МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

С.В. Ефимов, М.И. Пушкарев, А.С. Фадеев

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ АВТОМАТИЗИРОВАННЫХ СИСТЕМ УПРАВЛЕНИЯ ТЕХНОЛОГИЧЕСКИМИ ПРОЦЕССАМИ

*Рекомендовано в качестве учебного пособия
Редакционно-издательским советом
Томского политехнического университета*

Издательство
Томского политехнического университета
2020

УДК 004.896:004.42(075.8)
ББК 32.966/5:32.972я73
Е91

Ефимов С.В.

Е91 Программное обеспечение автоматизированных систем управления технологическими процессами : учебное пособие / С.В. Ефимов, М.И. Пушкарев, А.С. Фадеев ; Томский политехнический университет. – Томск : Изд-во Томского политехнического университета, 2020. – 128 с.

ISBN 978-5-4387-0919-0

Пособие разработано с целью изучения автоматизированных систем управления технологическими процессами, их классификации, структуры, архитектуры, программируемых логических контроллеров и программного обеспечения, используемых при проектировании и автоматизации технологических процессов на производствах. Также рассматривается практическая сторона программирования логических контроллеров в программном пакете CoDeSys на языках стандарта МЭК 61131-3.

Предназначено для студентов, обучающихся по направлению 15.03.04 «Автоматизация технологических процессов и производств».

УДК 004.896:004.42(075.8)
ББК 32.966/5:32.972я73

Рецензенты

Заместитель директора Научно-технической компании «АККО»
В.Н. Шатрова

Главный специалист отдела систем автоматизации
Проектного института ООО «Элком+»
А.В. Поливанов

ISBN 978-5-4387-0919-0

ВВЕДЕНИЕ

В настоящее время развитие технического, экономического, социального и других направлений невозможно представить без автоматизации тех или иных процессов. Автоматизация таких процессов строится на базе автоматизированных систем управления, сочетающих в себе различные датчики, приборы, исполнительные механизмы, агрегаты, микропроцессорные устройства и т. д.

Автоматизированная система управления (АСУ) – это комплекс программно-аппаратных средств, обеспечивающий автоматизированный сбор информации, ее обработку, передачу и хранение, необходимой для оптимизации управления в различных областях жизнедеятельности человека согласно принятым критериям. Сегодня АСУ применяются в различных сферах жизнедеятельности человека: в энергетике, транспорте, промышленности и т. п. Термин «автоматизированная» означает использование человеческого ресурса в такой системе. Другими словами, часть функций управления тем или иным процессом возлагается на человека.

Опыт, накопленный человечеством при разработке и внедрении АСУ, показывает, что управление такого рода процессами основывается на ряде правил, критериев, законов и требований к таким системам управления.

Пособие посвящено разработке автоматизированных систем управления, включающих в себя классификацию и виды АСУ, требования к АСУ согласно стандартам, архитектуру АСУ, программируемые логические контроллеры (ПЛК), предназначенные для автоматизации процессов, программное обеспечение (ПО) для ПЛК и рабочих мест операторов, управляющих различными процессами.

1. КЛАССИФИКАЦИЯ АСУ

В настоящее время в литературе, посвященной АСУ, можно встретить распространенную классификацию, согласно которой их делят на два класса: информационные и управляющие системы. В каждой из таких систем человеку, ПЛК, операторской станции (ОС) в процессе управления отводится своя роль.

1.1. Информационные системы

Информационные системы обеспечивают сбор, обработку и выдачу измеренных информационных сигналов о ходе технологического или производственного процесса в виде, удобном для отображения: таблицы, мнемосхемы, графики, тренды и т. д. На базе полученных данных и соответствующих расчетов оператор или система управления вычисляют, какие необходимо произвести управляющие воздействия на процесс для достижения наиболее оптимальных и благоприятных условий его протекания. В свою очередь, полученная информация может служить рекомендациями для оператора о ходе ведения процесса. Для такого класса систем именно человек играет главную роль, принимает решения по управлению процессом на базе знаний, предоставленных информационной АСУ.

Основное назначение таких систем – сбор оператором информации о ходе технологического или производственного процесса с высокой точностью, что позволяет оператору предпринимать решения для эффективного управления процессом.

Информационные системы представляют информацию о ходе протекания процесса и информацию о ситуациях, вызванных отклонениями значений параметров процесса от нормы.

Информационные системы подразделяются на пассивные и активные. Задачей пассивных информационных систем является сбор информации о ходе протекания процесса для оператора по его запросу, а активные информационные системы предоставляют данные оператору через заданные промежутки времени или периодически и могут включать рекомендации по управлению процессом.

1.2. Управляющие системы

Управляющие системы наряду со сбором и обработкой информации обеспечивают выработку управляющих воздействий непосредственно на исполнительные механизмы или исполнителям команд. Выработка управляющих воздействий на исполнительные устройства осуществляется в режиме реального времени. Иными словами, в том же темпе, что

и характер протекания технологического или производственного процесса. В управляющих системах основную роль по выработке управляющих воздействий исполняют вычислительные средства.

Контрольные вопросы и задания

1. Сформулируйте основное назначение информационных систем.
2. На какие виды подразделяются информационные системы?
3. Какой вид информационных систем может содержать рекомендации для оператора по управлению процессом?
4. Сформулируйте основное назначение управляющих систем.
5. Охарактеризуйте скорость выработки управляющих воздействий в управляющих системах.

2. ВИДЫ АСУ

АСУ можно разделить на несколько видов в зависимости от их назначения и применения: автоматизированная система управления технологическим процессом (АСУ ТП), автоматизированная система управления производством (АСУ П), интегрированная автоматизированная система управления (ИАСУ).

Каждая из таких систем решает определенные задачи, возлагаемые на нее, и обладает рядом функций и свойств.

2.1. АСУ ТП

АСУ ТП – это система, обеспечивающая сбор и обработку информации, и выработку на основе полученной информации о технологическом процессе управляющих воздействий на объект управления в соответствии с установленными критериями управления.

На АСУ ТП возлагаются задачи оперативного управления и контроля состояния объекта автоматизации, технологического процесса и т. д.

АСУ ТП включает в себя программно-аппаратный комплекс, обеспечивающий автоматизацию технологических операций какого-либо производства в целом или его отдельных участков.

В состав АСУ ТП могут быть включены системы автоматического управления (САУ), а также микропроцессорные устройства, решающие локальные выделенные задачи управления тем или иным объектом. Все эти автоматизированные устройства, включая ПЛК, связываются в единый комплекс. АСУ ТП имеет уровневую структуру и, как правило, на верхнем уровне располагается ОС, в которой собирается информация о ходе протекания технологического процесса, в различных формах (тренды, таблицы, графики, мнемосхемы и др.); осуществляется хранение информации. Через интерфейс ОС оператор может дистанционно управлять технологическим процессом. В зависимости от сложности реализуемой АСУ ТП и ее структуры в системе может наблюдаться несколько ОС, например при решении таких задач, как резервирование или обслуживание АСУ ТП с отдельной инженерной станции и т. д. Для информационной связи всех локальных узлов и подсистем (ПЛК, ОС, микропроцессорные устройства и пр.) используются промышленные.

