОГО ОБУЧЕНИЯ И ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

Кафедра «Автоматизация производственных процессов»

Практикум

по дисциплине

«Микропроцессорные системы управления»

Авторы
Аль-Тибби В.Х.,
Тимофеев В.И.

Ростов-на-Дону, 2018

Аннотация

Практикум предназначен для студентов очной формы обучения направления 150304 «Автоматизация технологических процессов и производств».

Авторы

Доцент, к.т.н.,
доцент,
кафедры «Автоматизация производственных процессов»
Аль-Тибби В.Х.

Доцент,
доцент,
кафедры «Автоматизация производственных процессов»
Тимофеев В.И.

Оглавление

ОБЩИЕ СВЕДЕНИЯ О ПЛК

Данный практикум описывают программирование систем управления промышленной автоматикой, построенных на базе программируемых логических контроллеров (ПЛК). Основное внимание уделяется технологии создания программного обеспечения для систем, построенных на базе ПЛК и практическому программированию на языках стандарта МЭК 61131-3.

К сожалению, период широкого распространения стандарта МЭК 61131-3 пришелся в России на годы перестройки. Отсутствие спроса промышленности на средства автоматизации производства привело к распаду большинства коллективов, занятых применением ПЛК, что, естественно, отразилось и на уровне подготовки специалистов. В настоящее же время наблюдается существенный рост потребности в современных инструментах производства и автоматики. Сегодня персональные компьютеры (ПК) массово применяют на всех уровнях промышленной автоматизации, включая классические контроллерные задачи.

Между тем во многих случаях применение промышленных ПК не оправдано экономически и технически сложно. Даже там, где задача на ПЛК решается «в одно действие» и на два порядка дешевле, нередко применяют дорогостоящие промышленные ПК, операционные системы реального времени и заказное программное обеспечение. Единственной причиной такого подхода является наличие подготовленных специалистов.

В последнее десятилетие появился целый класс инструментов визуального прикладного проектирования для ПЛК. Понятие «программирование» для контроллеров все более вытесняется словом «проектирование». И действительно, процесс перетаскивания мышью графических объектов называть программированием сложно.

Следует сразу обратить внимание на то, что для программирования ПЛК не требуется знание всех пяти языков МЭК 61131-3. Так, используя даже простейший, похожий на ассемблер язык IL (список инструкций), можно реализовать проект любой сложности. В то же время выбор языка существенно влияет на способ мышления. В результате существует много задач, красивое решение которых на одном языке получается практически без усилий, а на другом языке требует применения малопонятных «трюков» и, естественно, серьезной отладки. Овладение же приемами работы на всех языках и возможность совмещения их в одной задаче позволяют работать быстро и надежно.

Определение ПЛК

Любая машина, способная автоматически выполнять некоторые операции, имеет в своем составе управляющий контроллер — модуль, обеспечивающий логику работы устройства. Технически контроллеры реализуются по-разному. Это может быть механическое устройство, пневматический или гидравлический автомат, релейная или электронная схема или даже компьютерная программа.

При создании машин, занятых в сфере промышленного производства, как правило, приходится иметь дело не более чем с единицами однотипных устройств. Кроме того, очень существенной здесь является возможность быстрой перенастройки оборудования на выпуск другой продукции.

В отличие от

- микроконтроллера, микросхемы предназначенной для управления электронными устройствами, областью применения ПЛК обычно являются автоматизированные процессы промышленного производства, в масштабе производственного предприятия;

- компьютеров, ПЛК имеют развитые устройства ввода-вывода сигналов датчиков и исполнительных механизмов в противовес слабым возможностям ввода-вывод стандартного ПК.

- встраиваемых систем - ПЛК устанавливается отдельно от управляемого при его помощи оборудования.

Физически, типичный ПЛК представляет собой блок, имеющий определенный набор выходов и входов, для подключения датчиков и исполнительных механизмов. Логика управления описывается программно на основе микрокомпьютерного ядра. Абсолютно одинаковые ПЛК могут выполнять совершенно разные функции. Причем для изменения алгоритма работы не требуется каких-либо переделок аппаратной части. Аппаратная реализация входов и выходов ПЛК ориентирована на сопряжение с унифицированными приборами и мало подвержена изменениям.

Рисунок 1 - Принцип работы ПЛК

Задачей прикладного программирования ПЛК является только реализация алгоритма управления конкретной машиной. Опрос входов и выходов контроллер осуществляет автоматически, вне зависимости от способа физического соединения. Эту работу выполняет системное программное обеспечение. В идеальном случае прикладной программист совершенно не интересуется, как подсоединены и где расположены датчики и исполнительные механизмы. Мало того, его работа не зависит от того, с каким контроллером и какой фирмы он работает. Благодаря стандартизации языков программирования прикладная программа оказывается переносимой. Это означает, что ее можно использовать в любом ПЛК, поддерживающем данный стандарт.

Программируемый контроллер — это программно управляемый дискретный автомат, имеющий некоторое множество входов, подключенных посредством датчиков к объекту управления, и множество выходов, подключенных к исполнительным устройствам. ПЛК контролирует состояния входов и вырабатывает определенные последовательности программно заданных действий, отражающихся в изменении выходов.

ПЛК предназначен для работы в режиме реального времени в условиях промышленной среды и должен быть доступен для программирования неспециалистом в области информатики.

Изначально ПЛК предназначались для управления последовательными логическими процессами, что и обусловило слово «логический» в названии ПЛК. Современные ПЛК помимо простых логических операций способны выполнять цифровую обработку сигналов, управление приводами, регулирование, функции операторского управления и т. д. В стандарте МЭК и очень часто в литературе для обозначения контроллеров применяется сокращение ПК (программируемый контроллер). Поскольку в России обозначение ПК устойчиво связано с персональными компьютерами, мы будем использовать сокращение ПЛК.

Конструкция ПЛК может быть самой разнообразной — от стойки, заполненной аппаратурой, до совсем миниатюрных ПЛК.

Впервые ПЛК были применены в США для автоматизации конвейерного сборочного производства в автомобильной промышленности (фирма Модикон, 1969 г.). Сегодня ПЛК работают в энергетике, в области связи, в химической промышленности, в сфере добычи, транспортировки нефти и газа, в системах обеспечения безопасности, в коммунальном хозяйстве, используются в автоматизации складов, в производстве продуктов питания и

напитков, на транспорте, в строительстве и т. д. Реально сфера применения ПЛК даже шире сферы применения персональных компьютеров. Хотя слава ПЛК значительно меньше.

Входы-выходы

На заре своего появления ПЛК имели только бинарные входы, т. е. входы, значения сигналов на которых способны принимать только два состояния — логического нуля и логической единицы. Датчиками, формирующими такой сигнал, являются кнопки ручного управления, концевые датчики, датчики движения, контактные термометры и многие другие.

Бинарный выход также имеет два состояния — включен и выключен. Сфера применения бинарных выходов очевидна: электромагнитные реле, силовые пускатели, электромагнитные клапаны, световые сигнализаторы и т. д.

В современных ПЛК широко используются аналоговые входы и выходы. Аналоговый или непрерывный сигнал отражает уровень напряжения или тока, соответствующий некоторой физической величине в каждый момент времени. Этот уровень может относиться к температуре, давлению, весу, положению, скорости, частоте и т. д. Словом, к любой физической величине.

Особые классы аналоговых входов представляют входы, предназначенные для подключения термометров сопротивления и термопар. Здесь требуется применение специальной аппаратной поддержки (трехточечное включение, источники образцового тока, схемы компенсации холодного спая, схемы линеаризации и т. д.).

Помимо «классических» дискретных и аналоговых входов-выходов многие ПЛК имеют специализированные входы выходы. Они ориентированы на работу с конкретными специфическими датчиками, требующими определенных уровней сигналов, питания и специальной обработки. Например, квадратурные шифраторы, блоки управления шаговыми двигателями, интерфейсы дисплейных модулей и т. д.

Входы-выходы ПЛК не обязательно должны быть физически сосредоточены в общем корпусе с процессорным ядром. В последние годы все большую популярность приобретают технические решения, позволяющие полностью отказаться от прокладки кабелей для аналоговых цепей. Входы-выходы выполняются в виде миниатюрных модулей, расположенных в непосредственной

близости от датчиков и исполнительных механизмов. Соединение подсистемы ввода-вывода с ПЛК выполняется посредством одного общего цифрового кабеля. Например, в интерфейсе Actuators/Sensors interface (ASI) применяется плоский профилированный кабель («желтый кабель») для передачи данных и питания всего по двум проводfv

Режим реального времени и ограничения на применение ПЛК

Для математических систем характеристикой качества работы является правильность найденного решения. В системах реального времени помимо правильности решения определяющую роль играет время реакции. Логически верное решение, полученное с задержкой более допустимой, не является приемлемым.

Принято различать системы жесткого и мягкого реального времени. В системах жесткого реального времени существует выраженный временной порог. При его превышении наступают необратимые катастрофические последствия. В системах мягкого реального времени характеристики системы ухудшаются с увеличением времени управляющей реакции. Система может работать плохо или еще хуже, но ничего катастрофического при этом не происходит.

Классический подход для задач жесткого реального времени требует построения событийно управляемой системы. Для каждого события в системе устанавливается четко определенное время реакции и определенный приоритет. Практическая реализация таких систем сложна и всегда требует тщательной проработки и моделирования.

Для ПЛК существенное значение имеет не только быстродействие самой системы, но и время проектирования, внедрения и возможной оперативной переналадки.

Абсолютное большинство ПЛК работают по методу периодического опроса входных данных (сканирования). ПЛК опрашивает входы, выполняет пользовательскую программу и устанавливает необходимые значения выходов. Специфика применения ПЛК обусловливает необходимость одновременного решения нескольких задач. Прикладная программа может быть реализована в виде множества логически независимых задач, которые должны работать одновременно.

На самом деле ПЛК имеет обычно один процессор

И выполняет несколько задач псевдопараллельно, последовательными порциями. Время реакции на событие оказывается зависящим от числа одновременно обрабатываемых событий. Рассчитать минимальное и максимальное значения времени реакции, конечно, можно, но добавление новых задач или увеличение объема программы приведет к увеличению времени реакции. Такая модель более подходит для систем мягкого реального времени. Современные ПЛК имеют типовое значение времени рабочего цикла, измеряемое единицами миллисекунд и менее. Поскольку время реакции большинства исполнительных устройств значительно выше, с реальными ограничениями возможности использования ПЛК по времени приходится сталкиваться редко.

В некоторых случаях ограничением служит не время реакции на событие, а обязательность его фиксации, например, работа с датчиками, формирующими импульсы малой длительности. Это ограничение преодолевается специальной конструкцией входов. Так, счетный вход позволяет фиксировать и подсчитывать импульсы с периодом во много раз меньшим времени рабочего цикла ПЛК. Специализированные интеллектуальные модули в составе ПЛК позволяют автономно отрабатывать заданные функции, например, модули управления сервоприводом.

Условия работы ПЛК

К негативным факторам, определяющим промышленную среду, относятся: температура и влажность, удары и вибрация, коррозионно активная газовая среда, минеральная и металлическая пыль, электромагнитные помехи. Перечисленные факторы, весьма характерные для производственных условий, обусловливают жесткие требования, определяющие схемотехнические решения, элементную и конструктивную базу ПЛК. В процессе серийного производства ПЛК обязательным является технический прогон готовых изделий, включающий климатические, вибрационные и другие испытания.

ПЛК — это конструктивно законченное изделие, физическое исполнение которого определяется требуемой степенью защиты, начиная от контроллеров в легких пластиковых корпусах, предназначенных для монтажа в шкафу (IP20), и до герметичных устройств в литых металлических корпусах, предназначенных для работы в особо жестких условиях (IP60).

IP - степень защиты оболочки (Internal Protection), Ingress Protection Rating — система классификации степеней защиты оболочки электрооборудования от проникновения твёрдых предме-

тов и воды в соответствии с международным стандартом IEC 60529 (DIN 40050, ГОСТ 14254-96).

Под степенью защиты понимается способ защиты, проверяемый стандартными методами испытаний, который обеспечивается оболочкой от доступа к опасным частям (опасным токоведущим и опасным механическим частям), попадания внешних твердых предметов и (или) воды внутрь оболочки.

Маркировка степени защиты оболочки электрооборудования осуществляется при помощи международного знака защиты (IP) и двух цифр, первая из которых означает защиту от попадания твердых предметов, вторая — от проникновения воды.

За цифрами могут идти одна или две буквы, дающие вспомогательную информацию. Например, бытовая электрическая розетка может иметь степень защиты IP22 — она защищена от проникновения пальцев и не может быть повреждена вертикально или почти вертикально капающей водой. Максимальная защита по этой классификации — IP68: пыленепроницаемый прибор, выдерживающий длительное погружение в воду.

Правильно подобранный по условиям эксплуатации контроллер нельзя повредить извне без применения экстремальных методов. Штатными для ПЛК являются такие аппаратные решения, как полная гальваническая развязка входов-выходов, защита по току и напряжению, зеркальные выходные каналы, сторожевой таймер задач и микропроцессорного ядра.

Интеграция ПЛК в систему управления предприятием

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

Далеко не всегда удается создать полностью автоматическую систему. Часто «общее руководство» со стороны квалифицированного человека — диспетчера необходимо. В отличие от автоматических систем управления такие системы называют автоматизированными. Еще 10 — 15 лет назад диспетчерский пульт управления представлял собой табло с множеством кнопок и све-

товых индикаторов. В настоящее время подобные пульты применяются только в очень простых случаях, когда можно обойтись несколькими кнопками и индикаторами. В более «серьезных» системах применяются ПК.

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

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

Все это «многоэтажное» объяснение призвано подчеркнуть еще одно немаловажное преимущество ПЛК — средства системной интеграции являются составной частью базового программного обеспечения современного ПЛК (рис. 2). Допустим, вы написали и отладили автономный проект на контроллере при помощи системы подготовки программ CoDeSys. Как теперь нужно доработать программу, чтобы связать ПЛК с системой диспетчерского управления, базой данных или Интернет-сервером? Ответ: никак. Никакого программирования далее вообще не потребуется. В комплекс программирования ПЛК входит OPC-сервер. Он умеет получать доступ к данным ПЛК также прозрачно, как и отладчик. Достаточно обеспечить канал передачи данных ПЛК — OPC-сервер. Обычно такой канал уже существует, вы использовали его при отладке. Вся дальнейшая работа сводится к определению списка доступных переменных, правильной настройке сети, конфигурированию OPC-сервера и SCADA-системы. В целом, операция очень напоминает настройку общедоступных устройств локальной сети ПК.

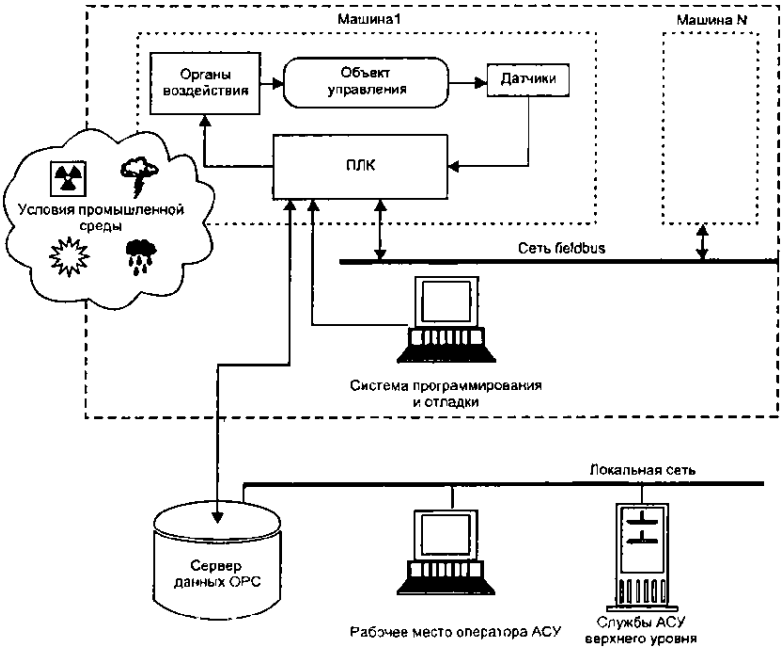


Рисунок 2 - Место ПЛК в АСУ ТП

Второй часто возникающей задачей является интеграция нескольких ПЛК с целью синхронизации их работы. Здесь появляются сети, обладающие рядом специфических требований.

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

Благодаря продуктивному развитию средств сетевой интеграции появилась возможность создания распределенных систем управления. В 80-х гг. XX в. доминировали ПЛК с числом входов-выходов несколько сотен. В настоящее время большим спросом пользуются микро ПЛК с количеством входов-выходов до 64. В распределенных системах каждый ПЛК решает локальную задачу. Задача синхронизации управления выполняется компьютерами

среднего звена АСУ. Распределенные системы выигрывают по надежности, гибкости монтажа и простоте обслуживания.

Доступность программирования

Главным требованием к ПЛК всегда была и остается возможность его эксплуатации существующим техническим персоналом и возможность быстрой замены старого оборудования. Поэтому языки программирования компьютеров и встраиваемых микропроцессорных систем управления плохо подходят для программирования ПЛК. Здесь нужны более простые и наглядные языки, позволяющие излагать задачу в близких к применяемым технологиям категориях. Привлечение же к программированию специализированной фирмы неизбежно порождает зависимость, если реализация не является достаточно прозрачной. Сложный язык программирования ПЛК снижает его шансы на конкурентном рынке существенно больше, чем массогабаритные показатели.

Кроме тех схемотехнических решений, которые принято считать «настоящими ПЛК», существуют также и другие, в частности:

- Программируемое (интеллектуальные) реле
- ПЛК на базе универсальных микропроцессоров
- Программный ПЛК

Программные приложения, имитирующие технологию ПЛК на компьютере (оснащенном платами ввода-вывода), получили название программный ПЛК (soft PLC). Программная эмуляция ПЛК удобна тем, что благодаря наличию многозадачной операционной системы можно совместить в одном месте контроллер, среду программирования и систему диспетчерского управления. Существенный минус такого решения — большое время выхода на рабочий режим после включения питания или зависания компьютера. Особенно опасно, если перезапуск произвел «сторожевой таймер» в автоматическом режиме, в то время как состояние исполнительных механизмов не соответствует исходным позициям.

Загрузка операционной системы может отнимать несколько минут, все это время система оказывается неуправляемой. Для ПЛК время «холодного» запуска измеряется миллисекундами.

Для достижения сравнимых с ПЛК технических показателей по надежности компьютер, конечно, должен быть промышленного исполнения (на базе магистралей PC/104 или VME), а не дешевый офисный.

Рабочий цикл

Задачи управления требуют непрерывного циклического контроля. В любых цифровых устройствах непрерывность достигается за счет применения дискретных алгоритмов, повторяющихся через достаточно малые промежутки времени. Таким образом, вычисления в ПЛК всегда повторяются циклически. Одна итерация, включающая замер, обсчет и выработку воздействия, называется рабочим циклом ПЛК. Выполняемые действия зависят от значения входов контроллера, предыдущего состояния и определяются пользовательской программой.

По включению питания ПЛК выполняет самотестирование и настройку аппаратных ресурсов, очистку оперативной памяти данных (ОЗУ), контроль целостности прикладной программы пользователя. Если прикладная программа сохранена в памяти, ПЛК переходит к основной работе, которая состоит из постоянного повторения последовательности действий, входящих в рабочий цикл.

Рабочий цикл ПЛК состоит из нескольких фаз.

1. Начало цикла.
2. Чтение состояния входов.
3. Выполнение кода программы пользователя.
4. Запись состояния выходов.
5. Обслуживание аппаратных ресурсов ПЛК.
6. Монитор системы исполнения.
7. Контроль времени цикла.
8. Переход на начало цикла.

В самом начале цикла ПЛК производит физическое чтение входов. Считанные значения размещаются в области памяти входов. Таким образом, создается полная одномоментная зеркальная копия значений входов.

Далее выполняется код пользовательской программы. Пользовательская программа работает с копией значений входов и выходов, размещенной в оперативной памяти. Если прикладная программа не загружена или остановлена, то данная фаза рабочего цикла, естественно, не выполняется. Отладчик системы программирования имеет доступ к образу входов-выходов, что позволяет управлять выходами вручную и проводить исследования работы датчиков.

После выполнения пользовательского кода физические выходы ПЛК приводятся в соответствие с расчетными значениями

(фаза 4).

Обслуживание аппаратных ресурсов подразумевает обеспечение работы системных таймеров, часов реального времени, оперативное самотестирование, индикацию состояния и другие аппаратно-зависимые задачи.

Монитор системы исполнения включает большое число функций, необходимых при отладке программы и обеспечении взаимодействия с системой программирования, сервером данных и сетью. В функции системы исполнения обычно включается: загрузка кода программы в оперативную и электрически перепрограммируемую память, управление последовательностью выполнения задач, отображение процесса выполнения программ, пошаговое выполнение, обеспечение просмотра и редактирования значений переменных, фиксация и трассировка значений переменных, контроль времени цикла и т. д.

Пользовательская программа работает только с мгновенной копией входов. Таким образом, значения входов в процессе выполнения пользовательской программы не изменяются в пределах одного рабочего цикла. Это фундаментальный принцип построения ПЛК сканирующего типа. Такой подход исключает неоднозначность алгоритма обработки данных в различных его ветвях. Кроме того, чтение копии значения входа из ОЗУ выполняется значительно быстрее, чем прямое чтение входа. Аппаратно чтение входа может быть связано с формированием определенных временных интервалов, передачей последовательности команд для конкретной микросхемы или даже запросом по сети.

Если заглянуть глубже, то нужно отметить, что не всегда работа по чтению входов полностью локализована в фазе чтения входов. Например, АЦП обычно требуют определенного времени с момента запуска до считывания измеренного значения. Часть работы системное программное обеспечение контроллера выполняет по прерываниям. Грамотно реализованная система исполнения нигде и никогда не использует пустые циклы ожидания готовности аппаратуры. Для прикладного программиста все эти детали не важны. Существенно только то, что значения входов обновляются автоматически исключительно в начале каждого рабочего цикла.

Общая продолжительность рабочего цикла ПЛК называется временем сканирования. Время сканирования в значительной степени определяется длительностью фазы кода пользовательской программы. Время, занимаемое прочими фазами рабочего цикла, практически является величиной постоянной. Для задачи среднего объема в ПЛК с системой исполнения CoDeSys время

распределится примерно так: 98% — пользовательская программа, 2% — все остальное.