Внедрение АСУ ТП на производстве позволяет решить ряд задач:

1) обеспечить безопасность протекания технологических процессов за счет внедрения противоаварийных защит, тем самым исключая человеческий фактор, несогласованность в действиях операторов или время их реакции;

2) сократить стоимость системы управления за счет сокращения числа локальных приборов, регистрирующих устройств и регуляторов (однако этот пункт выполняется не всегда);

3) снизить себестоимость, увеличить производительность и повысить качество выпускаемой продукции за счет автоматизации ряда процедур при производстве продукции и оптимизации режимов работы оборудования, задействованного в технологическом процессе;

4) реализовать функционирование сложных алгоритмов с использованием большого числа параметров технологического процесса, что возможно только при выполнении определенного числа задач ПЛК.

2.2. АСУ П

АСУ П – это система, основным назначением которой является решение задач управления, связанных с производственно-хозяйственной деятельностью предприятия.

Назначением АСУ П является решение задач организации производства: выполнение экономических задач (снабжение, реализация продукции, финансовые средства), не менее важную роль в АСУ занимает управление трудовыми ресурсами (подготовка приказов и распоряжений, расчет заработной платы, контроль за приемом и увольнением кадров). Посредством АСУ П осуществляется документооборот предприятия. В такой системе в качестве управляющих воздействий выступают различные документы: распоряжения, приказы, отчеты и т. д., а их реализация осуществляется непосредственно персоналом предприятия.

В состав задач АСУ П могут входить планирование выпускаемой продукции с учетом мощностей производства, анализ полученной продукции, моделирование процессов производства, реализация и сбыт продукции. Внедрение АСУ П ведет к повышению эффективности производственно-хозяйственной деятельности предприятия.

2.3. ИАСУ

Как АСУ ТП, так и АСУ П могут функционировать самостоятельно. Однако для повышения эффективности функционирования предприятия их могут объединять в единую систему – ИАСУ. ИАСУ сочетает в себе совместное управление технологией изготовления продукции или сырья и организацией производства в рамках всего предприятия. Внедрение ИАСУ позволяет наладить взаимосвязанное управление и контроль за технологическими и производственными процессами, оптимизировать деятельность предприятия в целом. Однако следует отметить, что в России таких предприятий, имеющих воз-

можность внедрить, немного. Это связано с необходимостью проведения глубокого реинжиниринга и высокими финансовыми затратами. Внедрение таких систем могут себе позволить такие организации, как Газпром, Роснефть, Норильский никель.

Далее в учебном пособии под АСУ будем рассматривать АСУ ТП.

Контрольные вопросы и задания

1. Перечислите виды АСУ.
2. Дайте определение понятию «АСУ ТП».
3. Какие задачи позволяет решать внедрение АСУ ТП?
4. Сформулируйте основное назначение АСУ П.
5. Что является управляющим воздействием в АСУ П?
6. С какой целью объединяют АСУ ТП и АСУ П в единую систему?

3. АРХИТЕКТУРА АСУ

Архитектура АСУ носит уровневый характер. На одном из уровней располагается полевое оборудование: чувствительные элементы, датчики, исполнительные механизмы, на другом – кросс-стойки, шкафы системы управления, включающие в свой состав ПЛК, на третьем – ОС, с которой оператор осуществляет управление технологическим процессом. Однако при реализации конкретных систем возникает множество нюансов и сложных практических вопросов, связанных со стандартизацией, безопасностью, коммерческой эффективностью, совместимостью технологического оборудования и т. д., что не может не отразиться на архитектуре разрабатываемой АСУ.

3.1. Требования к архитектуре

Архитектура АСУ – это ее абстрактное представление, которое включает в себя идеализированные составляющие системы, а также средства взаимодействия между этими компонентами. Элементы архитектуры взаимосвязаны и образуют единую систему, которая обеспечивает решение необходимой задачи автоматизации технологического процесса на уровне архитектуры. Однако при проектировании системы ее архитектура формируется исходя из принятых технических решений за счет проектно-компонуемого оборудования, различных интерфейсов передачи информации, датчиков и преобразователей с различными интерфейсами. Поэтому корректно спроектированная система допускает множество различных реализаций путем выбора компонентов архитектуры и методов их взаимодействия.

Элементами архитектуры являются датчики, устройства ввода-вывода, ПЛК, ОС, интерфейсы, промышленные сети, исполнительные устройства, драйверы и т. д.

При разработке архитектуры АСУ должны быть обеспечены следующие свойства:

- слабая связанность компонентов архитектуры между собой;
- тестируемость;
- диагностируемость;
- ремонтопригодность;
- надежность;
- безопасность;
- простота в обслуживании и эксплуатации;
- защищенность;
- экономичность;
- модифицируемость;

- функциональная расширяемость;
- наращиваемость;
- открытость.

3.2. Разновидности архитектур

В настоящее время распространены различные структуры АСУ: иерархическая, централизованная, децентрализованная и др. Каждая из них имеет свои достоинства и недостатки. Структура разрабатываемой системы выбирается исходя из задач, которые необходимо реализовать в рамках автоматизации производства, сложности технологических алгоритмов, оборудования, на основе которого она строится.

На рис. 1–3 схематично представлены различные структуры АСУ.

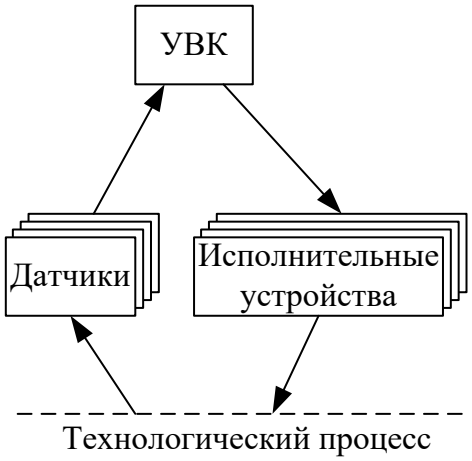


Рис. 1. Централизованная структура АСУ

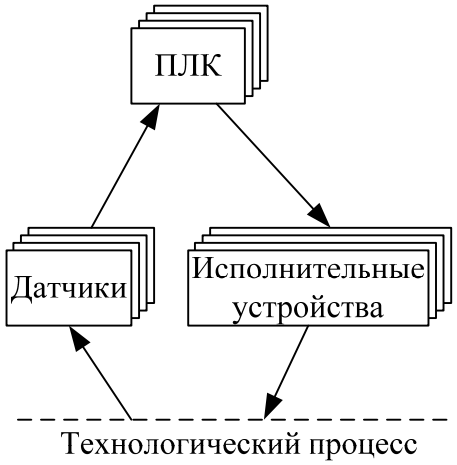


Рис. 2. Децентрализованная структура АСУ

В АСУ с централизованной структурой для сбора и обработки всех данных, а также выработки управляющих воздействий на исполнительные механизмы используется единое устройство – управляющий вычислительный комплекс (УВК). Учитывая факт, что вся информация о ходе технологического процесса аккумулируется в УВК, то, с одной стороны, это повышает качество управления, с другой – ведет к понижению надежности системы в целом, т. к. выход из строя УВК влечет за собой останов всей АСУ. Очевидно, что проблема может быть решена резервированием, но это повышает стоимость АСУ.

АСУ с децентрализованной структурой содержит в себе ряд ПЛК, автономно решающих различные задачи. Управление работой ПЛК осуществляется путем задания начальных настроек или программ.

Достоинством системы с такой структурой является более высокая надежность по сравнению с АСУ с централизованной структурой за счет

способности АСУ функционировать при выходе из строя одного из контроллеров. Техническое решение разведения задач по отдельным автономным ПЛК, позволяющее повысить надежность АСУ в целом, ведет к снижению качества управления, т. к. в данной структуре не имеется устройства, координирующего работу всех ПЛК.

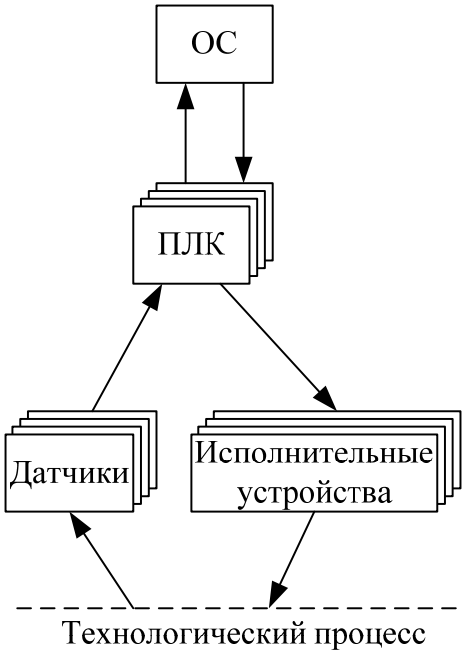


Рис. 3. Иерархическая структура АСУ

Недостаток АСУ с децентрализованной структурой в АСУ с иерархической структурой решается путем ввода координирующего работу контроллеров устройства – ОС, что способствует как повышению качества управления, так и сохранению надежности АСУ благодаря решению задач в различных ПЛК.

3.3. Информационные потоки

В настоящее время существует огромное количество разнообразных датчиков и сенсоров (температуры, давления, уровня, скорости, ускорения, частоты, влажности) различные по исполнению и принципу работы. Как правило, датчики преобразуют физическую величину в унифицированные электрические сигналы. Далее такой сигнал подается на модуль ввода ПЛК, в большинстве случаев имеющих встроенный аналогово-цифровой преобразователь (АЦП), а также микропроцессор, позволяющий линеаризовать характеристики датчика, компенсировать погрешности, установить фильтрацию измеренных значений. Модуль ввода непосредственно подключается к ПЛК. Ранее уже отмечалось, что ПЛК

проектно-компонуемые, это означает, что для каждой из разрабатываемых АСУ подбирается процессор, интерфейсы, линейка модулей ввода-вывода исходя из количества параметров системы, сложности технологических алгоритмов и задач.

Существуют различные модули ввода-вывода, отличающихся по количеству обрабатываемых сигналов, исполнению (аналоговые, дискретные, числоимпульсные и др.). В свою очередь, модули аналогового ввода-вывода подразделяются на токовые (4…20 мА, 0…20 мА, 0…5 мА), по напряжению (0…10 В, 0…5 В, 0…100 мВ и т. д.), модули термосопротивления (ТСМ, ТСП и т. д.), термопары (ТХК, ТХА и т. д.)

Помимо модулей аналогового ввода широко используются дискретные модули ввода, которые не включают в свой состав АЦП и обрабатывают сигналы, имеющие два уровня (концевые выключатели технологического оборудования, датчики-сигнализаторы уровня, давления, температуры, датчики открытия дверей и т. д.).

Модули с числоимпульсным входом имеют дискретный вход и позволяют считать количество импульсов или их частоту. Широко применяются, например, при измерении скорости вращения вала электродвигателя.

Для реализации технологических алгоритмов разрабатывается ПО, осуществляющее работу ПЛК согласно требуемым алгоритмам, с помощью программных средств, позволяющих программировать ПЛК. Для связи ПЛК с ОС используются промышленные сети Ethernet, Controlnet, RS-485, Profibus и др.

На ОС устанавливается ПО – SCADA-система, в которой разрабатывается графический интерфейс АСУ, описывающий технологический процесс (мнемосхемы, графики, тренды, технологические схемы и т. д.) и средства дистанционного и автоматического управления с диспетчерской станции оператором (кнопки, панели регулирования, регуляторы и др.).

Связь между SCADA и ПЛК может осуществляться различными способами. Однако наиболее широко распространенный способ – использование OPC-сервер.

Материал, касающийся программирования ПЛК, SCADA, OPC-серверов и их взаимодействия более детально будет рассмотрен в последующих разделах.

Управляющие сигналы, поступающие с ОС (со SCADA АСУ) в ПЛК, а также управляющие сигналы, формирующиеся в ПЛК в автоматическом режиме (согласно принятым критериям и алгоритмам технологического процесса), поступают на модули вывода.

Модули вывода могут выдавать различные сигналы: дискретные, частотные, аналоговые. Как правило, модуль вывода содержит встроенный

цифро-аналоговый преобразователь (ЦАП), если в нем есть необходимость. Преобразованные сигналы поступают в схемы управления технологическим оборудованием: клапаны, насосы, электродвигатели, задвижки и др.

В настоящее время наметилась тенденция стирания грани между ПЛК и компьютером. В свое время внедрение ПЛК объяснялось дороговизной компьютеров, его размерами и другими причинами. В настоящее время на производстве используется все больше и больше промышленных компьютеров, которые имеют специальное конструктивное исполнение, предназначены для круглосуточной работы и удовлетворяют всем требованиям, предписанным ПЛК.

Контрольные вопросы и задания

1. Назовите основные свойства, которые должны обеспечиваться при разработке архитектуры АСУ.
2. Назовите разновидности структур АСУ и их особенности.
3. Назовите основные достоинства и недостатки АСУ с централизованной структурой.
4. Назовите основные достоинства и недостатки АСУ с децентрализованной структурой.
5. Приведите пример организации передачи данных между различными уровнями АСУ с иерархической структурой.

4. ФУНКЦИИ АСУ

Функция АСУ – это некий перечень действий системы, направленный на выполнение определенной задачи.

В литературе можно встретить различные классификации функций. По одной из них выделяют:

- информационно-вычислительные функции;
- управляющие функции.

Предоставление оператору информации о ходе протекания технологического процесса является результатом исполнения информационно-вычислительных функций.

Задача управляющих функций – формирование управляющих воздействий на объект автоматизации.

В состав информационно-вычислительных функций входят:

1) сбор, обработка и хранение данных о ходе технологического процесса; первичная обработка данных включает в себя: фильтрацию значений полученных сигналов в ПЛК, проверку информации на достоверность, аналитическую градуировку полученных данных исходя из технических характеристик датчиков и других устройств;

2) сигнализация состояний параметров оборудования и технологического процесса, например состояние насосов, задвижек, клапанов и т. д.;

3) косвенное измерение параметров процесса и состояния оборудования, вычисление значений параметров по значениям иных параметров, например вычисление объема жидкости через интегрирование ее расхода;

4) расчет эксплуатационных и технико-экономических показателей технологического процесса; в качестве примеров могут быть рассмотрены время наработки на отказ оборудования, расчет стоимости продукта и т. д.;

5) регистрация требуемых параметров, состояния технологического оборудования и результатов, произведенных в системе или ПЛК, расчетов в соответствии с техническим заданием на разработку АСУ;

6) регистрация отклонений параметров процесса и состояния оборудования от установленных значений, которые, как правило, вносятся в журналы SCADA;

7) анализ и выявление первопричин возникших блокировок и технологических защит оборудования, вычисление производится в ПЛК, т. к. время обновления SCADA значительно больше рабочего цикла ПЛК;

8) прогнозирование значений параметров технологического процесса и состояния оборудования с учетом динамики изменения их значений;

9) диагностика состояния технических средств АСУ, таких как короткое замыкание, обрыв линии и другие виды неисправностей;

10) отображение информации о ходе технологического процесса и предоставление рекомендаций по управлению технологическим процессом, информация может быть представлена в различном виде: цифровые значения, графики, мнемосхемы, таблицы и т. д.; в АСУ могут быть внедрены элементы экспертной системы, содержащей выработку различных рекомендаций исходя из состояния технологического оборудования, например рекомендация включить или отключить насос, закрыть или открыть клапан и т. д.;

11) выполнение процедур обмена информации с другими АСУ по различным интерфейсам и протоколам (см. раздел 3.3).