Существуют задачи, в которых плавающее время цикла существенно влияет на результат, например, это автоматическое регулирование. Для устранения этой проблемы в развитых ПЛК предусмотрен контроль времени цикла. Если отдельные ветви кода управляющей программы выполняются слишком быстро, в рабочий цикл добавляется искусственная задержка. Если контроль времени цикла не предусмотрен, подобные задачи приходится решать исключительно по таймерам.

Время реакции

Время реакции — это время с момента изменения состояния системы до момента выработки соответствующей реакции. Очевидно, для ПЛК время реакции зависит от распределения моментов возникновения события и начала фазы чтения входов. Если изменение значений входов произошло непосредственно перед фазой чтения входов, то время реакции будет наименьшим и равным времени сканирования. Худший случай, когда изменение значений входов происходит сразу после фазы чтения входов. Тогда время реакции будет наибольшим, равным удвоенному времени сканирования минус время одного чтения входов. Иными словами, время реакции ПЛК не превышает удвоенного времени сканирования.

Помимо времени реакции ПЛК, существенное значение имеет время реакции датчиков и исполнительных механизмов, которое также необходимо учитывать при оценке общего времени реакции системы.

Существуют ПЛК, которые реализуют команды непосредственного доступа к аппаратуре входов и выходов, что позволяет обрабатывать и формировать отдельные сигналы с длительностью меньшей длительности рабочего цикла.

Для уменьшения времени реакции сканирующих контроллеров алгоритм программы разбивается на несколько задач с различным периодом исполнения. В наиболее развитых системах пользователь имеет возможность создавать отдельные программы, исполняемые по прерыванию, помимо кода, исполняемого в рабочем цикле. Такая техника позволяет ПЛК существенно форсировать ограничение реакции временем сканирования при небольшом количестве входов, требующих сверхскоростной реакции.

Время цикла сканирования является базовым показателем быстродействия ПЛК. При измерении времени рабочего цикла пользовательская программа должна содержать тысячи логических команд. Для ПЛК, поддерживающих стандарт МЭК 61131-3, используют команды на языке IL. Иногда изготовители приводят несколько значений времени цикла, полученных при работе с переменными различной разрядности.

Ориентировочно о скорости обработки различных типов данных можно судить по тактовой частоте и разрядности центрального процессора. Хотя нет ничего удивительного в том, что восьмиразрядные ПЛК не редко оказываются быстрее 32-разрядных при выполнении битовых операций. Объясняется это тем, что в 8-разрядных микропроцессорах более распространена аппаратная поддержка работы с битами. Так, в PC-совместимых процессорах для выделения бита приходится использовать логические команды и циклический сдвиг.

Устройство ПЛК

Аппаратно ПЛК является вычислительной машиной. Поэтому архитектура его процессорного ядра практически не отличается от архитектуры компьютера. Отличия заключены в составе периферийного оборудования, отсутствуют видеокарта, средства ручного ввода и дисковая подсистема. Вместо них ПЛК имеет блоки входов и выходов.

Конструктивно контроллеры подразделяют на моноблочные, модульные и распределенные. Моноблочные, или одноплатные, ПЛК имеют фиксированный набор входов-выходов. В модульных контроллерах модули входов-выходов устанавливаются в разном составе и количестве в зависимости от требуемой конфигурации. Так достигается минимальная аппаратная избыточность. В распределенных системах модули или даже отдельные входы-выходы, образующие единую систему управления, могут быть разнесены на значительные расстояния.

Характерным для современных контроллеров является использование многопроцессорных решений. В этом случае модули ввода-вывода имеют собственные микропроцессоры, выполняющие необходимую предварительную обработку данных. Модуль центрального процессора имеет выделенную скоростную магистраль данных для работы с памятью и отдельную магистраль (сеть) для общения с модулями ввода-вывода.

Еще одним вариантом построения ПЛК является мезонинная технология. Все силовые цепи, устройства защиты контроллера

выполняются на несущей плате. Процессорное ядро контроллера, включающее систему исполнения, выполнено на отдельной сменной (мезонинной) плате. В результате появляется возможность составлять несколько комбинаций процессорного ядра и разных силовых плат без необходимости корректировки программного обеспечения. При необходимости процессор можно заменить даже в готовой системе.

Системное и прикладное программное обеспечение

Системное программное обеспечение (СПО) непосредственно контролирует аппаратные средства ПЛК. СПО отвечает за тестирование и индикацию работы памяти, источника питания, модулей ввода-вывода и интерфейсов, таймеров и часов реального времени. Система исполнения кода прикладной программы является составной частью СПО. Система исполнения включает драйверы модулей ввода-вывода, загрузчик кода программ пользователя, интерпретатор команд и отладочный монитор. Код СПО расположен в ПЗУ и может быть изменен только изготовителем ПЛК. Тем не менее, существует и ПЛК с открытой архитектурой, использующие распространённые ОС для встроенных приложений, например Linux, Windows Embedded и др.

Код прикладной программы размещается в энергонезависимой памяти, чаще всего это электрически перепрограммируемые микросхемы. Изменение кода прикладной программы выполняется пользователем ПЛК при помощи системы программирования и может быть выполнено многократно.

Контроль времени рабочего цикла

Правильно составленная пользовательская программа не должна содержать бесконечных циклов. В противном случае управление системе исполнения не будет передано, и, соответственно, нормальное функционирование контроллера будет нарушено. Для преодоления данной проблемы служит контроль времени цикла. Контроль осуществляется при поддержке аппаратно реализованного «сторожевого таймера». Если фаза пользовательского кода выполняется дольше установленного порога, то ее работа будет прервана. Таким образом, достигается предсказуемое поведение ПЛК при ошибках в программе и при «зависании» по причине аппаратных сбоев.

Обслуживание сторожевого таймера выполняется в рабочем цикле ПЛК. Выполнять эту операцию по прерыванию нельзя, поскольку при «зависании» процессора система прерываний достаточно часто продолжает исправно работать.

ПРАКТИЧЕСКИЕ РАБОТЫ

Подключение ПЛК ОВЕН к среде CoDeSys

1 Цель работы:

- создание проекта в среде CoDeSys с подключением ПЛК и его конфигурированием;
- изучение работы ПЛК на простейших примерах.

2 Краткая теория

Программируемый контроллер — это программно управляемый *дискретный автомат*, имеющий некоторое множество входов, подключенных посредством датчиков к объекту управления, и множество выходов, подключенных к исполнительным устройствам. ПЛК контролирует состояния входов и вырабатывает определенные последовательности программно заданных действий, отражающихся в изменении выходов.

ПЛК предназначен для работы в режиме реального времени в условиях промышленной среды и должен быть доступен для программирования неспециалистом в области информатики.

Большинство ПЛК имеют *дискретные входы,* т. е. входы, значения сигналов на которых способны принимать только два состояния — логического нуля и логической единицы. Датчиками, формирующими такой сигнал, являются кнопки ручного управления, концевые датчики, датчики движения, контактные термометры и др. *Дискретные выход* также имеет два состояния — включен и выключен. Сфера применения дискретных выходов очевидна: электромагнитные реле, силовые пускатели, электромагнитные клапаны, световые сигнализаторы и т. д.

Задачи управления требуют непрерывного циклического контроля, поэтому вычисления в ПЛК всегда повторяются циклически. Одна итерация, включающая замер, обсчет и выработку воздействия, называется *рабочим циклом* ПЛК. Выполняемые действия зависят от значения входов контроллера, предыдущего состояния и определяются пользовательской программой.

Подавляющее число ПЛК программируются на персональном компьютере в инструментальных средах, предлагаемых разработчиком и функционирующих в соответствии с международным стандартом МЭК 61131-3, который описывает языки программирования ПЛК. Главная задача инструментов комплекса программирования ПЛК состоит в автоматизации работы разработчика.

Комплекс CoDeSys разработан фирмой 3S (Smart Software Solutions). Это универсальный инструмент программирования контроллеров и встраиваемых систем на языках МЭК 61131-3, не привязанный, к какой-либо аппаратной платформе и удовлетворяющий современным требованиям быстрой разработки программного обеспечения.

Базовый состав комплекса программирования ПЛК состоит из двух обязательных частей: системы исполнения и рабочего места программиста. Система исполнения функционирует в контроллере и, кроме непосредственно исполнения управляющей программы, обеспечивает загрузку кода прикладной программы и отладочные функции. В простейшем случае ПЛК подключается к компьютеру через стандартный COM-порт (RS232) нуль-модемным кабелем. В условиях цеха может использоваться более помехоустойчивый и обеспечивающий большие расстояния передачи данных интерфейс (RS485 или токовая петля). В комплексе CoDeSys посредником между средой разработки и ПЛК служит специальное приложение — шлюз связи (gateway). Шлюз связи взаимодействует с интегрированной средой через Windows сокет-соединение, построенное на основе протокола TCP/IP. Благодаря этому программист может абсолютно полноценно работать на удаленном компьютере. Причем удаленность не ограничивается рамками локальной сети. ПК, выполняющий задачу шлюза связи, может одновременно взаимодействовать с ПК программиста через Интернет и с ПЛК через модемное соединение.

Контроллер с точки зрения МЭК программы имеет несколько областей памяти, имеющих разное назначение: 1) Область входов ПЛК; 2) Область выходов ПЛК; 3) Область прямо адресуемой памяти; 4) Оперативная память пользователя (ОЗУ). Привязка к конкретным адресам задается при помощи прямой адресации. Для создания *прямо адресуемой* переменной используется следующее объявление:

имя переменной *AT%* прямой адрес тип.

В прямом адресе указывается *номер элемента*. Это коренным образом отличается от физических адресов микропроцессора. Входы ПЛК — это переменные с прямыми адресами в области I. Они доступны в прикладных программах только по чтению. Выходы Q — только по записи. Прямые адреса можно использовать в программах непосредственно либо заменять компактным обозначением с созданием соответствующей глобальной

3 Задание

3.1 Установка Target-файлов.

Перед тем как начать работу в среде CoDeSys следует выполнить инсталляцию Target-файлов. В Target-файлах содержится информация о ресурсах программируемых контроллеров, с которыми работает CoDeSys. Target-файл поставляется производителем контроллера. Например, для контроллера ПЛК150-220.UL фирмы ОВЕН необходимо устанавливать файл PLC150.UL.tnf. Инсталляция Target-файлов производится при помощи утилиты InstallTarget, поставляемой вместе со средой программирования.

Порядок инсталляции Target-файлов:

1) Запустить утилиту InstallTarget из меню Пуск > Все программы > 3S Software > CoDeSys V2.3 > InstallTarget.

2) В открывшемся при запуске окне (рис. 1) – нажать кнопку Open и указать путь доступа к инсталлируемому Target-файлу (имеющему расширение *.tnf, Target Information File).

3) После открытия требуемого файла в области «Possible Targets» окна отобразится папка «Owen».

4) Открыв папку «Owen» и выделив находящуюся там строку, нажать кнопку Install. В области «Installed Targets» окна отобразится список инсталлированных Targett-файлов.

3.2 Создание проекта. Выбор контроллера.

1) Запустить среду CoDeSys из меню Пуск > Все программы > 3S Software > CoDeSys V2.3 > CoDeSys V2.3.

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

3) После создания проекта нужно выбрать Targett-файл, соответствующий названию контроллера. Targett-файл предварительно должен быть инсталлирован (см.п.3.1). Окно выбора Targett-файла представлено на рис.2. Затем откроется окно настроек Targett-файлов. Как правило, настройки установлены производителем и не требуют изменения (кроме изменения объема Retaill-памяти).

4) После подтверждения настроек Targett-файла необходимо создать основной POU (главную программу проекта). Окно этого диалога представлено на рис. 3. Главная программа всегда должна иметь тип Program и имя PLC_PRG. Поэтому в данном диалоге необходимо выбрать только язык программирования (Language of the POU).

5) В зависимости от выбранного языка программирования откроется окно, в котором необходимо создать программу, испол-

няемую на контроллере.

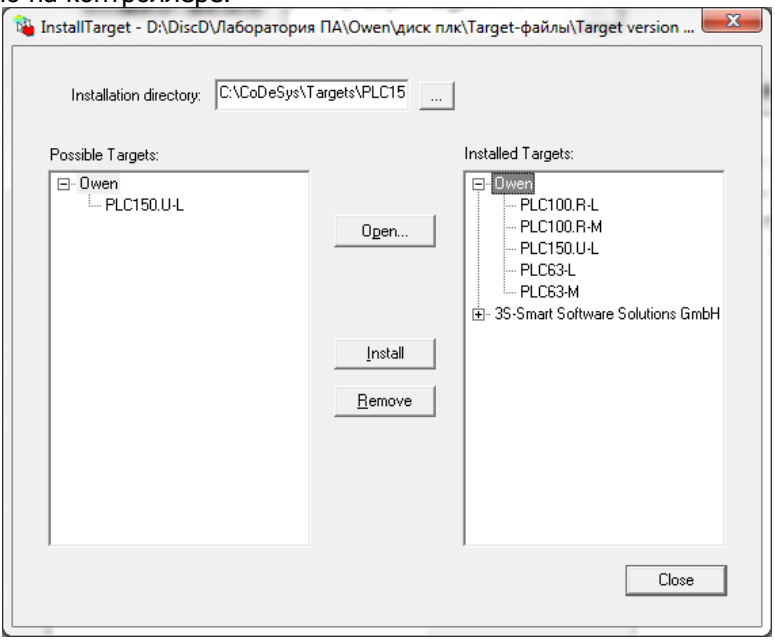


Рисунок 1 – Окно «InstallTarget» утилиты InstallTarget

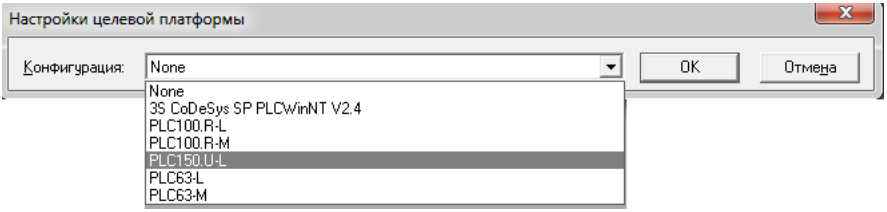


Рисунок 2 – Окно выбора Target-файла

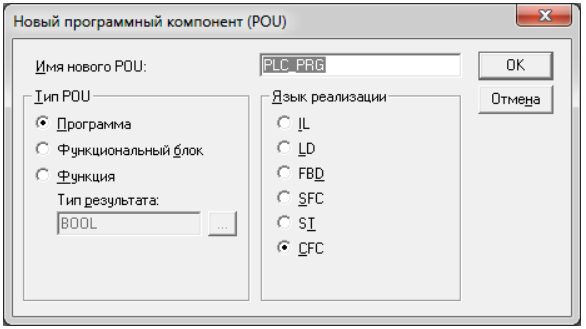


Рисунок 3 – Окно создания основного POU

3.3 Установка связи и конфигурирование контроллера.

Установка связи с контроллером возможна по интерфейсам Ethernet, Debug RS-232 или через последовательный модем (подключенный к порту Debug RS-232). Настройка канала соединения с контроллером производится в окне «Communication parameters», вызываемом командой меню Online > Communication parameters в среде CoDeSys (рис. 4). **Далее необходимо:**

1) Нажать кнопку New в этом окне. Откроется окно «Communication parameters: New Channel». В этом окне задать имя нового соединения (например, Owen) и выбрать из перечня интерфейс соединения: Tcp/Ip (Level 2) для связи по интерфейсу Ethernet, Serial (RS232) для связи через порт Debug RS횛232 напрямую или Serial (Modem) для связи через последовательный модем.

2) Для установки соединения по интерфейсу Ethernet контроллер и компьютер должны находится в одной IP-подсети. Возможны два варианта: изменение имеющегося IP-адреса контроллера в соответствии с настройками сети пользователя или задание компьютеру дополнительного IP-адреса, входящего в подсеть контроллера. Изменение IP-адреса контроллера возможно при помощи команды SetIP, подаваемой через PLC-Browser. При этом связь с контроллером должна быть установлена через интерфейс Debug RS-232. Задание дополнительного IP-адреса компьютеру происходит в свойствах протокола TCP/IP в настройках сетевого окружения Windows.

При изготовлении устанавливается IP-адрес контроллера **10.0.6.10**. Поэтому необходимо присвоить компьютеру дополнительный IP-адрес в подсети **10.0.6**, отличный от адреса 10.0.6.10. Маску подсети задать равной **255.255.0.0.**

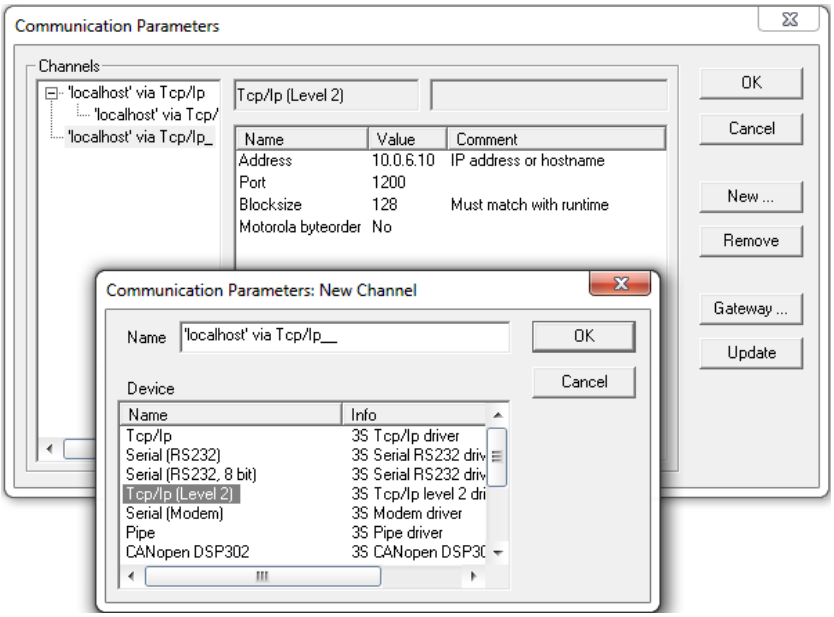


Рисунок 4 – Настройка коммуникационных параметров для соединения с ПЛК

3) После настройки соединения подать команду меню Online > Login, устанавливающую связь с контроллером. При этом флаг перед строкой меню Simulation Mode должен быть снят. Для установки связи необходимо, чтобы программа была создана.

3.4 Создание программы пользователя

В качестве языка программирования будем использовать не описанный в стандарте, но наиболее простой для освоения и входящий в состав CoDeSys язык CFC. По структуре данный язык схож с FBD и позволяет осуществлять программирование при помощи функциональных блоков.

Прежде чем приступить к написанию программы, осуществим конфигурирование дискретных входов и выходов ПЛК, для чего перейдем на вкладку структуры проекта Ресурсы > Конфигурация ПЛК (см. рис. 5.). В поле непосредственного адреса области входов вместо непосредственного адреса (двойной щелчок на префиксе AT) необходимо задать удобные для использования в программе имена входов (глобальные переменные). Например, для области входов: in0, in1…in5, а для области выходов – out0…out3.

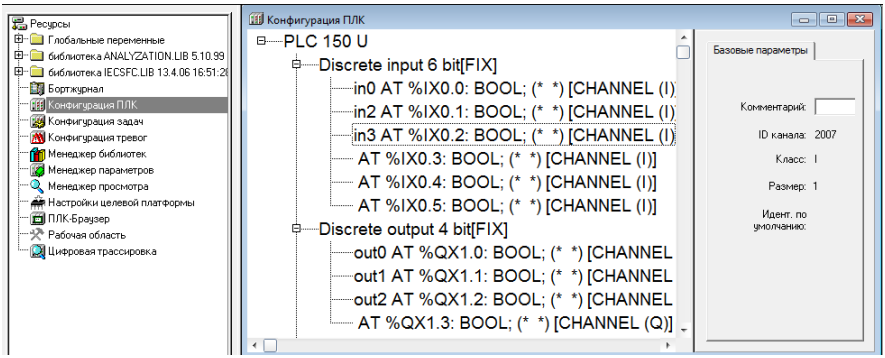


Рисунок 5 – Окно конфигурации ПЛК

Создадим далее простейшую программу, позволяющую передавать значение дискретного входа ПЛК in0 на дискретный выход out0. Для этого разместим в рабочем поле элементы вход и выход, расположенные на панели инструментов и соединим их при помощи левой кнопки мыши (см. рис. 6).

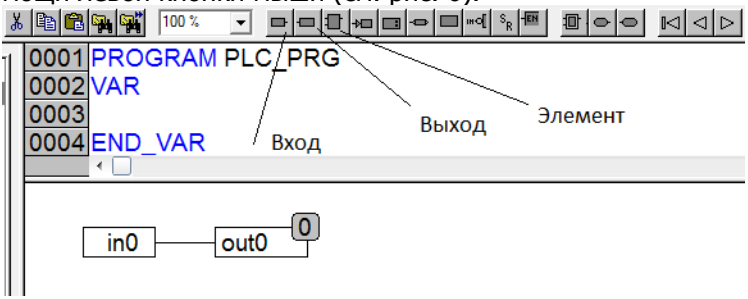


Рисунок 6 – Программа соединения дискретного входа и выхода ПЛК

Установим связь с контроллером, подадим на дискретный вход значение (при помощи кнопочного переключателя) логической единицы (либо нуля) и будем наблюдать аналогичное изменение значения сигнала на дискретном выходе. Для запуска программы необходимо нажать кнопку Старт на самом ПЛК, либо выбрать пункт меню Онлайн > Старт.

Для размещения функционального блока, реализующего различные функции в программе необходимо использовать Элемент на панели инструментов (см. рис. 6). Изменение типа элемента можно произвести, непосредственно задав его имя, либо нажав клавишу F2 на клавиатуре и выбрав его из списка доступных функциональных блоков (см. рис. 7).

Подключим в качестве примера и разместим в рабочем

поле элемент логическое И (AND) и подсоединим к нему входы ПЛК in0 и in1 и выход out0. Будем наблюдать изменение выходного сигнала.

При размещении функциональных блоков необходимо обращать внимание на их нумерацию, указываемую в правом верхнем углу. При этом блок, выполняемый в программе первым, имеет меньший номер. Для упорядочивания и задания номеров блоков используется пункт контекстного меню (щелчок правой кнопкой по блоку) Порядок.

Произведем инверсию входных сигналов при помощи щелчка правой кнопкой мыши на соответствующих входах элемента AND и выбрав пункт Инверсия. Пронаблюдаем за изменением выходного сигнала при изменении входных сигналов.

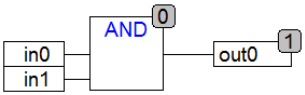


Рисунок 7 – Реализация логической функции И

3.5 Задания

1) Написать программу реализующую дешифратор 2x4 (дискретные входы in0, in1 и выходы out0-out3);

2) написать программу, переводящую дискретный выход out0 в состояние логической единицы при поступлении на вход in0 переднего фронта (переход и з 1 в 0) сигнала 10 раз. Для реализации программы использовать функциональный блок инкрементного счетчика CTU.

4 Контрольные вопросы:

1 Для чего в первую очередь предназначены ПЛК? Что такое режим реального времени? Рабочий цикл ПЛК?

2 Опишите предназначение стандарта МЭК 61131. Какие языки используются для программирования ПЛК? В чем их основное отличие?

3 Каково назначения комплекса 3S CoDeSys? Как осуществляется связь среды разработки и ПЛК?

4 Что представляет собой ПЛК с точки зрения стандарта МЭК?

5 Содержание отчета:

1 Цель работы.

2 Результаты выполнения задания (программы с комментариями).

3 Ответы на контрольные вопросы.

Изучение работы аналоговых входов ОВЕН ПЛК в среде CoDeSys

1 Цель работы:

- изучение работы с аналоговыми входами ПЛК на примере измерения реальных физических величин;

- изучение работы аналоговых выходов ПЛК в режиме ШИМ.

2 Краткая теория

Модули аналоговых входов позволяют измерять при помощи ПЛК различные непрерывно изменяющиеся во времени физические величины. В случае конфигурирования аналогового входа для измерения унифицированного сигнала – это напряжение или ток, в случае подключения первичных преобразователей (датчиков) – соответствующая физическая величина (температура, давление, расход и т.д.). В зависимости от конкретной модели ПЛК количество аналоговых входов может быть различным, а также различно их назначение. Так, для ОВЕН ПЛК возможны четыре стандартных варианта подключения аналоговых входов: датчик унифицированного сигнала, датчик типа термопара, датчик типа термосопротивление (терморезистор) и контактный датчик.

Не подвергнутые обработке сигналы от датчиков весьма разнообразны и диапазон их изменения составляет от нескольких милливольт (для термопары) до сотен вольт для тахогенератора. Они могут быть вызваны изменениями напряжения постоянного тока, переменного тока или сопротивления. Поэтому очевидно, что необходимо использовать некоторую стандартизацию сигналов, которую выполняют специальные нормирующие преобразователи, как правило, встроенные в датчик, либо подключаемые отдельно. После этого стандартизированный сигнал, несущий информацию об измеряемой величине, может быть подан на обычный аналоговый вход.

Каким должен быть стандартизированный сигнал? Самый распространенный стандарт представляет аналоговый сигнал в виде тока с диапазоном изменения 4—20 мА, где 4мА соответствует минимальному уровню сигнала, а 20 мА — максимальному. Его широкое распространение объясняется следующими причинами:

-на передачу токовых сигналов не оказывает влияние сопротивление соединительных проводов, поэтому требования к диаметру и длине соединительных проводов (а значит, и к стоимости) снижаются;

-токовый сигнал работает на низкоомную (по сравнению с сопротивлением источника сигнала) нагрузку, поэтому наведенные электромагнитные помехи в токовых цепях малы по сравнению с аналогичными цепями, в которых используются сигналы напряжения;

-обрыв линии передачи токового сигнала 4…20 мА однозначно определяется измерительными системами по нулевому уровню тока в цепи (в нормальных условиях он должен быть не меньше 4 мА);

-токовый сигнал позволяет не только передавать полезный информационный сигнал, но и обеспечивать электропитание самого нормирующего преобразователя – минимально допустимого уровня 4 мА достаточно для питания современных электронных устройств.

Унифицированные сигналы применяются не только для связи с первичными датчиками, но и для связи между собой других устройств промышленной автоматики: регистраторов, регуляторов, контроллеров и исполнительных устройств. Применение унифицированных сигналов регламентировано ГОСТ 26.011-80. Стандарт устанавливает допустимые диапазоны унифицированных сигналов, а также вводит ограничения на величину сопротивления источников и приемников этих сигналов. В ряду унифицированных сигналов есть сигналы напряжения 0…1, 0…10 В и сигналы тока 0…5, 0…20, 4…20 мА.

Термосопротивлением называется параметрический измерительный преобразователь, активное сопротивление которого изменяется при изменении температуры. Термоспротиивления бывают металлические (терморезисторы) и полупроводниковые(термисторы). Сопротивление терморезисторов при 0°С равно 10, 50, 100 Ом, что отражается в их обозначении: ТСМ10, ТСМ50, ТСМ100 (термосопротивление медное). Медные терморезисторы имеют практически линейную зависимость сопротивления R от температуры t: $R=R0*(1+\alpha*t)$, где R0 - сопротивление преобразователя при 0°С, α - температурный коэффициент (для меди равный $4{,}28*10^{-3}$ 1/°С).

Обычно при измерении температуры с помощью термосопротивления на его чувствительный элемент (ЧЭ) подают стабилизированный ток возбуждения. В результате на датчике возникает разность потенциалов, пропорциональная сопротивлению, а значит, и измеряемой температуре. Поскольку ЧЭ имеют малое номинальное сопротивление, сравнимое с сопротивлением подводящих проводов, то должны быть приняты меры по устранению

влияния сопротивления подводящих проводов на измерение температуры. Эффективность мер определяется методом измерения и способом подключения ко вторичному прибору. Основных схем подключения три: двухпроводная; трехпроводная; четырехпроводная.

В простейшей двухпроводной схеме влияние сопротивления подводящих проводов не устраняется. Напряжение измеряется не только на ЧЭ, но и на соединительных проводах.

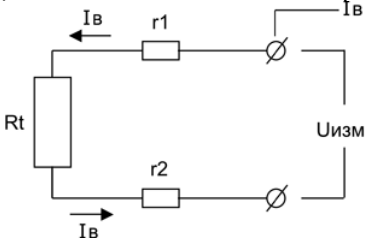


Рисунок 1 – Измерение температуры при помощи двухпроводной схемы

$$U_t = U_{изм} - U_{r1} - U_{r2}$$

Двухпроводная схема может быть использована в случае, если сопротивлением подводящих проводов (r1,r2) можно пренебречь по сравнению с Rt. Влияние сопротивления соединительных проводов в трехпроводной схеме устраняется путем компенсации. Компенсация возможна, если соединительные провода одинаковы. В этом случае появляется возможность выделить отдельно напряжение на соединительных проводах и скомпенсировать его.

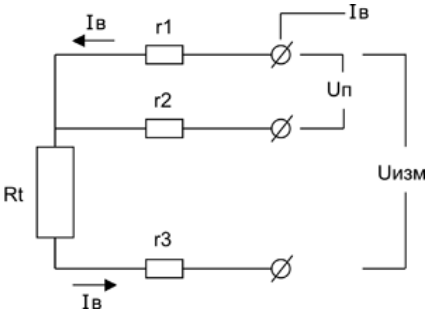
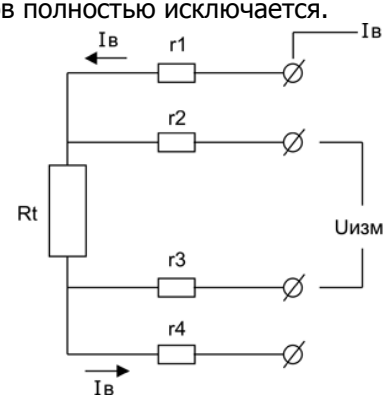


Рисунок 2 – Измерение температуры при помощи трехпроводной схемы

$$U_t = U_{изм} - U_{r1} - U_{r2} \approx U_{изм} - 2U_n$$

Равенство сопротивлений соединительных проводов и их

температурных зависимостей является основным условием применимости трехпроводной схемы. В четырехпроводной схеме питание ЧЭ током возбуждения производится с помощью одних проводов, а измерение разности потенциалов на ЧЭ – с помощью других. Если измерение напряжения производится высокоомным вольтметром (ток через r2 и r3 не течет), то влияние сопротивления всех проводов полностью исключается.



$U_t = U_{изм}$

Рисунок 3 – Измерение температуры при помощи четырехпроводной схемы

Погрешности измерения, которые возникают за счет влияния соединительных проводов, для различных схем приведены в таблице 1.

Таблица 1 – Вычисление погрешности при влиянии сопротивления соединительных проводов

Схема подключения	**Дополнительная погрешность**	**Примечание**
Двухпроводная	(r1+r2)/Rt	
Трехпроводная	Δr/Rt	Δr - разность сопротивлений соединительных проводов
Четырехпроводная	(r2+r3)/Rвх	Rвх – входное сопротивление вольтметра

Действующий стандарт на технические требования к терморезисторам: ГОСТ Р 8.625-2006 (Термометры сопротивления из

платины, меди и никеля. Общие технические требования и методы испытаний). Стандарт соответствует международному стандарту МЭК 60751 (2008).

Термопара (термоэлектрический измерительный преобразователь) представляет собой электрическую цепь, состоящую из двух различных проводников, соединенных между собой в двух точках (спаев) – свободный (холодный) спай и рабочий (горячий) спай. Если температуру одного спая сделать отличной от температуры другого, то в цепи потечет ток под действием ЭДС, называемой термоэлектродвижущей силой (термо ЭДС), которая в первом приближении пропорциональна разности температур. Зависимость температуры горячего спая от термо ЭДС при температуре холодного спая равной 0°С называют номинальной статической характеристикой (НСХ) термопары. Технические требования к термопарам определяются ГОСТ 6616-94. Стандартные таблицы для термоэлектрических термометров, классы допуска и диапазоны измерений приведены в стандарте МЭК 60584-1,2 и в ГОСТ Р 8.585-2001. В зависимости от типа НСХ термопары подразделяются на: платинородий-платиновые — ТПП13 — Тип R; платинородий-платиновые — ТПП10 — Тип S; платинородий-платинородиевые — ТПР — Тип B; железо-константановые (железо-медьникелевые) ТЖК — Тип J; медь-константановые (медь-медьникелевые) ТМКн — Тип T; нихросил-нисиловые (никель-хромникель-никелькремниевые) ТНН — Тип N; хромель-алюмелевые — ТХА — Тип K; хромель-константановые ТХКн — Тип E; хромель-копелевые — ТХК — Тип L; медь-копелевые — ТМК — Тип M; сильх-силиновые — ТСС — Тип I; вольфрам и рений — вольфрамрениевые — ТВР — Тип А-1, А-2, А-3.

Точный состав сплава термоэлектродов для термопар из неблагородных металлов в МЭК 60584-1 не приводится. НСХ для хромель-копелевых термопар ТХК и вольфрам-рениевых термопар определены только в ГОСТ Р 8.585-2001. В стандарте МЭК данные термопары отсутствуют. Тип L установлен только в немецком стандарте DIN 43710 и стандартные таблицы отличаются от таблиц для термопар ТХК.

Наиболее часто сигналы с датчиков (аналоговые входы ПЛК) используются в качестве уставки для регуляторов различных физических величин.

Регулятор— устройство, предназначенное для поддержания контролируемой величины на заданном уровне.

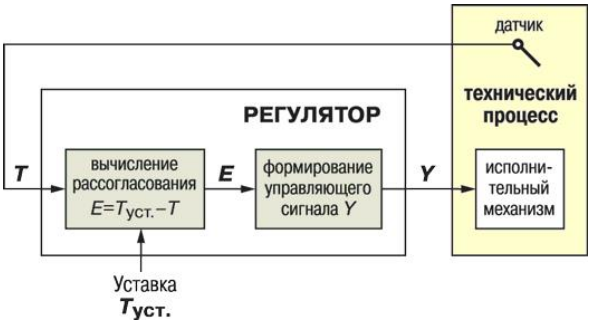


Рисунок 4 – Схема работы регулятора аналогового типа

В режиме аналогового регулирования ПЛК рассчитывает отклонение E текущего значения контролируемой величины T от заданной уставки $T_{уст.}$ (т. е. рассогласование). В результате на выходе регулятора вырабатывается аналоговый сигнал Y, который направлен на уменьшение рассогласования E. Этот сигнал подается на исполнительное устройство регулятора в виде тока или последовательности импульсов (ШИМ).

Широтно-Импульсная Модуляция (ШИМ) - это способ задания аналогового сигнала путём изменения ширины (длительности) прямоугольных импульсов.

В выходном устройстве регулятора дискретного типа (реле, транзисторная или симисторная оптопара, выход для управления твердотельным реле), выходной сигнал преобразуется в последовательность управляющих импульсов с длительностью D (см. рисунок 5):

$$D = Y*(T_{сл}/100\%)$$

D – длительность импульса, с; $T_{сл}$ – период следования импульсов, с (задается пользователем при программировании); Y – выходной сигнал регулятора.

Если в качестве выходного устройства используется ЦАП, выходной сигнал преобразуется в пропорциональный ему ток 4...20 мА или напряжение 0...10 В.

3 Задание

3.1 Подключение датчика типа термосопротивление

Датчики типа ТСМ50, ТСМ100 и др. имеют относительно небольшое сопротивление, сравнимое с сопротивлением соединительных проводов. Из-за этого вносится большая дополнительная погрешность. Обычно подключение таких термосопротивлений

осуществляется по трехпроводной схеме, но используемый в работе ОВЕН ПЛК150 не имеет такой возможности (датчики подключаются по двухпроводной схеме). Поэтому перед их использованием необходимо выполнить следующие действия:

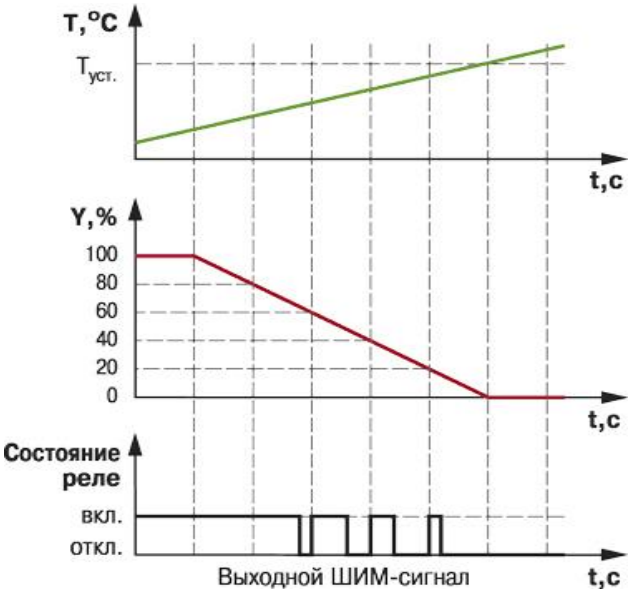


Рисунок 5 - Принцип ШИМ

1 Подключить датчик к ПЛК по двухпроводной схеме.

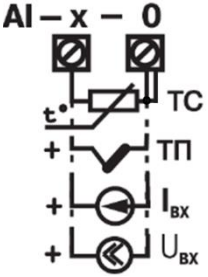


Рисунок 6 – Схема подключения датчиков к аналоговым входам ПЛК
ТС-термосопротивление; ТП-термопара.

2 Создать новый проект в среде CoDeSys для ПЛК150.

3 Настроить контроллер на работу с датчиком требуемого типа.

Для этого перейдем на вкладку структуры проекта Ресурсы > Конфигурация ПЛК и при помощи щелчка правой кнопкой мыши на элементе Unified signal sensor укажем тип датчика RTD sensor (см. рис. 7.)

Для датчика, используемого в работе, температурный коэффициент равен $4{,}28*10^{-3}$ 1/°C, а номинальное сопротивление датчика – 50 Ом. Поэтому в Параметрах модуля для параметра Type of sensor необходимо выбрать r428_50 (см. рис.8).

4 Со стороны датчика к линии связи подключить магазин сопротивлений с классом точности не менее 0,1 или эталонный резистор.

5 Установить на магазине сопротивлений значение 50 Ом (для датчика ТСМ50).

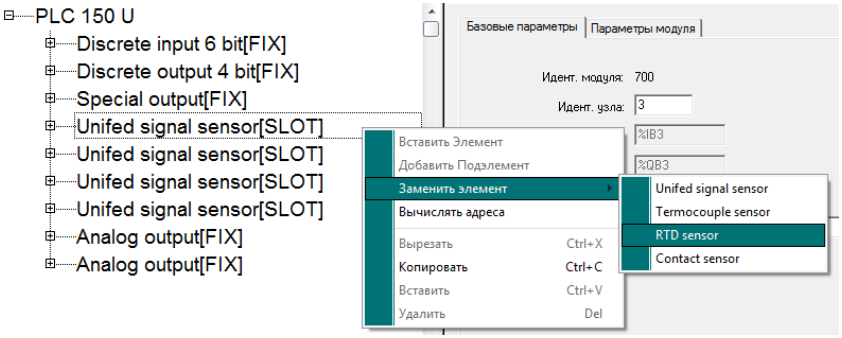


Рисунок 7 – Выбор типа сигнала для аналогового входа

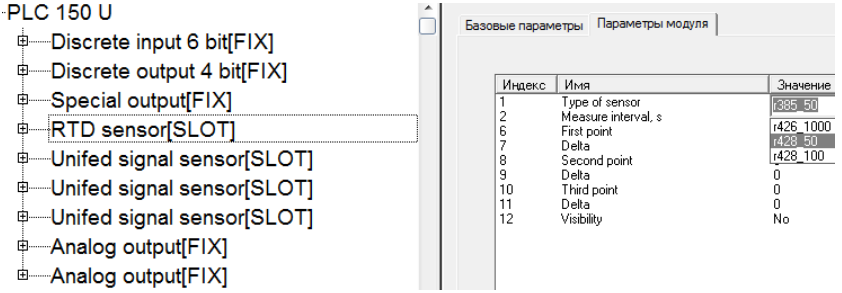


Рисунок 8 – Выбор типа датчика

6 Зафиксировать результаты измерения на аналоговом входе контроллера (Результаты измерения можно вызвать на вкладке RTD Sensor (параметр типа REAL) при установке курсора на канале модуля (см. рис. 9).

7 В параметре First point модуля аналогового входа задать

значение 0 (что соответствует значению 0°С) и в первом параметре Delta задать значение, равное измеренному, но с обратным знаком.

8 Повторно зафиксировать результаты измерения на аналоговом входе контроллера и удостовериться, что они равны или близки к 0.

9 Подсоединить вместо магазина сопротивлений датчик.

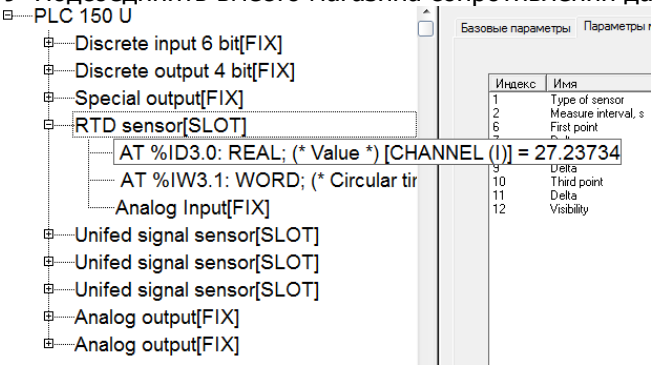


Рисунок 9 – Чтение значение сигнала датчика температуры

3.2 Подключение датчика температуры типа термопара

Калибровка термопары (компенсация холодного спая) и аналоговых входов ПЛК процесс более сложный, чем калибровка термосопротивления и зависит от типа термопары (калибровочной характеристики), типа измерительного преобразователя, условий окружающей среды и др. Как правило, для калибровки термопар с таким сложным устройством как ПЛК поставляются специальные библиотеки для калибровки. Калибровку необходимо производить в условиях, приближенных к условиям производственной среды (с установкой ПЛК в щит управления, а датчиков на предполагаемых точках измерения).

Для наблюдения результатов измерения температуры при помощи датчика типа термопара произведем следующие действия:

1 Настроим контроллер на работу с датчиком требуемого типа.

Для этого перейдем на вкладку структуры проекта Ресурсы> Конфигурация ПЛК и при помощи щелчка правой кнопкой мыши на элементе Unified signal sensor укажем тип датчика Thermocouple sensor

Датчик, подключенный к аналоговому входу ПЛК, облада-

ет характеристикой типа L, поэтому в Параметрах модуля для параметра Type of sensor необходимо установить пункт TP_L.

2 Зафиксировать результаты измерения на аналоговом входе контроллера (параметр типа REAL при установке курсора на канале модуля).

3.3. Подключение источника унифицированного сигнала

Источники унифицированного сигнала позволяют принимать по аналоговым входам ПЛК сигналы от различных датчиков, имеющих нормирующие преобразователи. Для подключения источника постоянного напряжения 0..10 В необходимо:

1 Перейти на вкладку структуры проекта Ресурсы> Конфигурация ПЛК и для элемента Unified signal sensor в поле типа датчика (Type of sensor) установить пункт U0_10.

2 Зафиксировать результаты измерения на аналоговом входе контроллера при изменении напряжения от источника (параметр типа REAL при установке курсора на канале модуля).

3.4 Регулирование частоты и скважности выходного сигнала по методу ШИМ

3.4.1 Использование блока «генератор прямоугольных импульсов»

Осуществить выдачу последовательности импульсов по методу ШИМ с регулированием длительности импульса и паузы в прямо пропорциональной зависимости от напряжения на аналоговом входе (уставки).

Один из способов формирования импульсной ШИМ – последовательности - использование входящего в состав дополнительной библиотеки Utill.lib блока BLINK. Для подключения библиотеки необходимо перейти на вкладку структуры проекта Ресурсы> Менеджер библиотек и из контекстного меню (см. рис. 10) добавить соответствующую библиотеку.

Для выполнения задания использовать следующие функциональные блоки (предварительно изучив принцип их работы, воспользовавшись справочной системой): BLINK, LIMIT, MUL, REAL_TO_TIME (см. рис. 11).