В состав управляющих функций входят различные типы управления:

1) одноконтурное (управление отдельными параметрами технологического процесса), выработка управляющего воздействия регулятором по рассогласованию заданного значения и значения сигнала с датчика;

2) логическое (выполнение блокировок и защит), выработка управляющихся сигналов согласно событиям. Например, если уровень в резервуаре больше заданной отметки, то выполняется команда на закрытие клапана или отключение насоса;

3) каскадное, использование вложенных контуров управления, где для внутреннего контура задающее воздействие формируется регулятором внешнего контура;

4) многосвязное, управление системами класса MIMO (много входов – много выходов, определенным образом влияющих друг на друга);

5) программное, формирование управляющего воздействия в соответствии с принятой программой управления;

6) оптимальное управление технологическими параметрами в установившемся режиме;

7) оптимальное управление технологическими параметрами в условиях переходных процессов;

8) адаптивное управление технологическими процессами путем изменения структуры регулятора и/или его настроечных параметров регулятора.

Контрольные вопросы и задания

1. Дайте определение понятию «функция АСУ».
2. Приведите пример классификации функций.
3. Какая основная задача информационно-вычислительных функций АСУ?
4. Перечислите основные информационно-вычислительные функции АСУ.
5. Какая основная задача управляющих функций АСУ?
6. Перечислите основные управляющие функции АСУ.

5. ОСНОВНЫЕ ФУНКЦИИ ПЛК И ОС

Перечисленные в разделе 4 функции АСУ распределяются между двумя основными вычислительными устройствами: ПЛК и операторской (диспетчерской) станцией.

Функции ПЛК:

1) сбор данных и их первичная обработка;
2) управление и контроль значений технологических параметров;
3) программное управление;
4) сигнализация состояния технологического процесса и задействованного оборудования;
5) выполнение блокировок и защиты оборудования;
6) передача данных из ПЛК к операторской станции;
7) считывание, обработка и исполнение управляющих команд оператора.

Функции станции оператора:

1) отображение различными способами (графики, таблицы, индикаторы, цифровые значения и др.) информации о ходе технологического процесса;
2) архивирование требуемых данных;
3) формирование отчетов, рапортов и др.;
4) конфигурирование и настройка параметров ПЛК, OPC;
5) ручное дистанционное управление.

Контрольные вопросы и задания

1. Перечислите основные функции ПЛК.
2. Перечислите основные функции станции оператора.
3. Может ли станция оператора выполнять функции ПЛК?

6. СОСТАВ АСУ

АСУ включает в свой состав техническое, математическое, информационное и организационное обеспечение.

К техническому обеспечению АСУ относятся такие программно-аппаратные средства, как ПЛК, станции оператора, инженерные станции, устройства сбора, маршрутизации и передачи данных, исполнительные механизмы и т. д.

Математическое обеспечение АСУ делят на алгоритмическое обеспечение – комплекс согласованных между собой алгоритмов, по которым функционирует АСУ, и программное – комплекс программ, разработанных на основе алгоритмов и способов обмена данными между ПЛК и ОС.

Организационное обеспечение представляет собой документацию, содержащую техническое описание АСУ, инструкции по ее эксплуатации, инструкции по обслуживанию и ремонту и т. д.

Информационное обеспечение включает в свой состав документацию о входных и выходных сигналах с их характеристиками.

На рис. 4 приведена схема информационных потоков составных частей АСУ.

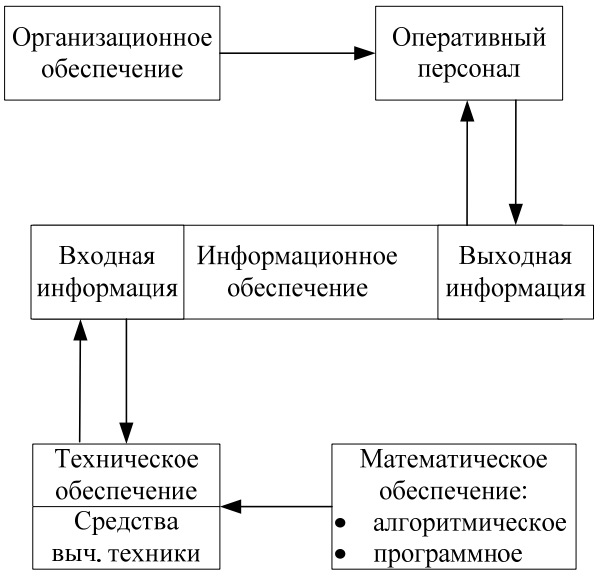


Рис. 4. Взаимосвязь составных частей АСУ

Контрольные вопросы и задания

1. Из каких частей состоит АСУ?
2. Назовите составляющие технического обеспечения АСУ.
3. Назовите составляющие математического обеспечения АСУ.
4. Назовите составляющие информационного обеспечения АСУ.
5. Назовите составляющие организационного обеспечения АСУ.

7. СТАНДАРТ МЭК 61131

В последние десятилетия активно развиваются промышленные системы автоматизации. Одно из центральных мест в АСУ занимают ПЛК. С повышением количества выпускаемых ПЛК возникает вопрос унификации программных средств, позволяющих запрограммировать их работу. Для этих целей был разработан стандарт IEC 1131, впоследствии переименованный в IEC 61131 (МЭК 61131).

7.1. История развития стандарта

Процесс разработки стандарта для ПЛК был инициализирован в 1979 г., создана специальная рабочая группа, в состав которой входили технические эксперты. Первая версия стандарта была издана в 1982 г. Однако это версия оказалась сложной для работы с таким документом, поэтому было принято решение разбить этот стандарт на пять направлений.

Образовано пять подразделений для разработки стандарта в следующих направлениях:

1) общая информация;
2) требования к оборудованию и тестам;
3) языки программирования;
4) руководства пользователя;
5) разработка сообщений.

Впоследствии пятое направление было разделено на два:

- разработка спецификации сообщений;
- программируемые контроллеры – связь.

В рамках рассматриваемого учебного пособия представляет интерес раздел «Языки программирования». Первоначально этот стандарт имел название IEC 1131-3, был опубликован в 1993 г., однако в 1997 г. МЭК (IEC) изменил систему обозначений, и в названии стандарта добавилась цифра «6».

В более современной редакции стандарт стал состоять из восьми частей:

1) общая информация;
2) требования к оборудованию и тестам;
3) языки программирования;
4) руководства пользователя;
5) спецификация сообщений;
6) полевые сети;
7) программирование с нечеткой логикой;
8) руководящие принципы применения и реализации языков.