Для добавления экземпляра блока, отличного от размещаемого по умолчанию блока AND необходимо изменить его название на соответствующее новому блоку либо нажать клавишу F2 (при установке курсора на название блока) и выбрать блок из списка доступных. Для каждого блока, не входящего в стандартную библиотеку необходимо задавать имя экземпляра латинскими символами и цифрами без пробелов (см. рис.12).

При задании длительности импульсов необходимо помнить, что единицей измерения времени для формата **TIME** являются **миллисекунды**. Период следования импульсов ограничить в пределах от 1 до 3 с.

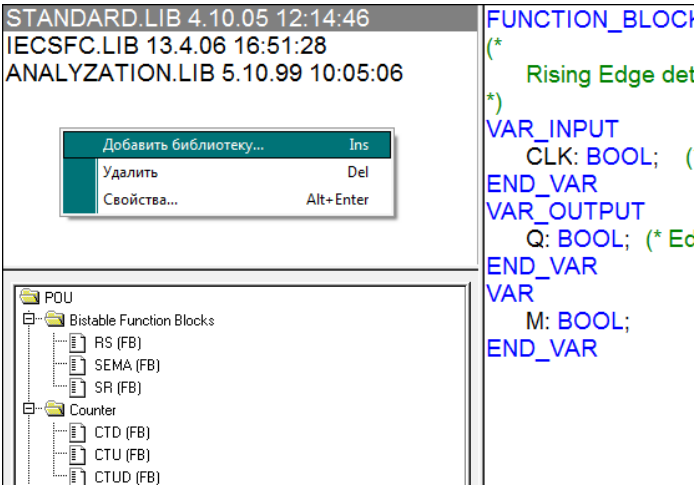


Рисунок 10 – Добавление новой библиотеки

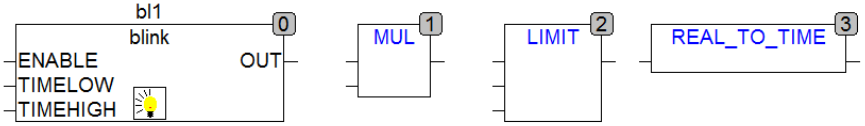


Рисунок 11 – Функциональные блоки, используемые в программе

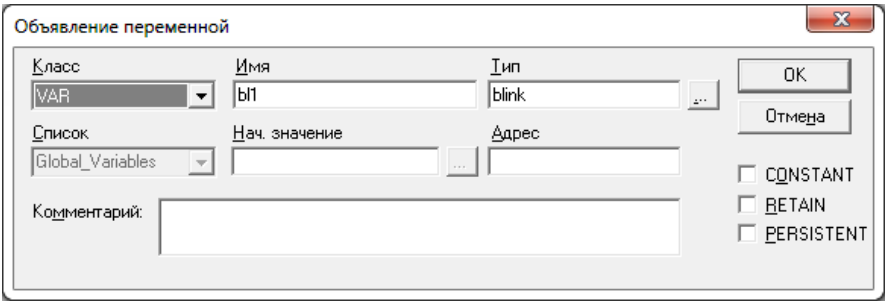


Рисунок 12 – Объявление экземпляра функционального блока

3.4.2 Использование встроенного в ПЛК модуля ШИМ

Модуль ШИМ (Pulse-wide modulator или PWM) – программный модуль, предназначенный для обеспечения функционирования генератора широтно-импульсной модуляции, подключенного к дискретному выходу. Модуль ШИМ является подчиненным подмодулем модуля дискретных выходов.

Модуль имеет 16-ти битовый канал Value – (формат WORD), задающий значение скважности ШИМа. Изменяется от 0 (0%) до 65535 (100%)

Параметры модуля:

– «Номер выхода» (Number of output) – от 0 до 7

– «Период ШИМ в 100 мксек» (Period of PWM in 100 mksec) – от 100 до 360000, значение по умолчанию – 100.

– «Минимальная длительность импульса ШИМ в 100 мксек» (Minimal duration of impulse in 100 mksec) – от 1 до 65000, значение по умолчанию – 30 = 100 мкс.

Для добавления модуля PWM необходимо из контекстного меня модуля дискретных выходов (Discrete outputs) выбрать пункт Добавить Pulse-wide modulator (см. рис. 13).

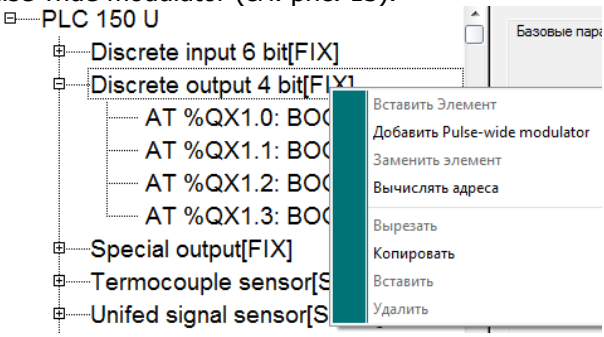


Рисунок 13 – Добавление модуля PWM

Осуществить выдачу последовательности импульсов по методу ШИМ с регулированием скважности импульсов в прямо пропорциональной зависимости от напряжения на аналоговом входе (уставки). Период следования импульсов задать равным 1 с. Скважность импульсов регулировать в пределах от 0 до 65535.

Для выполнения задания дополнительно воспользоваться блоком преобразования данных REAL_TO_WORD.

4 Контрольные вопросы:

1 Для чего необходима унификация уровня аналогового сигнала от первичных преобразователей? Перечислите основные стандартные уровни аналоговых сигналов.

2 С какой целью термосопротивления подключают по трех и четырех проводной схеме? Какая схема подключения используются в работе?

3 Что такое калибровочная характеристика термопары? Какого типа термопара используется в работе?

4 Опишите принцип работы регулятора аналогового типа. В чем заключается метод ШИМ?

5 В чем преимущества использования встроенного в контроллер модуля ШИМ по сравнению с использованием функционального блока генератор в программе?

5 Содержание отчета:

1 Цель работы.

2 Результаты выполнения задания (программы с комментариями).

3 Ответы на контрольные вопросы.

Управление шаговым двигателем при помощи дискретных выходов ПЛК ОВЕН в среде CoDeSys

1 Цель работы:

- изучение принципов конфигурирования и работы дискретных выходов ПЛК;
- освоение понятия «рабочий цикл» ПЛК и «время реакции» ПЛК на практических примерах;
- изучение принципов управления шаговым двигателем с постоянными магнитами.

2 Краткая теория

Задачи управления требуют непрерывного циклического контроля. В любых цифровых устройствах непрерывность достигается за счет применения дискретных алгоритмов, повторяющихся через достаточно малые промежутки времени. Таким образом, вычисления в ПЛК всегда повторяются циклически. Одна итерация, включающая замер, обсчет и выработку воздействия, называется *рабочим циклом* ПЛК. *Рабочий цикл ПЛК* состоит из нескольких фаз: начало цикла, чтение состояния входов, выполнение кода программы пользователя, запись состояния выходов, обслуживание аппаратных ресурсов ПЛК, монитор системы исполнения, контроль времени цикла, переход на начало цикла.

Существуют задачи, в которых время цикла существенно влияет на результат, например, это автоматическое регулирование. Для устранения этой проблемы в ПЛК предусмотрен контроль времени цикла, позволяющий управлять временем реакции ПЛК на внешние воздействия.

Время реакции — это время с момента изменения состояния системы до момента выработки соответствующей реакции. Очевидно, для ПЛК время реакции зависит от распределения моментов возникновения события и начала фазы чтения входов. Если изменение значений входов произошло непосредственно перед фазой чтения входов, то время реакции будет наименьшим и равным времени сканирования. Худший случай, когда изменение значений входов происходит сразу после фазы чтения входов. Тогда время реакции будет наибольшим, равным удвоенному времени сканирования минус время одного чтения входов. Иными словами, время реакции ПЛК не превышает удвоенного времени сканирования.

Помимо времени реакции ПЛК, существенное значение имеет

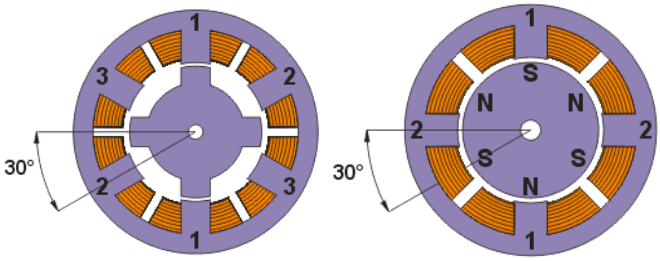
время реакции датчиков и исполнительных механизмов, которое также необходимо учитывать при оценке общего времени реакции системы.

Шаговый двигатель - это электромеханическое устройство, преобразующие сигнал управления в угловое (или линейное) перемещение ротора с фиксацией его в заданном положении без устройств обратной связи. При проектировании конкретных систем приходится делать выбор между сервомотором и шаговым двигателем. Когда требуется прецизионное позиционирование и точное управление скоростью, а требуемый момент и скорость не выходят за допустимые пределы, то шаговый двигатель является наиболее экономичным решением. В отличие от коллекторных двигателей, у которых момент растет с увеличением скорости, шаговый двигатель имеет больший момент на низких скоростях. но гораздо меньшую максимальную скорость.

Имеются следующие виды шаговых двигателей*:* двигатели с переменным магнитным сопротивлением, двигатели с постоянными магнитами, гибридные двигатели.

Шаговые двигатели с переменным магнитным сопротивлением имеют несколько полюсов на статоре и ротор зубчатой формы из магнитомягкого материала. Намагниченность ротора отсутствует. Для простоты на рисунке 1-а ротор имеет 4 зубца, а статор имеет 6 полюсов. Двигатель имеет 3 независимые обмотки, каждая из которых намотана на двух противоположных полюсах статора. Такой двигатель имет шаг 30 град.

При включени тока в одной из катушек, ротор стремится занять положение, когда магнитный поток замкнут, т.е. зубцы ротора будут находиться напротив тех полюсов, на которых находится запитанная обмотка. Если затем выключить эту обмотку и включить следующую, то ротор поменяет положение, снова замкнув своими зубцами магнитный поток. Таким образом, чтобы осуществить непрерывное вращение, нужно включать фазы попеременно. Двигатель не чувствителен к направлению тока в обмотках. Двигатели с переменным магнитным сопротивлением довольно редко используют в индустриальных применениях.



А) Б)

Рисунок 1 - Схема обмоток двигателей: а- с переменным магнитным сопротивлением; б- с постоянными магнитами.

Двигатели с постоянными магнитами состоят из статора, который имеет обмотки, и ротора, содержащего постоянные магниты (рис. 1-б). Чередующиеся полюса ротора имеют прямолинейную форму и расположены параллельно оси двигателя. Благодаря намагниченности ротора в таких двигателях обеспечивается больший магнитный поток и, как следствие, больший момент, чем у двигателей с переменным магнитным сопротивлением. Показанный на рисунке двигатель имеет 3 пары полюсов ротора и 2 пары полюсов статора. Двигатель имеет 2 независимые обмотки, каждая из которых намотана на двух противоположных полюсах статора. Такой двигатель, как и рассмотренный ранее двигатель с переменным магнитным сопротивлением, имеет величину шага 30 град. При включени тока в одной из катушек, ротор стремится занять такое положение, когда разноименные полюса ротора и статора находятся друг напротив друга. Для осуществления непрерывного вращения нужно включать фазы попеременно. На практике двигатели с постоянными магнитами обычно имеют 48 – 24 шага на оборот (угол шага 7.5 – 15 град).

Двигатели с постоянными магнитами подвержены влиянию обратной ЭДС со стороны ротора, котрая ограничивает максимальную скорость. Для работы на высоких скоростях используются двигатели с переменным магнитным сопротивлением.

По способу питания шаговые двигатели подразделяются на биполярные и униполярные. Биполярный двигатель имеет одну обмотку в каждой фазе, которая для изменения направления магнитного поля должна переполюсовываться. Всего биполярный двигатель имеет две обмотки и, соответственно, четыре вывода (рис. 3-а).

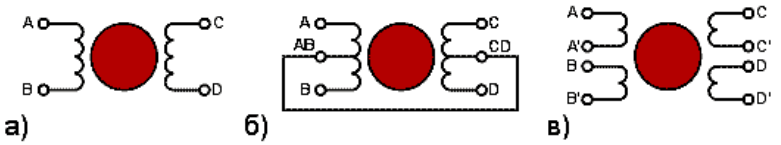


Рисунок 2 - Биполярный двигатель (а), униполярный (б) и четырехобмоточный (в).

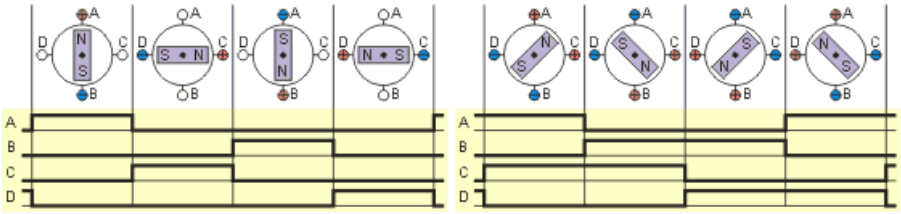
Униполярный двигатель также имеет одну обмотку в каждой фазе, но от середины обмотки сделан отвод. Это позволяет изменять направление магнитного поля, создаваемого обмоткой, простым переключением половинок обмотки. Средние выводы обмоток могут быть объединены внутри двигателя, поэтому такой двигатель может иметь 5 или 6 выводов (рис. 3-б). Иногда униполярные двигатели имеют раздельные 4 обмотки, по этой причине их ошибочно называют 4-х фазными двигателями.

Каждая обмотка имеет отдельные выводы, поэтому всего выводов 8 (рис. 3-в). При соответствующем соединении обмоток такой двигатель можно использовать как униполярный или как биполярный. Если сравнивать между собой биполярный и униполярный двигатели, то биполярный имеет более высокую удельную мощность. При одних и тех же размерах биполярные двигатели обеспечивают больший момент. На практике все же часто применяют униполярные двигатели, так как они требуют значительно более простых схем управления обмотками.

Способы управления фазами шагового двигателя:

Полношаговый режим.

Первый способ обеспечивается попеременной коммутации фаз, при этом они не перекрываются, в один момент времени включена только одна фаза (рис а). Этот способ называют ”one phase on” full step или wave drive mode. Точки равновесия ротора для каждого шага совпадают с «естественными» точками равновесия ротора у незапитанного двигателя. Недостатком этого способа управления является то, что для биполярного двигателя в один и тот же момент времени используется 50% обмоток, а для униполярного – только 25%. Это означает, что в таком режиме не может быть получен полный момент.



а) полношаговый режим, включена одна фаза, "one phase on" full step (wave drive mode)

б) полношаговый режим, включены две фазы, "two-phase-on" full step (full step mode)

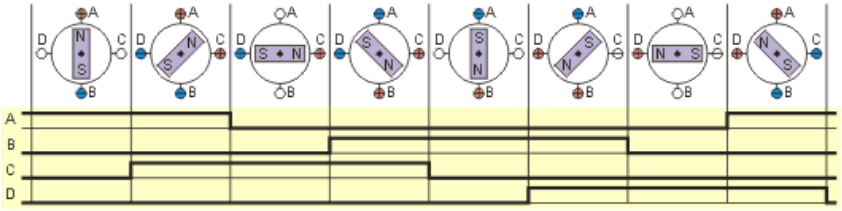
Рисунок 3 – Пошаговый режим

Второй способ - управление фазами с перекрытием: две фазы включены в одно и то же время. Его называют" two-phase-on" full step или просто full step mode. При этом способе управления ротор фиксируется в промежуточных позициях между полюсами статора и обеспечивается примерно на 40% больший момент, чем в случае одной включенной фазы. Этот способ управления обеспечивает такой же угол шага, как и первый способ, но положение точек равновесия ротора смещено на полшага.

В полношаговом режиме с двумя включенными фазами положения точек равновесия ротора смещены на полшага.

Полушаговый режим - комбинация пошаговых, "one and two-phase-on" half step или просто half step mode, когда двигатель делает шаг в половину основного. Этот метод управления достаточно распространен, так как двигатель с меньшим шагом стоит дороже и очень заманчиво получить от 100-шагового двигателя 200 шагов на оборот. Каждый второй шаг запитана лишь одна фаза, а в остальных случаях запитаны две. В результате угловое перемещение ротора составляет половину угла шага для первых двух способов управления.

По сравнению с полношаговым режимом, полушаговый режим имеет следующие преимущества: более высокая разрешающая способность без применения более дорогих двигателей, меньшие проблемы с явлением резонанса.



полушаговый режим, "one and two-phase-on" half step (half step mode)

Рисунок 4 – Полушаговый режим

Микрошаговый режим обеспечивается путем получения поля статора, вращающегося более плавно, чем в полно- или полушаговом режим ах. В результате обеспечиваются меньшие вибрации и практически бесшумная работа вплоть до нулевой частоты.

К тому же меньший угол шага способен обеспечить более точное позиционирование. Существует много различных микрошаговых режимов, с величиной шага от 1/3 полного шага до 1/32 и даже меньше. Шаговый двигатель является синхронным электродвигателем. Это значит, что положение равновесия неподвижного ротора совпадает с направлением магнитного поля статора. При повороте поля статора ротор тоже поворачивается, стремясь занять новое положение равновесия. Чтобы получить нужное направление магнитного поля, необходимо выбрать не только правильное направление токов в катушках, но и правильное соотношение этих токов. Во многих приложениях, где требуются малые относительные перемещения и высокая разрешающая способность, микрошаговый режим способен заменить механический редуктор. Однако в большинстве случаев для обычных двигателей нельзя гарантировать точного позиционирования в микрошаговом режиме.

3 Задание

3.1 Программа «моргающий индикатор»

ПЛК функционирует циклически — чтение входов, выполнение прикладной программы и запись выходов. В результате **прикладное программирование для ПЛК существенно отличается от традиционной модели**, применяемой при работе на языках высокого уровня ПК.

Чтобы понять этот принцип реализуем простейшую задачу: необходимо запрограммировать моргающий (раз в секунду) световой индикатор, подключенный к дискретному выходу ПЛК. Алгоритм работы программы следующий:

1) Включить выход;
2) Выдержать паузу (1 сек.);
3) Выключить выход;
4) Выдержать паузу (1 сек.);
5) Переход к шагу 1.

Реализуем по этому алгоритму программу для ПЛК (см. рис. 5).

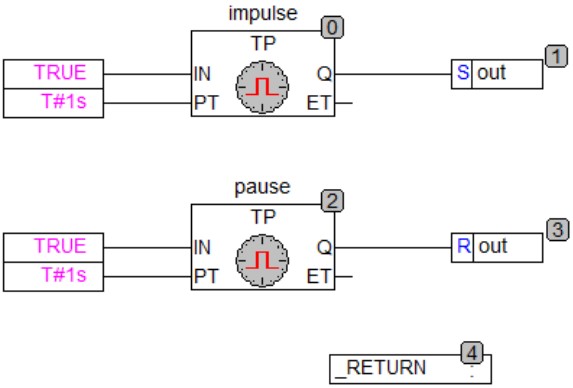


Рисунок 5 – Программа «моргающий светодиод» (вариант 1).

- Создадим новый проект в среде CoDeSys. В качестве языка программирования выберем CFC.

- Разместим в рабочем поле два экземпляра функционального блока TP – «таймер» (один назовем impulse, а другой pause).

- Для установки дискретного выхода контроллера в 1 или 0 воспользуемся кнопкой на панели инструментов (S – установка и фиксация значения 1 на выходе, R – установка и фиксация значения 0 на выходе).

- Для наблюдения результата работы программы воспользуемся режимом эмуляции (установить флажок Онлайн> Режим эмуляции).

- Порядок исполнения функциональных блоков программы определяется номером, указанным в левом верхнем углу каждого блока. В нашем случае программа содержит 3 блока с номерами 0, 1, 2, 3. Блок 4 (переход на начало программы) присутствует в любой программе для ПЛК по умолчанию и будет добавлен при запуске программы (Онлайн> Подключение; и затем Онлайн> Старт).

- Будем наблюдать результат работы программы. Зафиксируем состояние выхода out.

- Изменим программу таким образом, чтобы сперва выход выключался, а затем включался и также зафиксируем состояние выхода при работе программы.

Наблюдаем, что программа функционирует неправильно по следующим причинам:

- выход всегда будет оставаться в выключенном (или включенном) состоянии, поскольку физически установка значений

выходов производится по окончании прикладной программы один раз. Промежуточные изменения значений выходов не отображаются на аппаратные средства. Конечно, значение переменной будет изменяться многократно, но определяющим выход станет только последнее значение.

- ПЛК вынужден заниматься организацией задержки времени. Вполне вероятно, что, кроме мерцания одним выходом, ПЛК должен будет выполнять еще и другую работу.

Значит, выдержку времени необходимо организовать иначе.

Достаточно засечь время и заняться другими делами, контролируя периодически часы. Здесь нет ничего особенного. Так поступает обычно и большинство людей в ожидании назначенного часа.

Несмотря на описанные сложности, алгоритм получился в итоге проще. Так и должно быть. Технология ПЛК специально ориентирована на подобные задачи.

Реализуем программу по указанному алгоритму (рис. 6). В учебных целях для организации задержки времени будем использовать функциональный блок TON (задержка включения). Дополнительно предусмотрим возможность остановки и запуска программы по внешнему сигналу (дискретный вход контроллера - in).

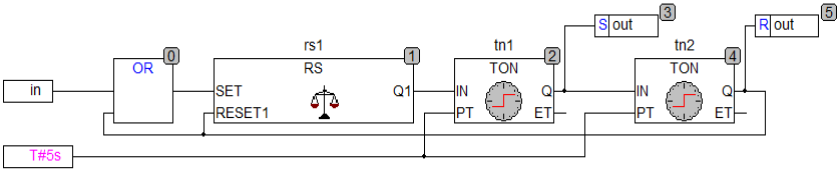


Рисунок 6 – Программа «моргающий светодиод» (вариант 2).

Сделать выводы о назначении элементов OR (или) и RS (триггер) в программе.

3.2 Управление шаговым двигателем.

3.2.1 Написать программу для управления шаговым двигателем в режиме полного шага. Включение двигателя должно осуществляться от дискретного входа ПЛК. Схема подключения двигателя приведена на рис. 7.