7.2. Характеристики стандарта МЭК 61131-3

Системы программирования, основанные на МЭК 61131-3, характеризуются следующими показателями:

- надежностью разрабатываемого программного обеспечения. Обеспечение надежности достигается за счет использования специализированного программного обеспечения, предназначенного для разработки кода ПЛК, содержащего различные средства разработки кода и его отладки, эмуляторы, фирменные сервисы и др.;
- допустимостью модификации и развития программного обеспечения;
- возможностью переносить программное обеспечение с одного ПЛК на другой;
- доступностью различных языков программирования.

7.3. Языки стандарта

Стандарт МЭК 61131-3 определяет языки программирования ПЛК. ПО строится так, что отдельные части кода могут быть реализованы на различных языках стандарта, а затем объединены в единое исполняемое ПО.

В состав стандарта МЭК 61131-3 были включены структурное программирование, инкапсуляция, абстрактные типы и др.

Внедрение языков МЭК 61131-3 основывалось на анализе множества языков, уже широко используемых на практике. МЭК 61131-3 устанавливает пять языков программирования.

Стандарт МЭК 61131-3 включает в свой состав графические языки: Релейно-контактных схем (Ladder diagram, LD) и Диаграммы функциональных блоков (Function block diagram, FBD). В данных языках представлено соответствие между графическими примитивами, описывающими решение конкретной задачи, и программами, реализующими алгоритмы решения задачи. Отдельный графический элемент решает определенную задачу при задании значений входных параметров.

Элементная база языка LD содержит набор символов для построчного программирования. Программы языка LD представляют собой релейные схемы, включающие в свой состав нормально замкнутые и нормально разомкнутые контакты, обмотки реле.

Элементная база языка FBD выглядит как функциональные блоки, соединенные между собой в электрическую цепь. Такое графическое представление делает этот язык удобным при работе с большим количеством прикладных программ, содержащих передачу данных между ее компонентами. Лежащие в основе этого языка функциональные блоки

являются программными объектами, выполняющими определенные алгоритмы управления, вычислений и т. д. Полученные данные на выходе такого блока можно передавать на входы других блоков по средствам соединительных линий.

В дополнение к представленным графическим языкам МЭК 61131-3 включает в свой состав язык «Схем последовательных функций» (Sequential function chart, SFC). Элементной базой языка SFC являются шаги и переходы, которые могут быть задействованы для исполнения алгоритмов, описанных на любых других языках стандарта.

Согласно описанию стандарта МЭК 61131-3 программная среда, предназначенная для программирования ПЛК, представлена такими текстовыми языками, как «Список инструкций» (Instruction list, IL) и «Структурированный текст» (Structured text, ST). IL является языком низкого уровня, подобным ассемблеру, а язык ST – языком высокого уровня, используемым для структурного программирования. Язык ST позволяет работать с арифметическими и булевыми операциями, а также содержит конструкции программирования: условия, циклы, оператор множественного выбора и т. д.

7.4. Требования к языкам стандарта

Согласно стандарту МЭК 61131-3 в основе его языков лежат следующие принципы:

- проект формируется как составное множество функциональных компонентов – Program Organization Units (POU). В качестве таких компонентов могут выступать программы, функциональные блоки и функции;
- соблюдение строгой типизации данных;
- многозадачность, обеспечивающая исполнение различных компонентов в разное время с различной периодичностью;
- поддержка структур разнородных данных, например объединение разного типа данных об одном объекте;
- возможность использования в одном проекте различных языков программирования, разработка компонентов POU на различных языках программирования;
- обеспечение возможности переноса кода с одного контроллера на другой;
- организация циклической работы ПЛК, включающей себя опрос модулей ввода, исполнение программ и формирование команд на устройства вывода.

7.5. Значимость стандарта

Несомненно, внедрение стандарта МЭК 61131-3 оказало свое воздействие как на пользователей, так и на разработчиков ПЛК.

Стандартный набор языковых средств программирования ПЛК позволяет подготавливать кадры для работы в этой отрасли. Наличие фиксированного списка языков делает возможным подготовку специалиста способного программировать промышленные ПЛК различных производителей, в силу того что базовое ПО построено на едином стандарте МЭК 61131-3. С одной стороны, инженеры и системные интеграторы вынуждены изучать пять языков программирования ПЛК, с другой – это обучение стандартизовано.

Как правило, производители ПЛК разрабатывают как аппаратную часть, так и программную, с контроллером поставляют ПО для его программирования. В некоторых случаях поставляется и ПО для разработки SCADA ОС, и ПО для связи между SCADA и ПЛК, в частности такие компании Siemens, Emerson, Rockwell Automation. Однако существуют и производители контроллеров, которые используют готовое ПО для программирования ПЛК. Примерами таких компаний являются Wago и ОВЕН, использующие в своих контроллерах ПО компании 3S Software.

Несмотря на то, что ПО разрабатывается на базе стандарта МЭК 61131-3, существуют отличия и в элементной базе ПО различных производителей, в синтаксисе и пр. Однако эти отличия не значительны и осваиваются в краткие сроки.

Еще одним достоинством от внедрения стандарта является возможность построения интегрированных систем управления, включающих в свой состав контроллеры различных производителей.

7.6. Тенденции развития стандарта

Вопросы развития стандарта МЭК 61131-3 регулярно обсуждаются специалистами на различных конференциях, семинарах, вебинарах и других мероприятиях во всем мире. Однако подавляющее большинство производителей систем программирования по-прежнему сосредотачивают свои усилия только на развитии сервисных средств, избегая коренных усовершенствований самих языков.

Исходя из сказанного, намечаются тенденции к расширению норм стандарта МЭК 61131-3 со следующими требованиями:

- расширения являются опциональными;
- возможность совмещения ООП с не ООП;
- поддержка существующих приложений с возможностью их плавной трансформации в ООП по мере необходимости;
- применение ООП во всех языках стандарта МЭК 61131-3.

Такое расширение норм стандарта уже ведется компанией 3S-Smart Software Solutions. Основное расширение МЭК 61131-3 касается трансформации функционального блока в класс. Аналогичным образом

структуры выросли в классы в языке C#. Это достигается за счет введения методов. Фактически методом является функция, встроенная в функциональный блок. В реализации функции доступны не только значения ее параметров и локальных переменных, но и данные экземпляра функционального блока. В итоге вызов метода всегда включает имена экземпляра и метода.

Повышение мощности от использования ООП дает возможность создания интерфейсов. Под интерфейсом понимается набор методов, работающих с одинаковыми параметрами, но разными реализациями для разных функциональных блоков. Интерфейс можно передать в качестве параметра, и программный компонент не будет в действительности заботиться о том, какой функциональный блок им применяется.

Контрольные вопросы и задания

1. С какой целью разрабатывался стандарт МЭК 61131?
2. Назовите основные части стандарта МЭК 61131.
3. Перечислите языки стандарта МЭК 61131 с классификацией на текстовые и графические языки.
4. Сформулируйте основные принципы, лежащие в основе языков МЭК 61131.
5. Назовите основные характеристики систем программирования, разработанных согласно МЭК 61131.
6. Какие требования к расширению норм стандарта МЭК 61131 вы знаете?