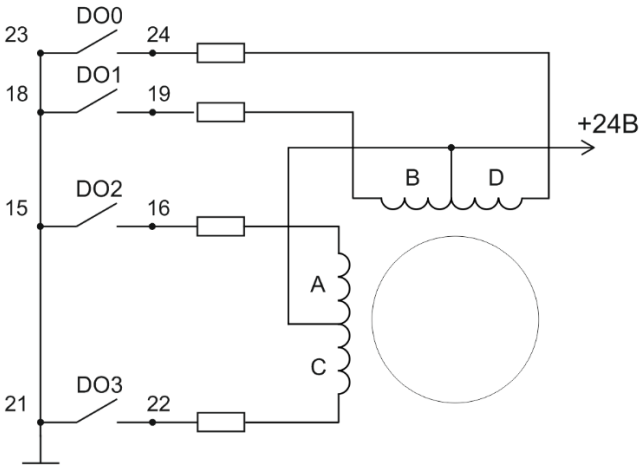


Рисунок 7 – Схема подключения ШД к дискретным выходам ПЛК.

3.2.2 Изменить программу из пункта 3.2.1 для обеспечения возможность реверса двигателя (для изменения направления вращения использовать дискретный выход ПЛК).

3.2.3 Написать программу для управления шаговым двигателем в режиме полушага.

3.2.4 Предусмотреть возможность вращения двигателя на заданный угол (по значению переменной программы).

4 Контрольные вопросы:

1 Перечислите фазы, входящие в рабочий цикл ПЛК. С чем по вашему связана необходимость записи копии значения входов в ОЗУ?

2 Каким образом осуществляется контроль времени цикла в ПЛК, как записанная в ПЛК программа защищена от «зависания»?

3 Какие способы управления шаговыми двигателями вы знаете? Перечислите их достоинства и недостатки.

4 Укажите назначение функциональных блоков TP, TON, TOF.

5 Для чего в программе используются атрибуты выходов R и S?

5 Содержание отчета:

1 Цель работы.

2 Результаты выполнения задания (программы с комментариями).

3 Ответы на контрольные вопросы.

Программирование системы доступа в помещение на языках IL, LD, ST в среде CoDeSys

1 Цель работы:

- закрепление понятия «рабочий цикл» ПЛК на практических примерах;
- изучение принципов программирования ПЛК на языках LD, IL, ST;

2 Краткая теория

Чтобы писать хорошие программы для ПЛК, нужно научиться думать определенным образом. Секрет состоит в том, чтобы представлять себе контроллер не как машину, последовательно выполняющую команды программы, а как конечный автомат. В любом автомате существует множество входов (X), множество выходов (Y) и множество возможных состояний (S). В нашем случае это конечные множества, поскольку число входов-выходов ПЛК ограничено, так же как и объем памяти переменных (определяющих возможные состояния). Начальное состояние (s0) однозначно определено. Автомат работает по тактам, для ПЛК это рабочий цикл. В каждом такте значения входов известны. Значения выходов определяются (функция выходов X) значениями входов и текущим состоянием. Реакция автомата зависит только от текущего состояния без предыстории, т. е. не важно, как он пришел в данное состояние. Вместе с тем текущее состояние также изменяется по тактам, автомат переходит в новое состояние (функция переходов). В теории автоматов описанные шесть объектов принято называть конечным автоматом Мили.

Контроллер вычисляет программно заданную функцию выходов и функцию переходов. В каждом рабочем цикле ПЛК выполняет расчет новых значений для выходов, которые необходимо изменить. В итоге классическая прикладная программа ПЛК оказывается более похожей на вычисление по формуле.

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

Внедрение стандарта на программирование ПЛК дало фундамент для создания единой школы подготовки специали-

стов. Человек, прошедший обучение по программе, включающей стандарт МЭК 61131, сможет работать с ПЛК любой фирмы. В то же время, если он имел ранее опыт работы с любыми ПЛК, его навыки окажутся полезными и существенно упростят изучение новых возможностей.

Вообще стандартные компоненты МЭК для программиста, как дороги для автомашин. Количество возможных путей всегда очень ограничено. Ближе полем, но по дороге быстрее.

Список инструкций IL (instruction list)

. Практически все производители ПЛК Европы создавали подобные системы программирования, похожие на современный язык IL. Наибольшее влияние на формирование современного IL оказал язык программирования STEP контроллеров фирмы Siemens.

На IL можно реализовать алгоритм любой сложности, хотя текст будет достаточно громоздким. В составе МЭК-языков IL применяется при создании **компактных компонентов**, требующих **тщательной проработки**, на которую не жалко времени. При работе с IL гораздо адекватнее, чем с другими языками, можно представить, как будет выглядеть **оттранслированный код**. Текст на IL — это текстовый список последовательных инструкций. Каждая инструкция записывается на отдельной строке. Инструкция может включать 4 поля, разделенные пробелами или знаками табуляции:

Метка: Оператор, Операнд Комментарий

Абсолютное большинство инструкции IL выполняют некоторую операцию с содержимым аккумулятора. В аккумулятор можно поместить значение типа BOOL, затем INT или REAL, транслятор не будет считать это ошибкой. В стандарте МЭК вместо термина «аккумулятор» используется термин «**результат**». Так, инструкция берет «текущий результат» и формирует «новый результат». Основные операторы IL представлены в Таблице

Оператор	**Пример**	**Наименование**
LD	LD *1*	Загрузить значение операнда в аккумулятор
LDN	LDN *1*	Загрузить значение операнда в аккумулятор с инверсией
ST	ST *X*	Присвоить значение аккумулятора операнду
STN	STN *X*	Присвоить значение аккумулятора с инверсией операнду

S	S *X*	Если аккумулятор ИСТИНА, установить логический операнд (ИСТИНА)
R	R *X*	Если аккумулятор ИСТИНА, сбросить логический операнд (ЛОЖЬ)
NOT	NOT	Поразрядная инверсия аккумулятора
ND	AND *4*	Поразрядное И
OR	OR *1*	Поразрядное ИЛИ
XOR	XOR *14*	Поразрядное исключающее ИЛИ
EQ	EQ X	A = X
NE	NE X	A <> X
GE	GE X	A >= X
GT	GT X	A > X
LE	LE X	A <= X
LT	LT X	A < X
ADD	ADD X	A=A+X
SUB	SUB X	A=A-X
MUL	MUL X	A=A*X
DIV	DIV X	A=A/X
MOD	MOD X	A=остаток (A/X)
SHL	SHL	Сдвиг аккумулятора влево
SHR	SHR	Сдвиг аккумулятора вправо
ROL	ROL	Циклический сдвиг аккумулятора влево
ROR	ROR	Циклический сдвиг аккумулятора вправо
JMP	JMP m1	Безусловный переход
JMPC	JMPC m2	Переход если аккумулятор ИСТИНА
JMPCN	JMPCN m3	Переход если аккумулятор ЛОЖЬ

Структурированный текст ST (structured text)

ST – текстовый язык высокого уровня (схож с паскалем). ST – это лучший язык для программирования условий и циклов (if, while, case, for). Основой ST-программы служат выражения. В текст могут быть введены комментарии. Пустое выражение состоит из точки с запятой «;» Для точки с запятой транслятор не генерирует никакого кода. Пример использования IF:

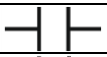
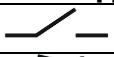
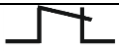
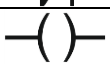
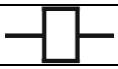
```
IF (x = 1) AND (z=2) THEN
y:=1;
ELSIF x > 1 THEN
y := y - 1;
ELSE
y:=2;
END_IF
```

Релейные диаграммы LD (ladder diagram)

Язык релейных диаграмм или релейно-контактных схем (РКС) — графический язык, реализующий структуры электрических цепей. Графически LD-диаграмма представлена в виде двух вертикальных шин питания. Между ними расположены цепи, образованные соединением контактов. Нагрузкой каждой цепи служит реле. Каждое реле имеет контакты, которые можно использовать в других цепях. LD идеально подходит не только для построения релейных автоматов, но и для программной реализации комбинационных логических схем. Благодаря возможности включения в LD функций и функциональных блоков, выполненных на других языках, сфера применения языка практически не ограничена.

В LD каждому контакту ставится в соответствие логическая переменная, определяющая его состояние. Если контакт замкнут, то переменная имеет значение ИСТИНА. Если разомкнут ЛОЖЬ. Имя переменной пишется над контактом и фактически служит его названием. Последовательное соединение контактов или цепей равноценно логической операции И. Параллельное соединение образует монтажное ИЛИ. Цепь может быть либо замкнутой (ON), либо разомкнутой (OFF). Это как раз и отражается на обмотке реле и соответственно на значении логической переменной обмотки (ИСТИНА/ЛОЖЬ).

Идеология релейных схем подразумевает параллельную работу всех цепей. Ток во все цепи подается одновременно. В LD решение диаграммы выполняется последовательно слева направо и сверху вниз. **В каждом рабочем цикле однократно выполняются все цепи диаграммы**, что и создает эффект параллельности работы цепей.

LD	ЕСКД	Описание
		Нормально разомкнутый контакт
		Нормально замкнутый контакт
		Обмотка реле

3 Задание

3.1 Написание программы.

Инфракрасные датчики Д1 и Д2 установлены внутри и снаружи помещения. Датчики оснащены электромагнитным реле, которое замыкается при их срабатывании. Выходы реле подклю-

чены к дискретным входам контроллера. При срабатывании внешнего, а затем внутреннего датчика происходит регистрация посетителя, зашедшего в помещение и значение счетчика числа посетителей увеличивается на 1. Когда срабатывает сначала внутренний датчик, а затем внешний, значение счетчика уменьшается на 1. Если в помещении находятся посетители, то автоматически включается освещение (дискретный выход ПЛК).

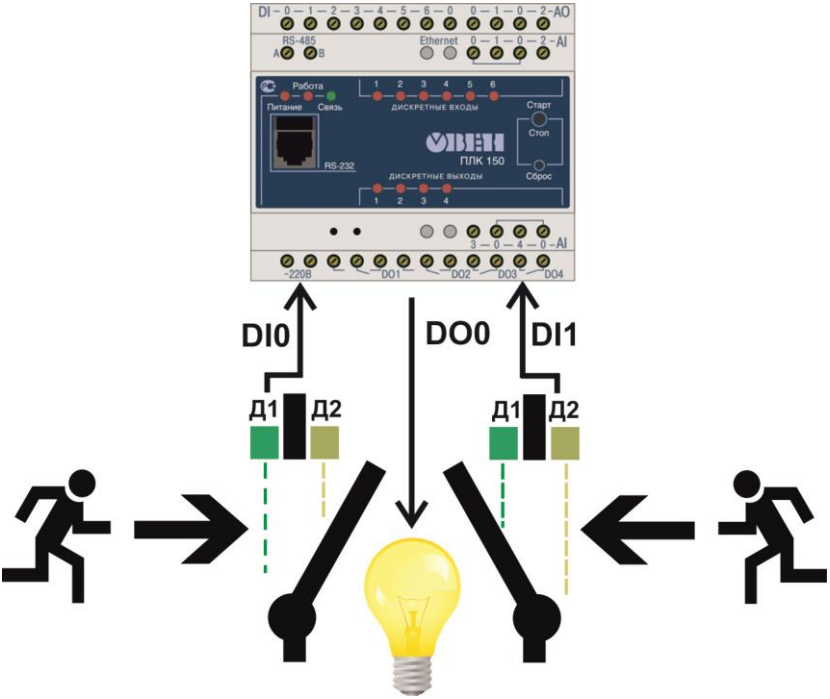


Рисунок 1 – Система подсчета числа посетителей.

3.1.1 Добавим в проект программу на языке IL.

Сконфигурируем дискретные входы и выходы ПЛК. Входы DI0 и DI1 обозначим как **in1** и **in2** соответственно, выход DO0 – **lamp**. Для запоминания состояния дискретных входов в предшествующем текущему цикле ПЛК введем в программе две переменные (типа BOOL) – **in** и **ex**. Создадим также (см. рис 2.) глобальную переменную **people** типа INT (переменную которую можно использовать во всем проекте, а не только в конкретной программе).

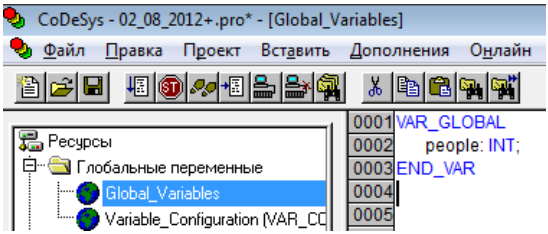


Рисунок 2 – Создание глобальной переменной

Добавьте следующий текст программы в редакторе IL:

```
VAR
in, ex: BOOL;
END_VAR

        LD          in
        NOT
        AND         in1
        AND         in2
        JMPCN       m1
        LD          people
        ADD         1
        ST          people
m1:
        LD          ex
        NOT
        AND         in1
        AND         in2
        JMPCN       m2
        LD          people
        SUB         1
        ST          people
m2:
        LD          people
        GE          1
        JMPC        m3
        LD          0
        ST          lamp
        JMP         m4
m3:
        LD          1
```

```
        ST          lamp
m4:
        LD          in1
        ST          ex
        LD          in2
        ST          in
```

Запустите программу на исполнение и проверьте правильность ее функционирования (подсчет числа посетителей и включение освещения). Напишите комментарии, поясняющие назначение каждой команды в программе.

3.1.2 Программа на языке ST

Удалите из проекта программу на языке IL и добавьте новую программу на языке ST. Задайте локальные переменные аналогично п.п. 3.1.1.

```
IF in=FALSE AND in1=TRUE AND in2=TRUE THEN
people:=people+1;
END_IF
IF ex=FALSE AND in2=TRUE AND in1=TRUE THEN
people:=people-1;
END_IF
IF people>=1 THEN
lamp:=TRUE;
ELSE
lamp:=FALSE;
END_IF
ex:=in1;
in:=in2;
```

Запустите программу на исполнение и проверьте правильность ее функционирования. Напишите комментарии, поясняющие назначение каждой команды в программе.

3.1.3 Программа на языке LD

Удалите из проекта программу на языке ST и добавьте новую программу на языке LD (см. рис. 3). Задайте локальные переменные аналогично п.п. 3.1.1.

Запустите программу на исполнение и проверьте правильность ее функционирования. Напишите комментарии, поясняющие назначение блоков программы.

4 Контрольные вопросы:

1 В чем состоит назначение стандарта МЭК 61131-3? Что из себя представляет ПЛК с точки зрения инженера?

1 Исходя из результатов моделирования сделайте вывод о

возможностях, предназначении, достоинствах и недостатках (в сравнении друг с другом) языков МЭК – IL, ST, LD.

2 Опишите принцип опроса входного и выходного инфракрасных датчиков. По какой причине передний фронт сигнала анализируется на датчике, срабатывающем во вторую очередь?

3 Имеет ли значение на каком расстоянии относительно друг друга установлены инфракрасные датчики? Учитывается ли расстояние между датчиками в программе подсчета?

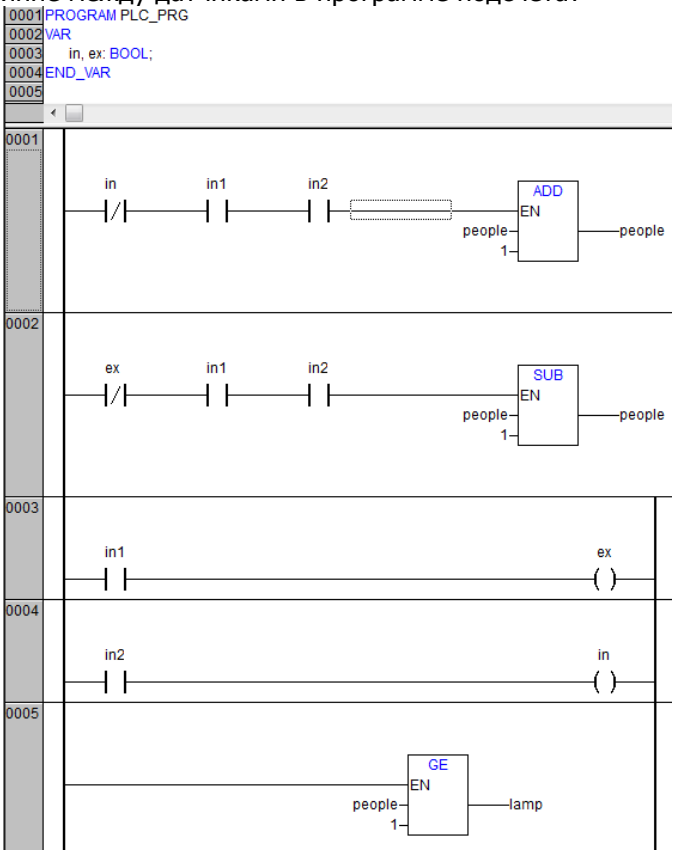


Рисунок 3 – Программа на языке LD

5 Содержание отчета:

1 Цель работы.

2 Результаты выполнения задания (программы с комментариями).

3 Ответы на контрольные вопросы.

Использование средств визуализации, программирование на языках FBD и CFC в среде CoDeSys.

1 Цель работы:

- закрепление особенностей программирования ПЛК на практических примерах;
- изучение принципов программирования ПЛК на языках FBD и CFC;
- изучение средств визуализации в среде CoDeSys.

2 Краткая теория

Функциональные диаграммы FBD.

FBD (Function Block Diagram) — это графический язык программирования. Диаграмма FBD очень напоминает принципиальную схему электронного устройства на микросхемах. В отличии от LD «проводники» в FBD могут проводить сигналы (передавать переменные) любого типа (логический, аналоговый, время и т. д.). Выходы блоков могут быть поданы на входы других блоков либо непосредственно на выходы. Программа в FBD не обязательно должна представлять большую единую схему. Как и в LD, диаграмма образуется из множества цепей, которые выполняются одна за другой. Вход блока может быть соединен с выходом блока, расположенного слева от него. Помимо этого, вход может быть соединен с переменной или константой. Выполнение FBD-цепей идет слева направо, сверху вниз. Обратные соединения запрещены. Для создания обратной связи используются явно объявленные внутренние переменные.

Непрерывные функциональные схемы (CFC)

В редакторе CFC (Continuous Flow Chart) нет цепей (как на LD и FBD), и поэтому элементы могут располагаться где угодно. К элементам языка CFC относятся блоки, входы, выходы, возвраты, произвольные переходы, метки и комментарии. Основное преимущество CFC редактора перед FBD заключается в том, что в схемы можно непосредственно добавлять линии обратной связи.

Визуализация (HMI) (*human-machine interface* — «человеко-машинный интерфейс») — средство, предназначенное для визуализации параметров процесса (объекта) и/или осуществления операторского управления. Типовая визуализация предоставляет пользователю следующую функциональность: отображение параметров технологического процесса (или объекта) в текстовом или графическом режимах; управление и обработка аварийных

сообщений, регистрация времени и даты возникновения аварийных сообщений; ручное управление с помощью функциональных кнопок; построение диаграмм и трендов, отображение сводных отчетов. В графическом режиме визуализация процесса происходит с помощью интерактивных мнемосхем. В текстовом режиме процесс отображается в виде строк или в виде специальных таблиц.

3 Задание

3.1 Написание программы.

Инфракрасные датчики Д1 и Д2 установлены внутри и снаружи помещения. Датчики оснащены электромагнитным реле, которое замыкается при их срабатывании. Выходы реле подключены к дискретным входам контроллера.

При срабатывании внешнего, а затем внутреннего датчика происходит регистрация посетителя, зашедшего в помещение и значение счетчика числа посетителей увеличивается на 1. Когда срабатывает сначала внутренний датчик, а затем внешний, значение счетчика уменьшается на 1. Если в помещении находятся посетители, то автоматически включается освещение (дискретный выход ПЛК).

3.1.1 Добавим в проект программу на языке FBD.

Сконфигурируем дискретные входы и выходы ПЛК. Входы DI0 и DI1 обозначим как **in1** и **in2** соответственно, выход DO0 – **lamp**. Для запоминания состояния дискретных входов в предшествующем текущему циклe ПЛК введем в программе две переменные (типа BOOL) – **in** и **ex**. Создадим также глобальную переменную **people** типа INT

Запустите программу на исполнение и проверьте правильность ее функционирования (подсчет числа посетителей и включение освещения). Напишите комментарии, поясняющие назначение каждого блока в программе.

3.1.2 Визуализация

Добавим в проект возможность визуализации подсчета числа посетителей, отображения состояния датчиков и освещения в процессе отладки программы. Для этого на вкладке Визуализация структуры проекта щелчком правой кнопки мыши добавим объект с именем people (см. рис.1).

Во вновь созданном объекте при помощи инструментов Прямоугольник изобразим план помещения. Наличие освещения будем отображать на схеме при помощи инструмента Эллипс. Для привязки его вида к значению дискретного выхода lamp произве-

дем следующие действия:

- двойным щелчком мыши откроем свойства элемента Эллипс;

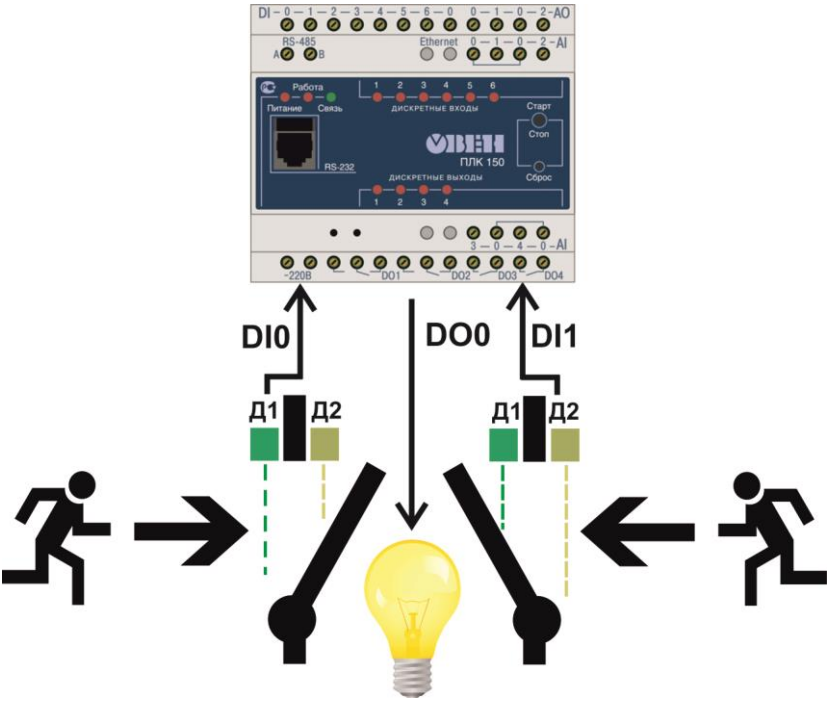


Рисунок 1 – Система подсчета числа посетителей.

- на вкладке Цвета выберем в качестве заливки тревожного цвета – желтый, а цвет элемента по умолчанию – серый;

- на вкладке Переменные в поле Изменение цвета нажмем клавишу F2. Выберем в качестве переменной заливки значение дискретного выхода lamp (Global_variables > lamp);

Состояние входных датчиков будем отображать при помощи элементов Прямоугольник предварительно привязав изменение цвета заливки к переменным входов in1 и in2.

Для отображения числа посетителей разместим в поле визуализации элемент прямоугольник и откроем его свойства.

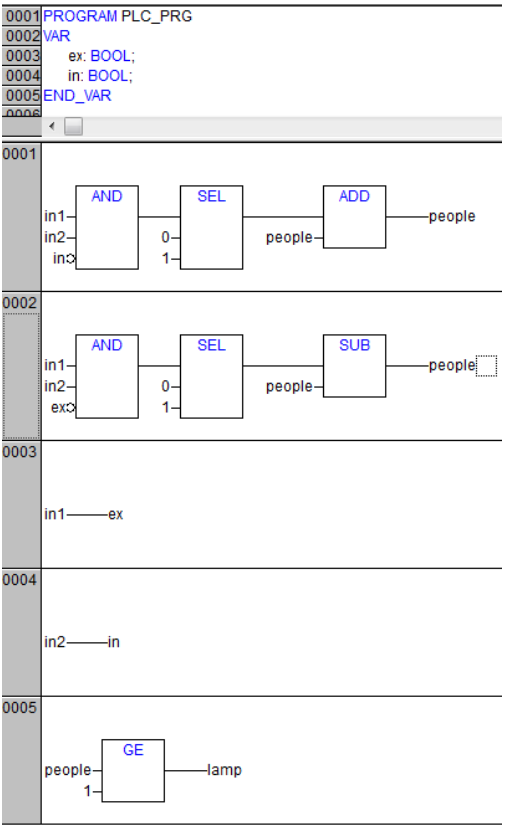


Рисунок 2 – Программа на языке FBD

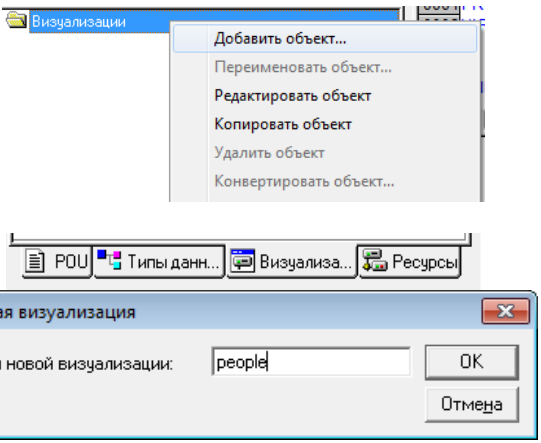


Рисунок 3 – Создание новой визуализации

Рисунок 4 – Примерный вид визуализации

На вкладке Текст в поле строка введем текст: Количество посетителей: %s. Запись %s означает что в данной позиции будет отображаться привязанная к элементу прямоугольник переменная. Привязку переменной people осуществим на вкладке Переменные в поле Вывод текста по нажатию F2.

Откроем в среде CoDeSys одновременно окно визуализации и редактор программы. Для удобства можно воспользоваться опцией Окно > По горизонтали. Будем наблюдать работу программы для проверки правильности работы визуализации.

3.1.3 Составьте аналогичную п.п. 3.1.1 программу на языке CFC.

Для активации процесса подсчета людей используйте входы активации блоков EN (аналогично п.п. 3.1.3 в работе №4).

Запустите программу на исполнение и проверьте правильность ее функционирования. Напишите комментарии, поясняющие назначение каждого блока в программе.

3.1.4 Составьте программу на языке CFC, учитывающую возможность изменения расстояния между инфракрасными датчиками Д1 и Д2. Для составления программы используйте дополнительно блоки R_TRIG и RS (предварительно изучив по справочной системе принцип их работы).

Запустите программу на исполнение и проверьте правильность ее функционирования. Напишите комментарии, поясняющие назначение каждого блока в программе.

4 Контрольные вопросы:

1 В каких случаях удобно использовать язык функциональных блоков FBD, в чем его преимущества и недостатки?

2 Укажите отличия языка CFC от FBD, его основные преимущества.

3 Что из себя представляет встроенная в CoDeSys визуализация, в чем ее основное предназначение?

5 Содержание отчета:

1 Цель работы.

2 Результаты выполнения задания (программы с комментариями и визуализация).

3 Ответы на контрольные вопросы.

Программирование при помощи SFC диаграмм в среде CoDeSys

1 Цель работы:

- изучение особенностей автоматного подхода для программирования ПЛК на практических примерах;
- изучение принципов программирования ПЛК на языке SFC;
- закрепление навыков работы со средствами визуализации.

2 Краткая теория

Как известно, ПЛК представляет собой не просто машину, последовательно выполняющую команды программы, а конечный автомат, имеющий (как и любой другой автомат) множество входов (X), множество выходов (Y) и множество возможных состояний (S). В нашем случае это конечные множества, поскольку число входов-выходов ПЛК ограничено, так же как и объем памяти переменных (определяющих возможные состояния). Начальное состояние (s0) однозначно определено. Автомат работает по тактам, для ПЛК это рабочий цикл. В каждом такте значения входов известны. Значения выходов определяются (функция выходов X) значениями входов и текущим состоянием. Реакция автомата зависит только от текущего состояния без предыстории, т. е. не важно, как он пришел в данное состояние. Вместе с тем текущее состояние также изменяется по тактам, автомат переходит в новое состояние (функция переходов). В теории автоматов описанные шесть объектов принято называть конечным автоматом Мили.

Контроллер вычисляет программно заданную функцию выходов и функцию переходов. В каждом рабочем цикле ПЛК выполняет расчет новых значений для выходов, которые необходимо изменить. В итоге классическая прикладная программа ПЛК оказывается более похожей на вычисление по формуле.

Несколько расширив понятие автомата, мы можем рассматривать переходы как функции событий. События не обязательно должны быть связаны с входами, это достаточно абстрактное понятие. Тогда окончание таймаута можно будет понимать, как событие, причем совершенно не важно, как конкретно реализован сам таймер. Модель такой системы удобно изобразить в виде направленного графа состояний (state charts). Состояния отображаются овалами, содержащими значения набора переменных, а переходы — направленными дугами.

Реально возможности ПЛК существенно превышают конечные автоматы. Далеко не все, что можно сделать на ПЛК, вписывается в рамки конечных автоматов. Это функции управления по времени, математическая обработка данных, регулирование и т. д. Тем не менее применение формализма конечных автоматов позволяет значительно упростить процесс проектирования. Причем это относится не только к ПЛК.

Диаграммы SFC

SFC (Sequential Function Chart). Благодаря SFC идея превращения модели системы (блок-схемы) в законченную программу стала реальностью. В отличие от применения вспомогательных средств моделирования SFC дает действующий непосредственно в ПЛК прототип.

Диаграммы SFC имеют выраженную направленность сверху вниз и отражаются прямыми линиями. Позиции в SFC называют шагами или этапами. На диаграмме они отражаются в виде прямоугольников. Задать несколько стартовых шагов в SFC нельзя, только один шаг диаграммы является начальным. Графическая диаграмма SFC состоит из шагов и переходов между ними. Разрешение перехода определяется условием. С шагом связаны определенные действия. Описания действий выполняются на любом языке МЭК. Сам SFC не содержит каких-либо управляющих команд ПЛК. Действия могут быть описаны и в виде вложенной SFC-схемы. Можно создать несколько уровней подобных вложений, но в конечном счете действия нижнего уровня все равно необходимо будет описать на IL, ST, LD или FBD.

Рисунок 1 – Пример SFC диаграммы

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

Ниже шага на соединительной линии присутствует горизонтальная черта, обозначающая переход. Условием перехода может служить логическая переменная, логическое выражение, константа или прямой адрес. Переход выполняется при соблюдении двух условий:

1) переход разрешен (соответствующий ему шаг активен);

2) условие перехода имеет значение TRUE.

Простые условия отображаются непосредственно на диаграмме справа от черты, обозначающий переход. В CoDeSys на диаграмме можно записывать только выражения на языке ST (T1 > 50).

Для громоздких условий применяется другой подход. Вместо условия на диаграмме записывается только идентификатор перехода. Само же условие описывается в отдельном окне с применением языка IL, ST, LD или FBD.

Каждая SFC-схема начинается с шага, выделенного графически двойными вертикальными линиями или по всему периметру. Это — начальный шаг. Наименование начального шага может быть произвольным (по умолчанию Init). Начальный шаг присутствует обязательно, хотя и может быть пустым.

Несколько ветвей SFC могут быть параллельными. Признаком параллельных ветвей на схеме является двойная горизонтальная линия. Каждая параллельная ветвь начинается и заканчивается шагом. То есть условие входа в параллельность всегда одно, условие выхода тоже одно на всех. Параллельные ветви выполняются теоретически одновременно. В жизни это означает — в одном рабочем цикле, слева направо. Условие перехода, завершающее параллельность, проверяется только в случае, если в каждой параллельной ветви активны последние шаги.

Несколько ветвей SFC могут быть альтернативными ветвями. Признаком альтернативных ветвей на схеме является одинарная горизонтальная линия. Каждая альтернативная ветвь начинается и заканчивается собственным условием перехода. Проверка альтернативных условий выполняется слева направо. Если верное условие найдено, то прочие альтернативы не рассматриваются. В альтернативных ветвях всегда работает только

одна из ветвей, поэтому ее окончание и будет означать переход к следующему за альтернативной группой шагу.

Упрощенный SFC

Помимо рассмотренной стандартной МЭК-технологии связи шагов и действий, в CoDeSys реализована упрощенная реализация. Смысл ее заключается в применении более простого, компактного и быстрого последовательного SFC-исполнителя. Помимо этого, сами диаграммы получаются компактнее и часто проще для понимания. Безусловно, возможности упрощенной реализации несколько уже — нельзя включать и выключать действия в разных шагах и управлять активностью действий по времени.

Действия могут быть трех классов — текущее, входное и выходное. Графически действия на диаграмме никак не отображаются, их редактирование выполняется в отдельных окнах. В упрошенной реализации действия принадлежат шагу. То есть действие нельзя вызвать из другого шага или откуда-либо еще. Можно считать, что каждый прямоугольник шага при его увеличенном рассмотрении содержит 3 раздела, соответствующие трем возможным действиям. Если шаг удалить, то и все его действия будут утрачены. Шаги, содержащие действие, на схеме отличаются тем, что верхний правый угол прямоугольника закрашен.

Весьма вероятен случай, когда определенные действия нужно выполнить в шаге только один раз. Например, включить нагрев в начале активности шага и выключить при переходе на другой шаг. С этой целью и предусмотрены входное и выходное действия. Входное действие обозначается сегментом 'E' (Entry) в нижнем левом углу прямоугольника шага и выполняется однократно при активизации шага. Выходное обозначается сегментом 'X' (eXit) в нижнем левом углу прямоугольника шага. Выходное действие выполняется однократно при завершении работы шага.

Стандартный SFC

В отличие от упрощенных действий, действия МЭК не принадлежат конкретному шагу, а являются самостоятельными программными элементами SFC-компонента. При применении МЭК-действий сначала определяются действия (виды работ), которые должна выполнять система, а затем уже составляется диаграмма, в которой определяется их порядок и взаимосвязь. Каждое действие сопоставляется одному или нескольким шагам. Причем вполне возможно, что некоторое действие должно запускаться в одном шаге и останавливаться в другом. Действия МЭК показываются на SFC-диаграмме в виде прямоугольников, расположенных справа от шага и привязанных к нему графически. Пря-

моугольник, отображающий действие, содержит в левой части специальное поле — классификатор. Классификатор определяет способ влияния активного шага на данное действие. Возможны следующие классификаторы:

N — несохраняемое действие (Non-stored). Данное действие будет выполняться в каждом рабочем цикле, пока активен шаг.

P — импульс (Pulse). Действие выполняется один раз при активации и второй раз после деактивации шага.

S — сохраняемое (Stored). Действие активируется и остается активным до сброса. Действие продолжит выполняться в каждом цикле даже тогда, когда шаг уже не активен.

R — сброс (Reset). Действие деактивируется.

L — ограниченное по времени (time Limited). Действие активируется вместе с шагом и остается активным на заданное время, но не дольше, чем шаг.

SL — сохраняемое и ограниченное по времени (Stored and time Limited). Действие активируется вместе с шагом и остается активным заданное время, вне зависимости от активности шага. Действие можно деактивировать досрочно из другого шага с классификатором R.

D — отложенное (Delayed). Действие активируется через заданное время после активации шага и остается активным, пока активен шаг. Если шаг окажется активным меньше заданного времени, то действие не будет активировано.

DS — отложенное сохраняемое (Delayed and Stored). Действие активируется через заданное время после активации шага и остается активным до сброса. Если шаг активен меньше заданного времени, то действие не будет активировано. При параллельном выполнении сброса в процессе отсчета времени (в другом шаге с классификатором R) действие не будет активироваться.

SD — сохраняемое отложенное (Stored and time Delayed). Действие активируется через заданное время после активации шага, даже если шаг уже не активен. Но если в процессе отсчета задержки активации выполнить сброс (в другом шаге с классификатором R), то активация не произойдет. Активированное действие остается активным до сброса. Классификаторы L, D, SD, DS и SL требуют указания константы времени в формате TIME. Например: D T#10s. Каждое активное действие выполняется еще один раз уже после деактивации. Это необходимо для того, чтобы действия могли отработать потерю активности и выполнить некоторые завершающие операции.

3 Задание

3.1 Программа «технология приготовления продукта»

Наиболее логичное использование языка SFC – это составление программы для процесса, который имеет четко определенную технологию или «рецепт» действий.

Рассмотрим принцип составления МЭК SFC программы. За примером не нужно ходить далеко. На коробке с овсянкой описана вполне достойная для выражения на SFC технологическая задача. В ней опущены отдельные очевидные для человека детали. Для контроллера их придется запрограммировать.

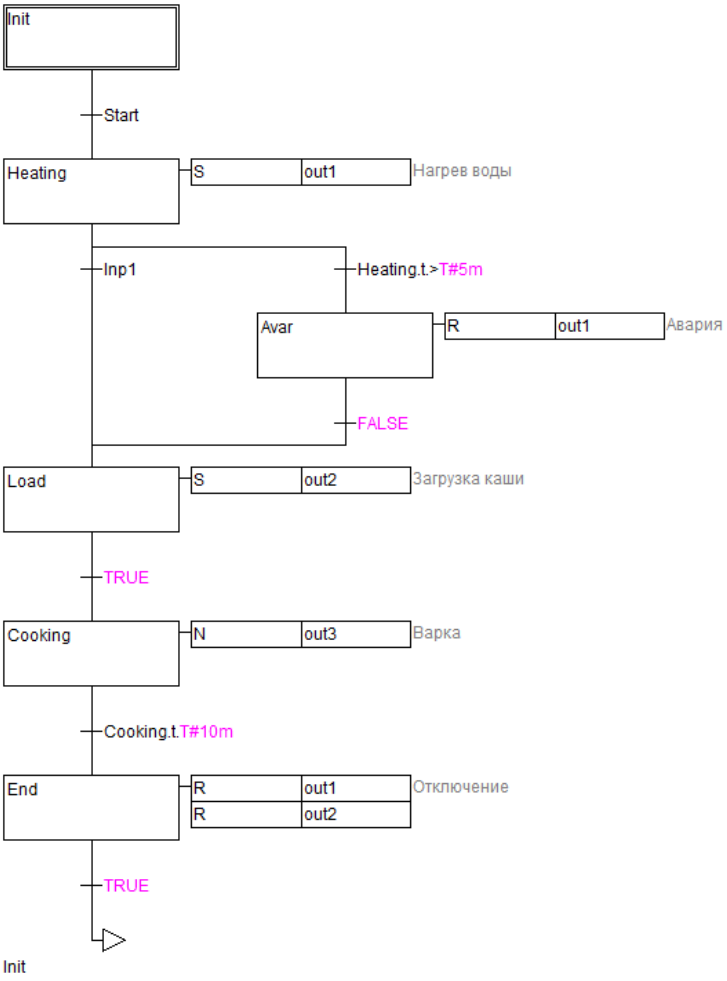


Рисунок 2 – МЭК SFC программа «овсянка»

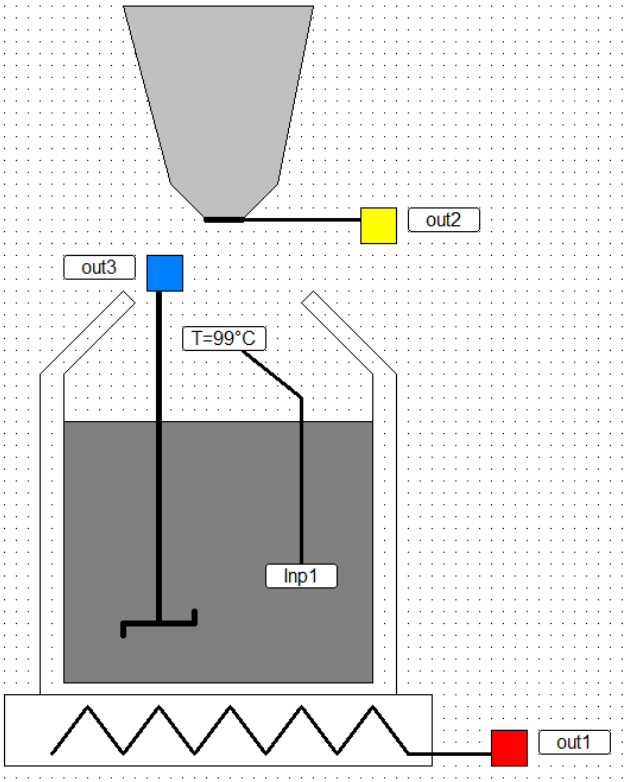


Рисунок 3 – Визуализация «овсянка»

Начальный шаг: пока нет команды начинать варку (кнопка Start), нужно просто ждать. На SCF диаграмме начальный шаг расположен вверху и обозначается как прямоугольник с двойной рамкой. От него вниз уходит линия связи с горизонтальной черточкой (как бы закрытый шлагбаум). Это так называемый переход. Ему соответствует определенная логическая переменная или выражение: условие перехода. Когда условие перехода принимает значение TRUE, текущий шаг прекращает работу и передает эстафету (маркер) следующему шагу.

Первый шаг: нужно нагреть до кипения литр воды. Для этого выполняется действие - включение нагревателя (выход Out1). Действия, если они есть, показываются в прямоугольниках справа от шага. Они собственно и определяют полезную работу, связанную с шагом. Условием перехода в этом шаге служит вклю-

чение датчика кипения (Inp1). По нему происходит переход на третий шаг.

Второй шаг: допустим, датчик кипения не сработал после 5 мин нагревания. Налицо аварийная ситуация. Ее обрабатывает шаг 2. Он отключает нагреватель и просто стоит в ожидании рестарта программы. Иным способом уйти из него невозможно, поскольку в условии его перехода задана логическая константа FALSE. Чтобы после первого шага попасть в данный шаг, в программе предусмотрен альтернативный переход, начинающий отдельную ветвь

Третий шаг: должен высыпать стакан овсянки. Он открывает соответствующий клапан (Out2) и безусловно (константа TRUE в условии) передает управление дальше.

Четвертый шаг: должен варить 10 мин. Время и будет единственным условием следующего перехода. Варить нужно помешивая. Для этого действие шага удерживает выход мешалки включенным (Out3), пока шаг имеет активность.

Пятый шаг: выключает все выходы, которые ранее были включены, и передает управление начальному шагу.

Создайте указанную программу и визуализацию в среде Codey. Убедитесь в правильности их функционирования. Объясните назначение идентификаторов действий, используемых в программе.

3.2 Программа «сверлильный станок».

Напишем программу, позволяющую наглядно продемонстрировать особенности упрощенной версии языка SFC, входящей в состав CoDeSys.

Перед началом работы оператор с помощью тумблера выбора определяет режим сверления (Mode). После нажатия оператором кнопки запуска (Start) контроллер начинает управление станком. Подается команда опустить сверло (Down) и начинается обратный отсчет координаты (Y). При достижении нижней точки (Y=0) снимается команда на опускание и подается команда на сверление (Drill). Если выбран автоматический режим (Mode=true), то команда сверления снимается через 5 секунд. Если выбран ручной режим (Mode=false) то команда сверления снимается после нажатия оператором кнопки останова (Ready) сверления. Затем контролер подает команду на подъем сверла (Up) и начинает прямой отсчет координаты. После достижения верхнего положения (Y=70) команда подъема снимается.