8. СТРУКТУРНЫЕ МОДЕЛИ

8.1. Модель программного обеспечения

Основные элементы языка программирования высокого уровня и их взаимосвязи приведены на рис. 5.

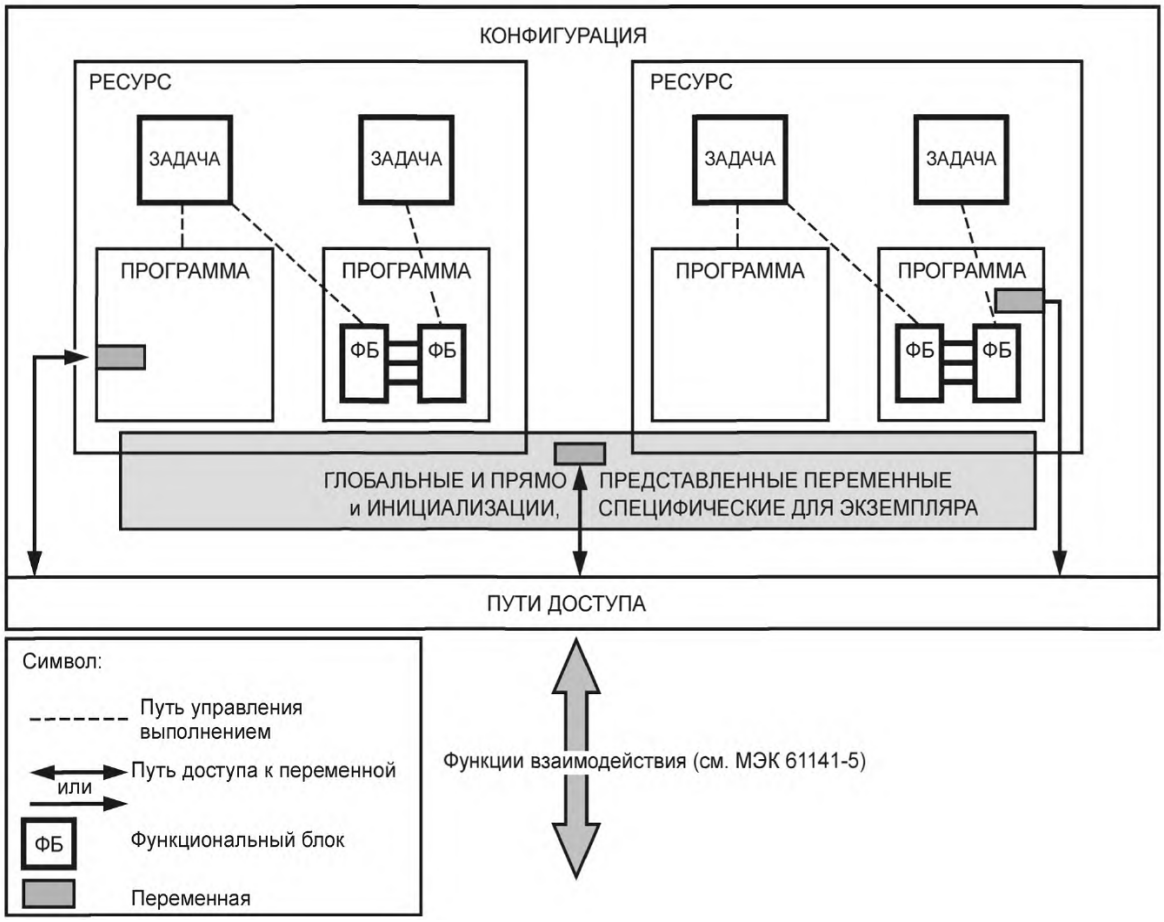


Рис. 5. Модель программного обеспечения

Данные элементы создаются и программируются на языках, определенных в настоящем стандарте, т. е. это программы и типы функциональных блоков, классы, функции и элементы конфигурации, а именно ресурсы, задачи, глобальные переменные, пути доступа и инициализации экземпляров.

Конфигурация является элементом языка, который соответствует системе программируемого контроллера, как определено в МЭК 61131-1. Ресурс соответствует «функции обработки сигналов» и ее «человеко-машинному интерфейсу» и «функциям интерфейса с датчиками и исполнительными механизмами» (при наличии таковых), как определено в МЭК 61131-1.

Конфигурация содержит один ресурс или более, каждый из которых содержит одну программу или более, выполняемых под контролем нуля или более задач.

Программа может содержать нуль или более экземпляров функциональных блоков или других элементов языка, как определено в настоящем стандарте.

Задача способна вызывать (например, на периодической основе) выполнение набора программ и экземпляров функциональных блоков.

Конфигурации и ресурсы могут запускаться и останавливаться через функции «интерфейс оператора», «программирование, тестирование и мониторинг» или «операционная система», определенные в МЭК 61131-1. Запуск конфигурации будет вызывать инициализацию ее глобальных переменных с последующим запуском всех ресурсов конфигурации. Запуск ресурса будет вызывать инициализацию всех переменных в ресурсе с последующей активацией всех задач в ресурсе. Останов ресурса будет вызывать прекращение всех его задач, в то время как останов конфигурации будет вызывать останов всех ее ресурсов.

Механизмы управления задачами определены в 6.8.2, а механизмы запуска и останова конфигураций и ресурсов через функции взаимодействия определены в МЭК 61131-5.

Программы, ресурсы, глобальные переменные, пути доступа (и соответствующие привилегии доступа) и конфигурации могут быть загружены или удалены «функцией взаимодействия», определенной в МЭК 61131-1. Загрузка или удаление конфигурации или ресурса будет эквивалентно загрузке или удалению всех элементов, которые там содержатся.

Пути доступа и их соответствующие привилегии доступа определяются в настоящем стандарте.

Отображение элементов языка на объекты взаимодействия определено в МЭК 61131-5.

8.2. Модель взаимодействия

Способы связи значений переменных с элементами программного обеспечения иллюстрируются на рис. 6.

Как показано на рис. 6, *a*, значения переменных в программе могут связываться прямо, соединением выхода одного программного элемента ко входу другого. Данное соединение явно показывается в графических языках и неявно в тестовых языках.

Значения переменных могут передаваться между программами в одной конфигурации через глобальные переменные, как переменная *x*, показанная на рис. 6, *b*. Такие переменные объявляются в конфигурации глобальными, а в программах – внешними.