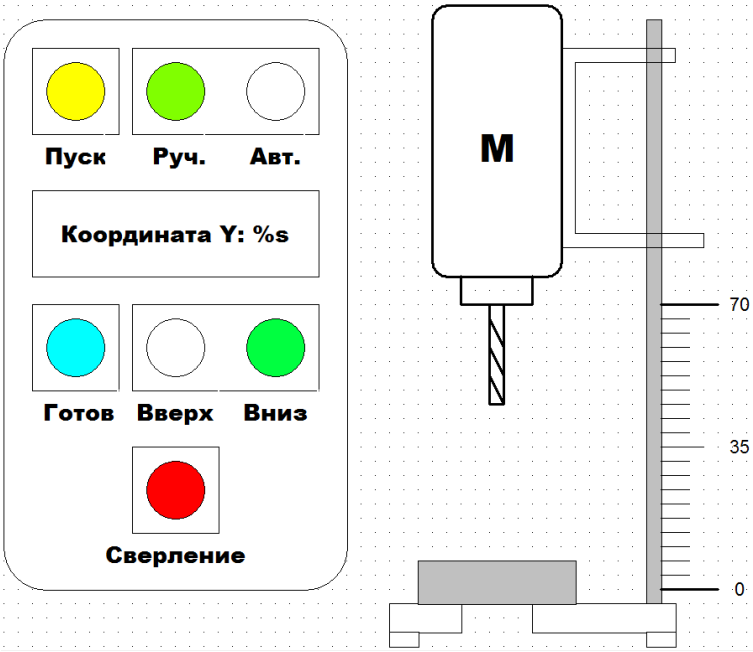


Рисунок 4 – Визуализация «сверлильный станок»

Для контроля времени выполнение шага необходимо в Атрибутах шага (щелчок правой кнопкой мыши по шагу) установить минимальное время выполнения шага (см. рис. 7).

Для обеспечения движения сверла на визуализации при изменении координаты Y необходимо в свойствах всех графических элементов, которые необходимо перемещать синхронно с изменением переменной Y программы на вкладке Положение установить Сдвиг по Y в зависимости от переменной Y (см. рис.8)

Для обеспечения движения сверла на визуализации при изменении координаты Y необходимо в свойствах всех графических элементов, которые необходимо перемещать синхронно с изменением переменной Y программы на вкладке Положение установить Сдвиг по Y в зависимости от переменной Y (см. рис.8)

Создайте указанную программу и визуализацию в среде Codey. Убедитесь в правильности их функционирования. Объясните назначение действий, используемых в программе.

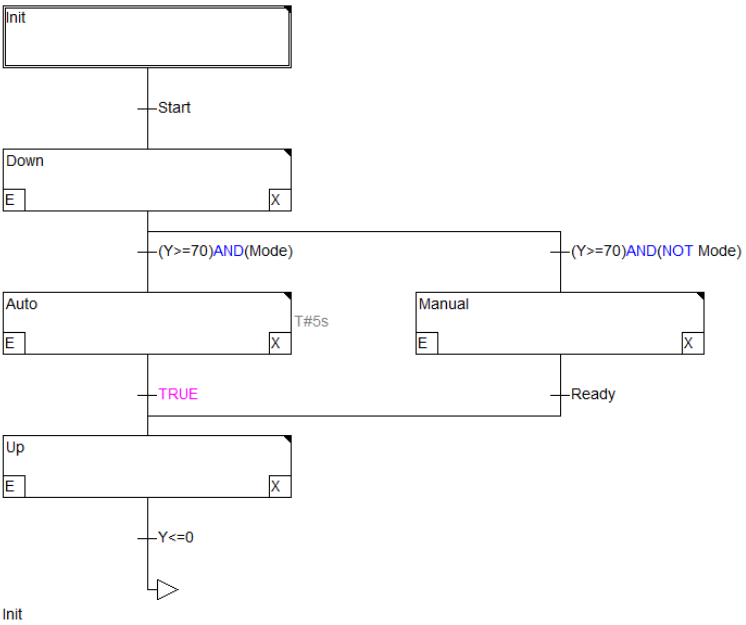


Рисунок 5 – Программа на упрощенном SFC «сверлильный станок»

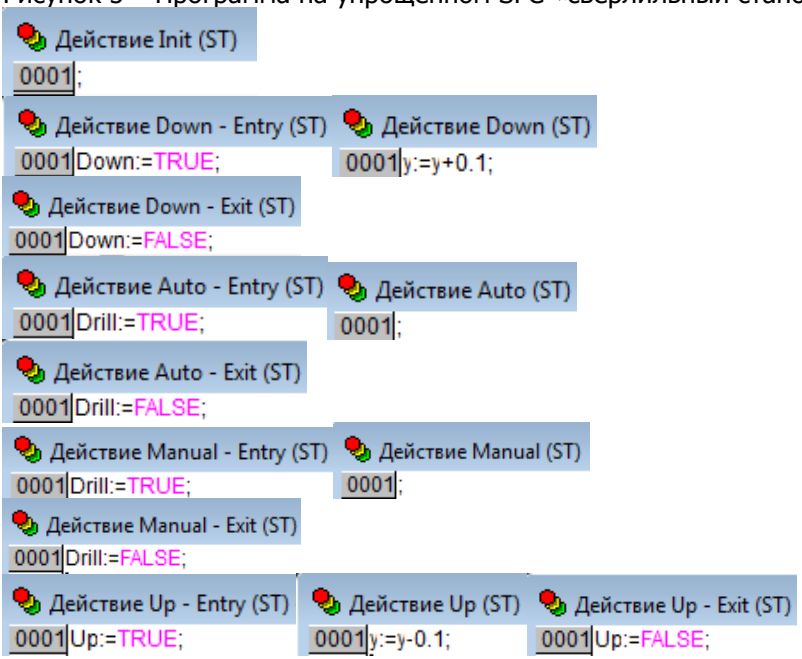


Рисунок 6 – Действия, используемые в программе «сверлильный станок»

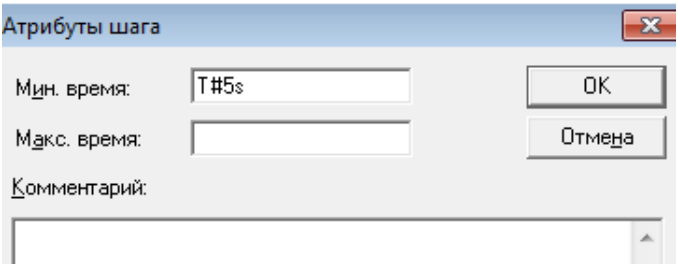


Рисунок 7 – Контроль времени выполнения шага

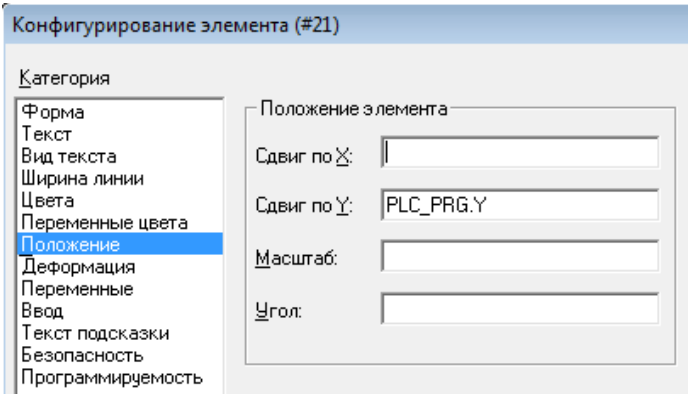


Рисунок 8 – Конфигурирование перемещения элемента на визуализации

3.3 Программа «светофор»

Написать программу работы автомобильного светофора на перекрестке. Светофор имеет три цветовых сигнала – зеленый, желтый и красный, включающихся в следующей последовательности:

1) 2 минуты – зеленый;
2) 2 секунды – моргающий зеленый;
3) 1 секунду – желтый;
4) 1 минуту – красный;
5) 1 секунду – желтый и красный.

Необходимо предусмотреть возможность досрочного включения пешеходом красного сигнала по кнопке пока горит зеленый свет. После нажатия кнопки идет переход к пункту 2. Предусмотреть визуализацию программы.

4 Контрольные вопросы:

1 В чем состоит основное преимущество автоматного подхода разработки программ для ПЛК? Каково назначение языка

SFC стандарта МЭК, упрощенного SFC? Реализует ли по вашему мнению язык SFC парадигму автоматного программирования?

2 В чем назначение действия SFC? Что такое идентификатор действия? Опишите последовательность выполнения SFC диаграммы в рабочем цикле ПЛК.

3 Что такое входное и выходное действия? Чем они отличаются от основного действия шага, когда их удобно использовать?

4 Какими способами можно контролировать время выполнения шага SFC диаграммы?

5 Содержание отчета:

1 Цель работы.

2 Результаты выполнения задания (программы с комментариями и визуализации).

3 Ответы на контрольные вопросы.

Создание функциональных блоков и работа с библиотеками в среде CoDeSys

1 Цель работы:

- изучение особенностей и принципов использования компонентов организации программ МЭК 61131;
- изучение принципов организации библиотек и библиотечных компонентов в среде CoDeSys;
- закрепление навыков программирования ПЛК на языках МЭК 61131.

2 Краткая теория

К компонентам организации программ (POU – Program Organization Unit) в МЭК-стандарте относятся функции, функциональные блоки и программы. Компонент обладает свойством инкапсуляции — работает как «черный ящик», скрывая детали реализации. Для работы с компонентом достаточно знать его интерфейс, включающий описание входов и выходов. Внутреннее его устройство знать необязательно. Благодаря инкапсуляции компоненты успешно решают задачу структурной декомпозиции проекта. На верхнем уровне представления мы работаем с крупными компонентами. Каждый из них выполняет значительную для данного проекта задачу. Лишние подробности на этом уровне только мешают пониманию проблемы. Раскрывая вложенные компоненты один за другим, мы можем добраться до самого детального представления. Готовый компонент всегда можно вскрыть, изучить и поправить. Это, конечно, относится только к пользовательским компонентам и открытым библиотекам.

Еще одной задачей, решаемой компонентами, является локализация имен переменных. Это означает, что в различных компонентах можно использовать повторяющиеся имена. Область видимости локальных переменных определяется рамками одного компонента. Экземпляры функциональных блоков, объявленные внутри других компонентов, также обладают локальной областью видимости. Программы и функции всегда определены глобально.

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

входы VAR_INPUT, выходы VAR_OUTPUT и переменные типа вход-выход VAR_IN_OUT. Завершают раздел объявлений локальные переменные VAR. В функциях разделы VAR_OUTPUT и VAR_IN_OUT отсутствуют.

Функция — это программный компонент, отображающий множество значений входных параметров на выход. Функция всегда возвращает только одно значение. При объявлении функции указывается тип возвращаемого значения, имя функции и список входных параметров. Вызов функции производится по имени с указанием значений входных параметров. Функция может использоваться в математических выражениях наряду с операторами и переменными. Функция не имеет внутренней памяти. Это означает, что функция с одними и теми же значениями входных параметров всегда возвращает одно и то же значение. Функция — это чистый код. Многократное использование функции не приводит к повторному включению кода функции при компоновке.

Тип функции (тип возвращаемого значения) может быть любым из числа стандартных типов данных или типов созданных пользователем. Тело функции может быть описано на языках IL, ST, LD или FBD. Использовать SFC нельзя. Вызывать функциональные блоки и программы из функций нельзя.

Функциональный блок — программный компонент, отображающий множество значений входных параметров на множество выходных. После выполнения экземпляра функционального блока все его переменные сохраняются до следующего выполнения. Следовательно, функциональный блок, вызываемый с одними и теми же входными параметрами, может производить различные выходные значения. Сохраняются все переменные, включая входные и выходные. Так, если мы вызовем экземпляр функционального блока, не определяя значения некоторых входных параметров, он будет использовать ранее установленные значения. Извне доступны только входы и выходы функционального блока, получить доступ к внутренним переменным блока нельзя.

Прежде чем использовать функциональный блок, необходимо создать его экземпляр. Эта операция аналогична по смыслу объявлению переменной. Описав новый блок, мы фактически создали новый тип данных подобный структуре. Каждый функциональный блок может иметь любое количество экземпляров.

После создания экземпляра функционального блока можно сразу начать работать с его данными. При этом совсем не обязательно вызывать его. Обращаться к переменным экземпляра можно так же, как к элементам структуры данных, через точку:

ctuTimeMeter.RESET := FALSE.

Операция начальной инициализации переменных функционального блока производится по сбросу, который выполняется непосредственно после загрузки проекта в память ПЛК, по команде отладчика или при перезапуске контроллера. Возможны случаи, когда экземпляру функционального блока нужна разумная инициализация. Например, для настройки блока необходимо провести некоторые вычисления. Специальной процедуры инициализации в функциональных блоках не предусмотрено. Здесь придется потратить на инициализацию один или несколько первых циклов выполнения экземпляра. Окончание сложной процедуры инициализации индицируют обычно выходом готовности (ENO).

Программа — глобальный программный элемент, отображающий множество значений входных параметров на множество выходных. Программа очень похожа на функциональный блок. Из всех программных компонентов МЭК-программа самый крупный. При помощи программ определяется верхний уровень проекта и реализуется управление многозадачностью. Программы являются глобальными компонентами и объявляются на уровне ресурсов.

Библиотека - сборник готовых подпрограмм или объектов, используемых для разработки программного обеспечения. В CoDeSys все файлы библиотек имеют расширения *.lib (Library) и находятся в папке Library. По умолчанию подключен (доступен) стандартный набор библиотек. Дополнительные библиотеки добавляются пользователем по мере необходимости в папку к уже имеющимся.

Для подключения новых библиотек к проекту соответствующие файлы переписываются пользователем в ту же папку, где находятся все используемые библиотеки. Для внутренних библиотек можно все данные, которые были включены в информацию о проекте. Для внешних библиотек отображается имя библиотеки и путь к ее файлам. Библиотека "standard.lib" доступна всегда. Она содержит все функции и функциональные блоки, требуемые стандартом МЭК 61131-3. Стандартные компоненты признаются неявно системой программирования. Исходный текст этих POU находится в С-библиотеке и является компонентом CoDeSys.

3 Задание

3.1 Функция «2 из 3-х»

Добавим в проект функцию с названием Decoder_FUN (см. рис.1 и рис.2), значение которой изменяется в зависимости от значений трех аргументов (d1, d2, d3). При активном (TRUE) зна-

чении двух любых аргументов Decoder=TRUE, в противном случае Decoder=FALSE.

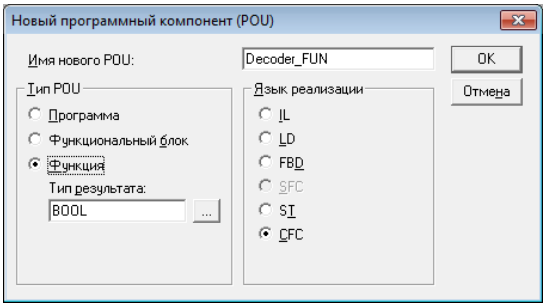


Рисунок 1 – Добавление функции в проект

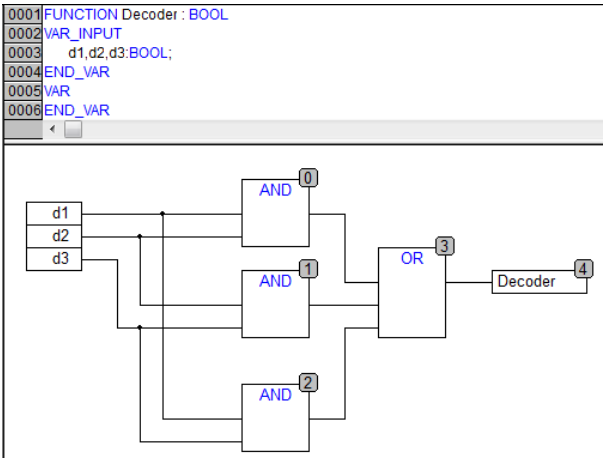


Рисунок 2 – Функция «Decoder_FUN»

3.2 Функциональный блок «2 из 3-х»

Добавим в проект функциональный блок с названием Decoder_FB (см. рис.1 и рис.2), которой имеет три входных переменных (d1, d2, d3) и две выходных – Dec и Dec_Inv. При активном (TRUE) значении двух любых аргументов Dec=TRUE и Dec_Inv=FALSE, в противном случае Dec=FALSE и Dec_Inv=TRUE.

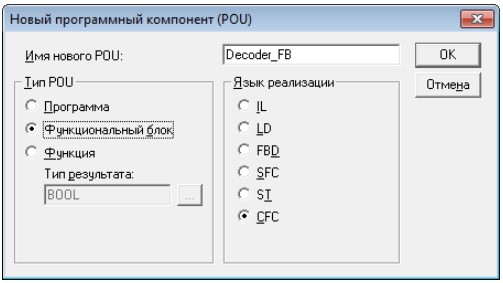


Рисунок 3 – Добавление функционального блока в проект

```
0001 FUNCTION_BLOCK Decoder_FB
0002 VAR_INPUT
0003     d1,d2,d3: BOOL;
0004 END_VAR
0005 VAR_OUTPUT
0006     Dec, Dec_Inv:BOOL;
0007 END_VAR
0008 VAR
0009 END_VAR
```

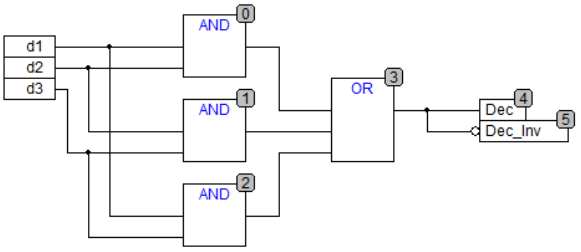


Рисунок 4 – Функциональный блок «Decoder_FB»

Добавим в основную программу проекта только что созданные компоненты: две копии одного экземпляр функционального блока и две функции. Свяжем выходные переменные компонентов с реальными выходами контроллера (out1, out2, out3, out4). Соединим выходы как показано на рис. 5. Зафиксируем состояние выходов элементов программы и реальных выходов контроллера. Соединим выходы как показано на рис. 6. Зафиксируем состояние выходов элементов программы и реальных выходов контроллера**.** *Сделать выводы о различиях в работе функции и функционального блока.*

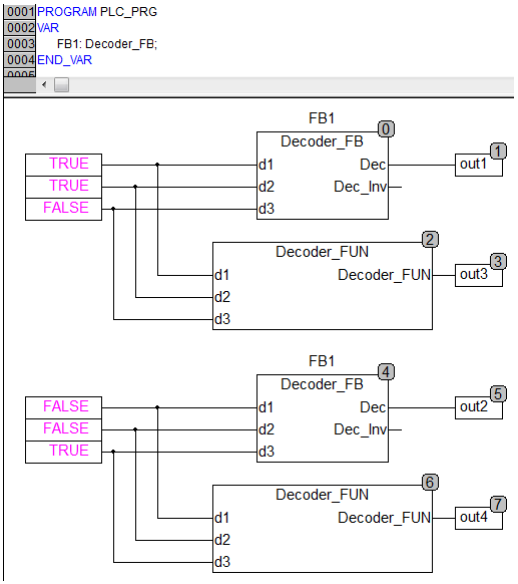


Рисунок 5 – Основная программа (вариант 1)

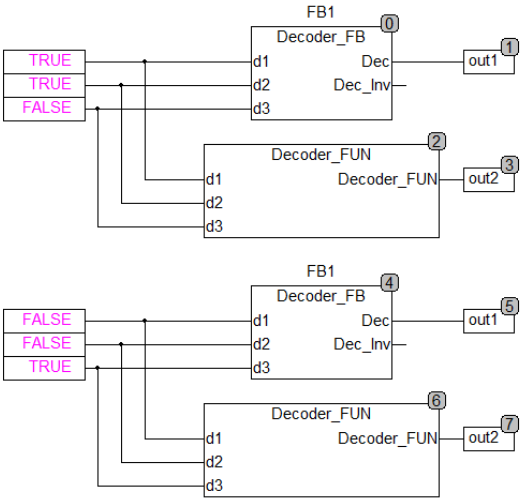


Рисунок 6 – Основная программа (вариант 2)

Сохраним созданные компоненты в собственную библиотеку с именем decoder.lib. Для этого необходимо в меню Файл выбрать пункт Сохранить как и в качестве типа файла установить Внутренняя библиотека (*.lib).

3.3 Программа «управление насосами»

В помещении насосной станции работают две группы насосов. В каждой группе имеется основной (n1, n3) и резервный (n2, n4) насос. Все насосы работают на наполнение емкости, имеющей три дискретных датчика уровня (d1, d2, d3). При срабатывании любых двух датчиков одновременно происходит переключение основного и резервного насосов в каждой группе.

Для описания указанного алгоритма переключения насосов создадим функциональный блок с именем Nasos на языке ST (см. рис. 7). Условимся, что на вход Decoder значение TRUE поступает после появления переднего фронта сигнала (по срабатыванию датчиков). Для управление активностью насосов в разделе локальных переменных ФБ добавим переменную num типа WORD.

```
0002 VAR_INPUT
0003     Decoder: BOOL;
0004 END_VAR
0005 VAR_OUTPUT
0006     out1, out2: BOOL;
0007 END_VAR
0008 VAR
0009     num: WORD:=1;
0010 END_VAR
```

```
0001 IF Decoder=TRUE THEN
0002 IF num=1 THEN
0003 num:=2;
0004 ELSE
0005 num:=1;
0006 END_IF;
0007 END_IF;
0008 out1:=num.0;
0009 out2:=num.1;
```

Рисунок 7 – ФБ Decoder

Для установления факта срабатывания датчиков воспользуемся созданным в п.п. 3.2 блоком Decoder. Для этого необходимо (см. рис. 8) подключить соответствующую библотеку (Ресурсы> Менеджер библиотек).

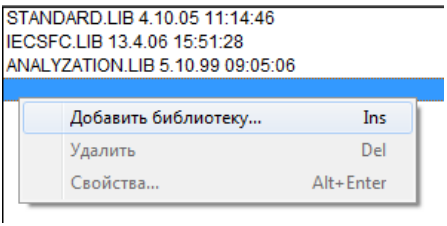


Рисунок 8 – Добавление дополнительной библиотеки в проект

Добавим созданные функциональные блоки в основную программу (см. рис. 9). Для детектирования переднего фронта сигнала с выхода блока Decoder воспользуемся стандартным блоком F_TRIG.

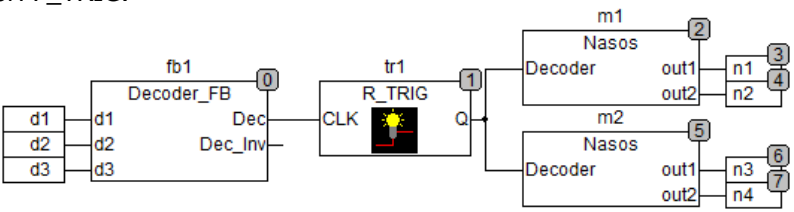


Рисунок 9 – Программа «управление насосами»

Запустим программу на исполнение и убедимся в правильности алгоритма ее функционирования.