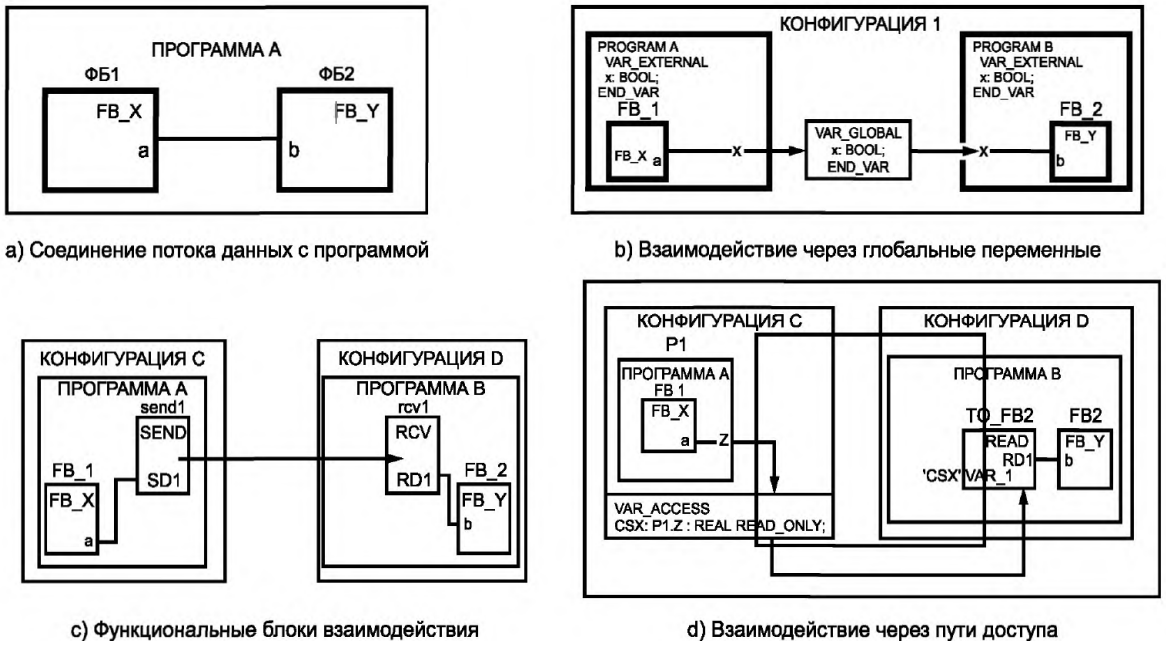


Рис. 6. Модель взаимодействия

Как показано на рис. 6, *c*, значения переменных могут передаваться между различными частями программы, между программами в одной или различных конфигурациях или между программой PC и системой без PC, используя функциональные блоки взаимосвязи, определенные в МЭК 61131-5.

Кроме того, системы с PC и системы без PC могут передавать данные, которые делаются доступными через пути доступа, как показано на рис. 6, *d*, используя механизмы, определенные в МЭК 61131-5.

8.3. Модель программирования

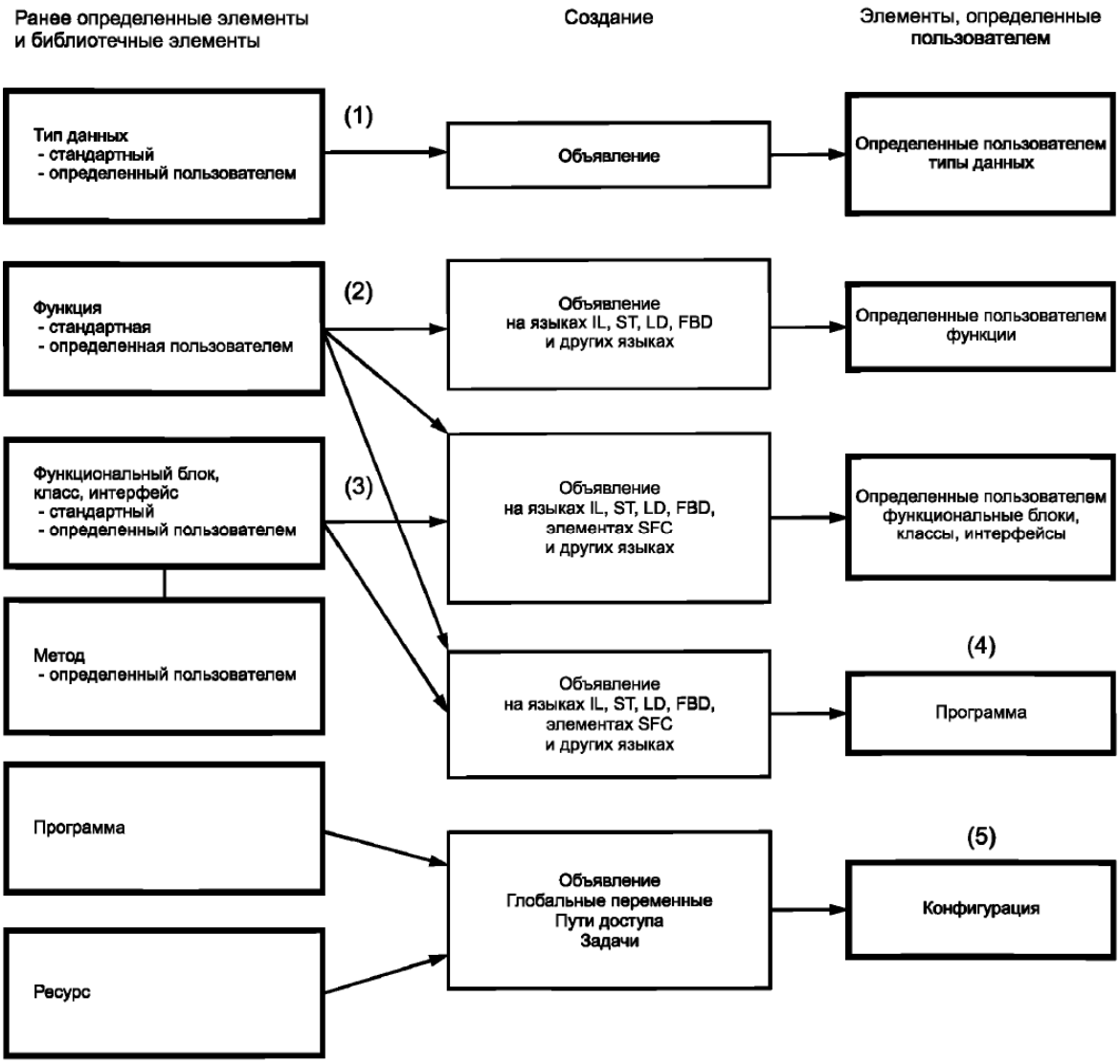
На рис. 7 показана сводка элементов языков программирования PLC. Комбинация этих элементов должна подчиняться следующим правилам:

1. Типы данных объявляются с использованием стандартных типов данных и любых ранее определенных типов данных.

2. Функции объявляются с использованием стандартных или определенных пользователем типов данных, стандартных функций и любых ранее определенных функций (данные объявления должны использовать механизмы, определенные для языков IL, ST, LD или FBD).

3. Типы функциональных блоков объявляются, используя стандартные и определенные пользователем типы данных, функции, стандартные типы функциональных блоков и любые ранее определенные типы функциональных блоков (данные объявления используют механизмы, определенные для языков IL, ST, LD или FBD, и могут включать в себя элементы последовательных функциональных схем (SFC)). Дополнительно можно

определять объектно ориентированные типы функциональных блоков или классы, которые используют методы и интерфейсы.



где LD — язык релейно-контактных схем;
FBD — язык функционально-блоковых диаграмм;
IL — язык списка инструкций;
ST — язык структурированного текста;
Другие — другие языки программирования.

Примечание 1 — Числа от (1) до (5) в скобках относятся к соответствующим параграфам 1)—5) выше.

Примечание 2 — Типы данных используются во всех способах создания. Для четкости, соответствующие связи опущены на данном рисунке.

Рис. 7. Сочетание элементов языка программируемых контроллеров

4. Программа объявляется, используя стандартные или определенные пользователем типы данных, функции, функциональные блоки и классы (данные объявления используют механизмы, определенные

в языках IL, ST, LD или FBD и могут в себя включать элементы последовательных функциональных схем (SFC)).