3.4 Управление системой пожарной сигнализации

В здании имеется две комнаты. В каждой из комнат установлены три релейных датчика пожарной сигнализации (d1, d2, d3). При срабатывании одного датчика для каждой комнаты загорается сигнальная лампа (lamp), установленная снаружи комнаты. При срабатывании сразу двух датчиков в любой из комнат загорается сигнальная лампа и включается общий сигнал тревоги (alarm). Имеется возможность ручного включения и отключения тревоги, для этого снаружи каждой из комнат есть кнопки Сброс и Установка (reset, set).

Необходимо написать программу, реализующую указанный алгоритм и сиротствующую визуализацию. При этом алгоритм срабатывания сигнализации для каждой комнаты описать при помощи пользовательского функционального блока Room.

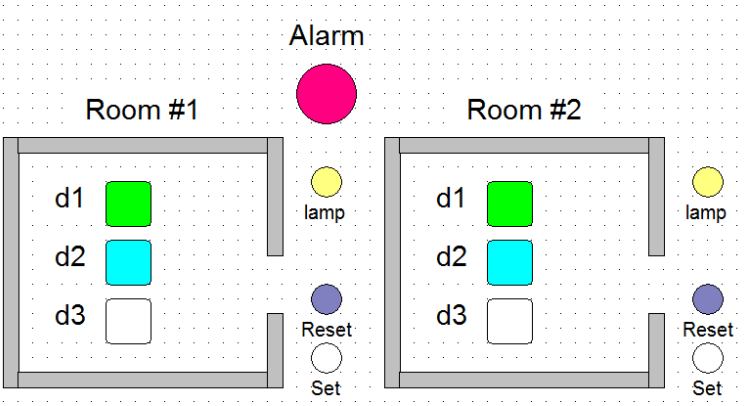


Рисунок 10 – Визуализация системы пожарной сигнализации

4 Контрольные вопросы:

1 В чем состоит преимущество использования целочисленных переменных с побитным обращением (WORD, BYTE) и др. при работе с дискретными выходами ПЛК?

2 Что такое интерфейс компонента POU?

3 В чем состоят особенности переменных интерфейса VAR IN?

VAR OUT? VAR IN_OUT? VAR?

4 В чем заключается основные отличия функции и функционального блока МЭК 61131?

5 Какие способы используется для вызова функции? Функционального блока?

5 Содержание отчета:

1 Цель работы.

2 Результаты выполнения задания (программы с комментариями и визуализации).

3 Ответы на контрольные вопросы.

- изучение особенностей автоматного подхода для программирования ПЛК на практических примерах;
- изучение принципов программирования ПЛК на языке SFC;
- закрепление навык

Работа с файлами, использование задач в среде CoDeSys

1 Цель работы:

- изучение возможностей библиотеки SysLibFile.lib для работы с файлами в среде CoDeSys;
- изучение средств управления задачами в среде CoDeSys;
- закрепление навыков программирования ПЛК на языках МЭК 61131.

2 Краткая теория

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

CoDeSys содержит специальный инструмент — менеджер задач (Task configuration), представляющий задачи и их программы в виде иерархического дерева. В любом проекте всегда существует, как минимум, одна задача. По умолчанию это циклическая задача, вызываемая в каждом рабочем цикле ПЛК. В CoDeSys она включает единственную программу PLC_PRG. Каждая задача обладает определенным приоритетом. Приоритет определяется числом от 0 до 32. Чем меньше число, тем выше приоритет. Если две или более задачи должны получить управление одновременно, то побеждает задача с более высоким приоритетом. При одинаковом приоритете управление получает задача, имеющая большее время ожидания. То есть две равно приоритетные задачи будут работать поочередно.

В системе исполнения CoDeSys реализована не вытесняющая многозадачность. Это означает, что любая задача, даже более приоритетная, дает доработать текущей задаче до конца одного рабочего цикла. Работа циклических задач является аппаратно независимой.

Процесс **записи и чтения файла** разделяется на три части: открытия файла (для чтения или записи), запись или чтение,

закрытие файла. В CoDeSys это можно сделать функциями из состава библиотеки SysLibFile.lib. Функции эти имеют несложные параметры, но при их использовании надо учесть, что реакция системы контроллера на каждую команду (вызов функции) может быть больше времени перехода от одной команды к другой, т.е. функции не блокирующие. Они возвращают управление раньше, чем ОС завершит выполнение соответствующей операции. В связи с этим чтобы не было неожиданных проблем с их использованием, необходимо проверять завершение одной функции перед выполнением другой.

3 Задание

3.1 Работа с файлами

Для открытия, записи и закрытия файла необходим последовательный вызов функций SysFileOpen (открытие файла), SysFileWrite (запись в файл), SysFileClose (закрытие файла).

С параметрами этих функций можно познакомиться через справочную систему редактора CoDeSys.

1) Перейдем в окно объявлений глобальных переменных (вкладка Ресурсы > Глобальные переменные) и запишем:

```
VAR_GLOBAL
StatusOfWriting, StatusOfReading: INT; (*переменные статуса чтения и записи*)
WriteBuffer1 : STRING[12]:='Hallo World!';(*строка для записи в файл*)
ReadBuffer1: STRING[12]; (*Строка для чтения из файла*)
END_VAR
```

Эти переменные мы можем изменять из любого компонента проекта, тем самым инициировать запись или чтение.

2) Создадим основную программу, которую теоретически необходимо вызывать один раз в 50 мс (время выбирается в зависимости от объема записи) Эта программа за один вызов будет выполнять только одну функцию. За два последовательных вызова через 50 мс программа выполнит две функции (чтение и запись). Для этого создаем проект на языке ST. Время выполнения программы будет задано позже.

Зададим код основой программы:

```
PROGRAM PLC_PRG
VAR
i, i1: BOOL;
x: INT;
END_VAR
x:=x+1;
IF i1 THEN StatusOfReading:=1; i1:= FALSE; END_IF;
IF NOT i THEN StatusOfWriting:=1; i:=TRUE; i1:=TRUE;
END_IF;
```

Используя два флага i и i1 получаем в первом цикле отработку нижней строки (произойдет запись) во втором цикле отработку строки выше (произойдет чтение). Во всех циклах производим инкремент переменной x, для наблюдения работы процесса.

3) Добавим в проект еще одну программу на языке ST и назовем ее WriteFile. Содержание программы следующее:

```
VAR
f: DWORD;
pChar: POINTER TO BYTE;
sizebuf: INT;
statusf: DWORD;
END_VAR
CASE StatusOfWriting OF
1: f:=SysFileOpen('b:\myfile.txt','w'); (*Открываем файл для записи*)
pChar:=ADR(WriteBuffer1); (*получаем адрес начала строки*)
sizebuf:=SIZEOF(WriteBuffer1);(*получаем размер строки*) StatusOfWriting:=2; (*приготовились к следующему действию в следующем цикле*)
2: statusf:=SysFileWrite(f,pChar,sizebuf);(*запись в файл*) StatusOfWriting:=3; (*приготовились к следующему действию в следующем цикле*)
3: SysFileClose(f); (*закрываем файл*) StatusOfWriting:=0; (*состояние - запись окончена*) END_CASE;
```

Программа записи в файл готова, ее также необходимо вызывать циклически через каждые 50 мс.

4) Добавим в проект программу на языке ST и назовем ее ReadFile. Чтение от записи отличается только названием функций. Интервал вызова у программы такой же как и в предыдущем слу-

чае. Содержание программы следующее:

```
PROGRAM ReadFile
VAR
f:DWORD;
pChar: POINTER TO BYTE;
sizebuf:INT;
statusf:DWORD;
END_VAR
CASE StatusOfReading OF
1: f:=SysFileOpen('b:\myfile.txt','r');
pChar:=ADR(ReadBuffer1);
sizebuf:=SIZEOF(ReadBuffer1);
StatusOfReading:=2;
2: statusf:=SysFileRead(f,pChar,sizebuf);
StatusOfReading:=3;
3: SysFileClose(f);
StatusOfReading:=0;
END_CASE;
```

Конечно, можно было заставить одну задачу вызывать обе программы, и на запись и на чтение, но в учебных целях, разведем их по разным задачам, чтобы показать некоторые проблемы с такого рода многозадачностью.

5) Переходим на вкладку Ресурсы> Конфигуратор Задач и в появившемся окне слева в любом месте правой клавишей мышки добавляем задачу и называем ее taskWriteFile. Ставим галочку Циклически и в поле интервал прописываем T#50ms.

Щелчком ПК мыши на этой задаче добавляем Вызов Программы WriteFile (рис. 1). Приоритет у данной задачи должен быть меньше, чем у задачи, которая будет вызывать основную программу PLC_PRG. Например, 15.

Необходимо помнить, что при активации механизма задач в CoDeSys программа PLC_PRG перестает быть обязательной и работать не будет пока мы ее не вызовем искусственно. Сделаем это аналогичным WriteFile образом при помощи отдельной задачи.

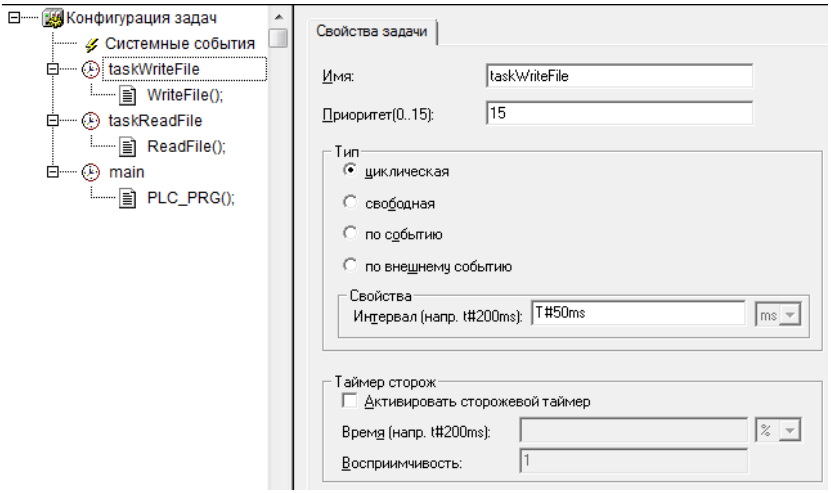


Рисунок 1 – Конфигурация задач

Обратите внимание на время вызова T#300ms программы PLC_PRG. Для чего установлен такой период узнаем ниже. Записывать программа начнет только в случае если StatusOfWriting получит значение 1. В этом случае программа начнет отрабатывать в каждом цикле по одному шагу конструкции CASE и в конце сбросит StatusOfWriting в 0, что будет означать конец процедуры записи. В нулевом состоянии StatusOfWriting, как и во всех остальных состояниях кроме 1,2,3, программа WriteFile будет вызываться в «холостую».

В итоге мы получаем 3 программы. Одна для записи WriteFile, вторая для чтения ReadFile и третья PLC_PRG. Чтобы программы записи и чтения выполнили свои функции надо соответствующим переменным StatusOfWriting и StatusOfReading присвоить значения 1. Это можно сделать из PLC_PRG. Но представьте: если программа чтения будет вызвана со статусом на чтение файла в то момент, когда программа WriteFile еще не закончила свое действие с этим файлом, у нас получиться неожиданная ошибка. При работе в нескольких задачах с общими ресурсами надо быть очень внимательным на последовательность использования этих общих ресурсов. Очевидно, можно применить методы защитного программирования, но так как наш пример далек от этих методов, мы в одном цикле PLC_PRG установим параметр StatusOfWriting:=1; во втором цикле StatusOfReading:=1.

Сделаем интервал вызова PLC_PRG достаточно большим, чтобы программы записи и чтения могли выполнить свои дей-

ствия полностью. Время работы всех задач равно 150 мс (основная программа, чтение и запись). Укажем двукратный запас в 300 мс, на что мы ссылались выше.

6) Загрузим программу в контроллер и запустим на исполнение.

Видим по растущему значению x, что происходит периодический вызов нашей PLC_PRG. В первом цикле (и только в первом) была произведена запись в файл переменной WriteBuffer1, во втором чтение в переменную ReadBuffer1. В остальных циклах идёт только инкремент x. Результат можно увидеть, перейдя в окно глобальных переменных. Значение WriteBuffer1 будет присвоено ReadBuffer1.

7) Если мы с помощью ПЛК-Браузера (Вкладка Ресурсы) зайдём и посмотрим диск ПЛК (Подключится к ПЛК и ввести команду filedir), то обнаружим там файл с соответствующим именем. Размер файла будет изменяться в зависимости от объёма записываемых в него данных.

3.2 Пример «стиральная машина».

Самый простой случай, когда многозадачность может быть полезна – это моделирование объекта управления. Классическая задача на применение языка SFC – это стиральная машина, основные этапы работы которой: контроль закрытия дверки, ожидание нажатия кнопки пуск, наполнение бака водой, нагрев, стирка, слив воды. Добавим также в нашу машину несколько режимов стирки, отличающихся температурой, временем стирки и количеством используемой воды. Для описания режима определим структуру WashPara (На вкладке Типы данных):

```
TYPE WashPara :
STRUCT
    WaterIn: TIME;
    WashTime: TIME;
    WashTemp: WORD;
    WaterOut: TIME;
END_STRUCT
END_TYPE
```

Создадим в проекте новый POU типа программа (PROGRAM), назовём его Wash и выберем упрощённый язык SFC.

```
PROGRAM Wash
VAR CONSTANT
    maxProgNr :BYTE:= 5;
    END_VAR
```

```
VAR_INPUT
        SFCInit: BOOL;
        MyPara:WashPara;
        MyPara1:WashPara;
        MyWashPara: ARRAY [0..maxProgNr] OF WashPara;
END_VAR
VAR
        SFCCurrentStep: STRING;
       SFCErrorAnalyzationTable: ARRAY [0..15] OF ExpressionResult;
        SFCError: BOOL;
        i: BYTE;
        CurStep: WORD;
        hFile: DWORD;
END_VAR
```

В разделе объявлений Wash объявим константу для числа режимов: maxProgNr :BYTE:= 5. Идея состоит в том, чтобы везде, где потребуется количество режимов, использовать maxProgNr, это позволит при необходимости легко изменять их число в единственном месте программы. Для хранения режимов объявим массив структур: MyWashPara

Далее мы хотим связать длительность выполнения определённых шагов SFC со значениями элементов массива структур, так чтобы мы смогли динамически влиять на работу диаграммы, изменяя только значения переменных. Вероятно, понадобятся таймеры или даже массив таймеров. Все гораздо проще, используем уже знакомые нам атрибуты шагов, а именно минимальное время активности шага. Привязка элементов массива к временам шага показана на рис. 3. Переходы после выполняемых по времени шагов всегда разрешены, то есть им присвоены константы TRUE

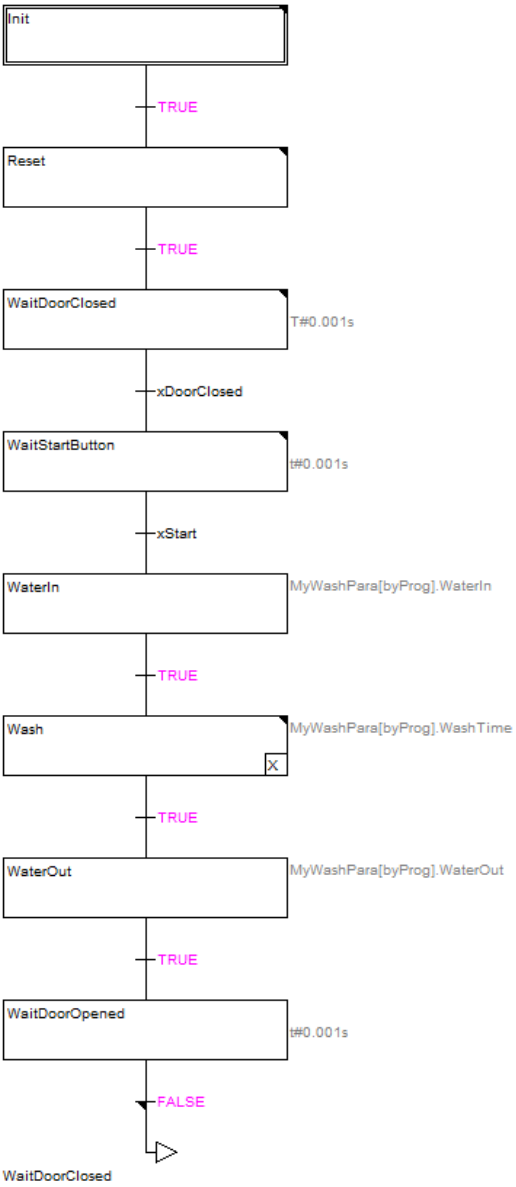


Рисунок 2– Программа Wash 8

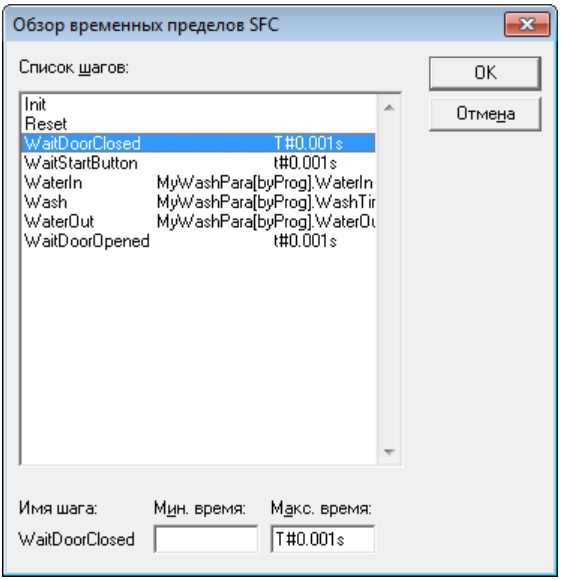


Рисунок 3 – Обзор времени выполнения шагов SFC - диаграммы

При этом для шагов WaitDoorClosed и WaitDoorOpend указываем максимальное время ожидания (выполнения шага), а для остальных шагов – минимальное время.

Для использования во всех POU проекта дополнительно введём следующие глобальные переменные:

```
VAR_GLOBAL
        xDoorClosed: BOOL; (*Дверь закрыта*)
        xStart: BOOL;    (*Нажата кнопка Пуск*)
        xLoad : BOOL;    (*Нажата кнопка Считать*)
        xSave: BOOL;     (*Нажата кнопка Сохранить*)
        xReset : BOOL;   (*Нажата кнопка Сброс*)
        xWaterOn : BOOL;  (*Вода залита*)
        xPump : BOOL; (*Помпа слива воды*)
        xMotOn: BOOL; (*Мотор включен*)
        byProg : BYTE; (*Режим*)
        xProgInc: BOOL;(*Кнопки выбора режимов*)
        xProgDec: BOOL;
        ProgInc: R_TRIG;
        ProgDec: R_TRIG;
        xActPrgCng: BOOL;
END_VAR
```

Элемент массива MyWashPara[byProg].WaterIn задает длительность шага WaterIn в зависмости от переменной выбора режима (byProg), MyWashPara[byProg].WashTime – для шага WashTime, MyWashPara[byProg].WaterOut - для шага WaterOut.

Для наглядной демонстрации работы нашей машины удобно использовать встроенную графическую визуализацию CoDeSys. Её возможный вариант показан на рис. 4.

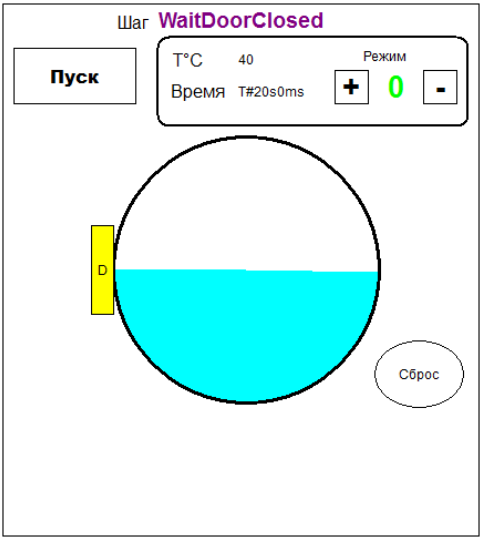


Рисунок 4 – Визуализация «стиральная машина»

Наша визуализация слишком уж статична, хотелось бы отобразить вращение барабана. Это можно сделать при помощи закрашенного сегмента. Напишем отдельную программу *Motor*, ее задача плавно циклически изменять переменную *wAngle* от 0 до 360 градусов, которую мы свяжем с углом сектора соответствующего элемента визуализации. Например, так:

```
IF xMotOn THEN
wAngle := wAngle + 5;
IF wAngle > 359 THEN
wAngle := 0;
END_IF
ELSE
wAngle := 180;
END_IF
```

Как видите, здесь нет таймеров, обеспечивающих достаточно медленную для зрительного восприятия скорость вращения барабана. Мы можем поступить проще, создадим для целей моделирования циклическую задачу с достаточно медленным циклом работы, который легко регулировать. После отладки эту задачу можно будет просто отключить, не изменяя ничего в основных программах. По умолчанию CoDeSys создает новый проект с единственной программой PLC_PRG, которая неявно включена в единственную циклическую задачу. Чтобы сделать многозадачный проект, необходимо использовать конфигуратор задач (Task configuration), расположенный на вкладке 'Ресурсы' менеджера проектов. Щелкните правой кнопкой мыши по элементу Task configuration и в появившемся контекстном меню выберете команду Append task. Сначала создадим главную задачу, она будет назваться *Main*, ее тип – циклический (cyclic) и интервал работы T#10ms. Аналогичным способом откройте контекстное меню для нашей новой задачи и добавьте в нее (Append program call) программу PLC_PRG();. Теперь создайте еще одну циклическую задачу *MotorSim* с интервалом T#100ms, в нее вставьте вызов программы Motor();.

Содержание отчета:

1 Цель работы.

2 Результаты выполнения задания (программы с комментариями и визуализации).

3 Ответы на контрольные вопросы.

СПИСОК ЛИТЕРАТУРЫ

1. И.В. Петров. Программируемые контроллеры: стандартные языки и приемы прикладного проектирования. Солон-Пресс, 2008 г. -256 с.
2. Минаев И.Г. Программируемые логические контроллеры: практическое руководство для начинающего инженера. Агрус, 2009 г. – 100с.
3. Программируемые устройства ОВЕН http://www.owen.ru