5. Программы могут собираться в конфигурации, используя элементы: глобальные переменные, ресурсы, задачи и пути доступа. Ссылка на «ранее определенные» типы данных, функции и функциональные блоки означает, что после того как некоторый элемент был объявлен, его определение доступно (например, в «библиотеке» ранее определенных элементов) для использования в дальнейших определениях. Для программирования функций, типов функциональных блоков и методов может использоваться язык программирования, отличный от языков, определенных в настоящем стандарте.

Контрольные вопросы и задания

1. Назовите основные элементы языков программирования.
2. Какое количество ресурсов содержит конфигурация?
3. Назовите способы связи значений переменных с элементами программного обеспечения.
4. Что вызывает на исполнение программу или набор программ?
5. Сформулируйте правила комбинации элементов языков программирования МЭК 61131.

9. ИНСТРУМЕНТЫ ПРОЕКТИРОВАНИЯ ПО ПЛК

В настоящее время программирование ПЛК осуществляется с помощью компьютеров или ноутбуков, имеющих специальное программное обеспечение, позволяющее разрабатывать, загружать, тестировать и отлаживать код для ПЛК.

Как правило, изготовители ПЛК имеют собственные инструментальные программные средства для разработки ПО. Такие инструментальные средства уже адаптированы под работу с конкретной аппаратурой. Очевидно, что изготовители ПЛК не заинтересованы в производстве универсальных средств программирования ПЛК в силу интереса продвижения только своей продукции на рынке.

Разработка стандарта МЭК 61131-3, регламентирующая требования к ПО, характеристики, наличие различных языков программирования и др., предопределила развитие компаний, специализирующихся как на разработке ПО для ПЛК, так и самих ПЛК.

Список таких инструментальных программных систем, построенных на базе стандарта МЭК 61131-3, превышает два десятка, наиболее распространенные приведены в табл. 1.

Таблица 1

Перечень инструментальных программных комплексов

Название инструментальной системы	Фирма-производитель
CoDeSys	Smart Software Solutions, Германия
OpenDK	Infoteam Software, Германия
ACCON-ProSys	Deltalogic, Германия
SUCOsoft S340	Klokner-Moeller, Германия
PUMA	KEBA, Австрия
PDS7	Philips, Нидерланды
NAIS CONTROL	Matsushita AC, Германия
Soft Control	Softing, Германия
SELECONTROL	Selectron Lyss, Швейцария
Concept	Schneider Electric, Франция
ISaGRAF	CJ International, Франция

В последнее время реализация МЭК 61131-3 в качестве дополнительных компонентов поддерживается в ряде SCADA-систем (FIX/Paradym-31, Factory Suite/InControl, WinCC/WinAC, TraceMode и др.).

Программирование на пяти языках МЭК 61131-3 рассмотрим на примере программной среды CoDeSys: инструменты исполнения кода, языки программирования, элементная база и пр.

9.1. Инструменты комплексов программирования ПЛК

Важной задачей инструментальной среды программирования контроллера является упрощение и облегчение труда разработчика ПО за счет автоматизации рутинных процедур. Это позволяет сократить время разработки ПО и избавить разработчика от однообразных действий.

В настоящее время каждая из компаний стремится повысить количество сервисных функций, ведущих к более комфортной и быстрой разработке ПО. Сервисные функции не входят в перечень требований стандарта МЭК 61131-3, однако во многом выбор той или иной программной среды зависит от удобства пользования.

9.1.1. Встроенный редактор

Встроенным редактором принято называть компилятор. Назначением компилятора является считывание всех программ, написанных на языках стандарта МЭК 61131-3, и преобразование текстового или графического кода в машинный код. В результате работы компилятора создается конечный вариант ПО, который затем и исполняется. Инструментальная программная среда подразумевает наличие встроенного редактора. В случае ошибок в программном коде компилятор указывает идентификатор программы, номер ее строки и тип возникшей ошибки, а противном случае указывает об отсутствии ошибок, как показано на рис. 8.

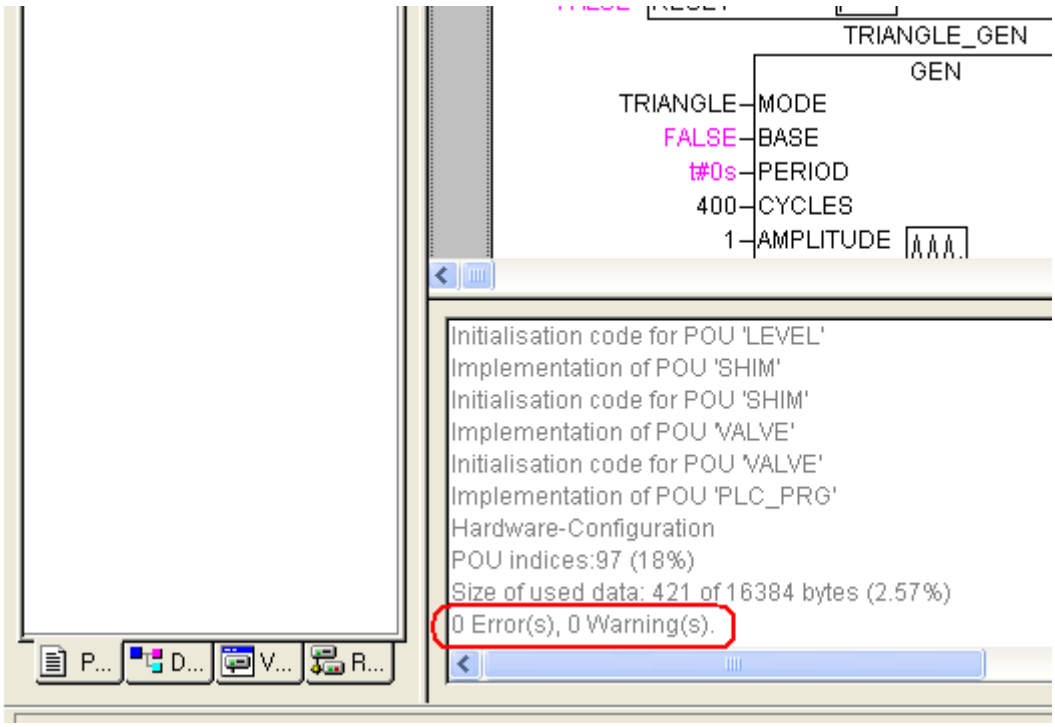


Рис. 8. Результаты работы компилятора

9.1.2. Текстовый редактор

Стандарт МЭК 61131-3, содержащий описание пяти языков программирования, подразумевает наличие как текстовых, так и графических языков программирования. К первым относятся такие языки, как IL и ST. Предполагается, что текстовые языки программирования обладают следующими свойствами:

- быстрый ввод текстовых элементов. Возможность назначения комбинаций горячих клавиш, которые способны обеспечивать вставку команд, функций, функциональных блоков и других стандартных элементов;
- автоматическое дополнение вводимых данных. Например в разделе объявлений переменные можно объявлять в сокращенной форме, а по окончании они преобразуются в требуемый для компилятора вид;
- автоматическое объявление переменных. Если в коде программы используется новый идентификатор, то автоматически появляется окно для ее объявления, за счет которого можно задать класс переменной, тип начальное значение и т. д., при создании выходной переменной с идентификатор valLevel появилось окно объявления переменной Declare Variable (рис. 9);

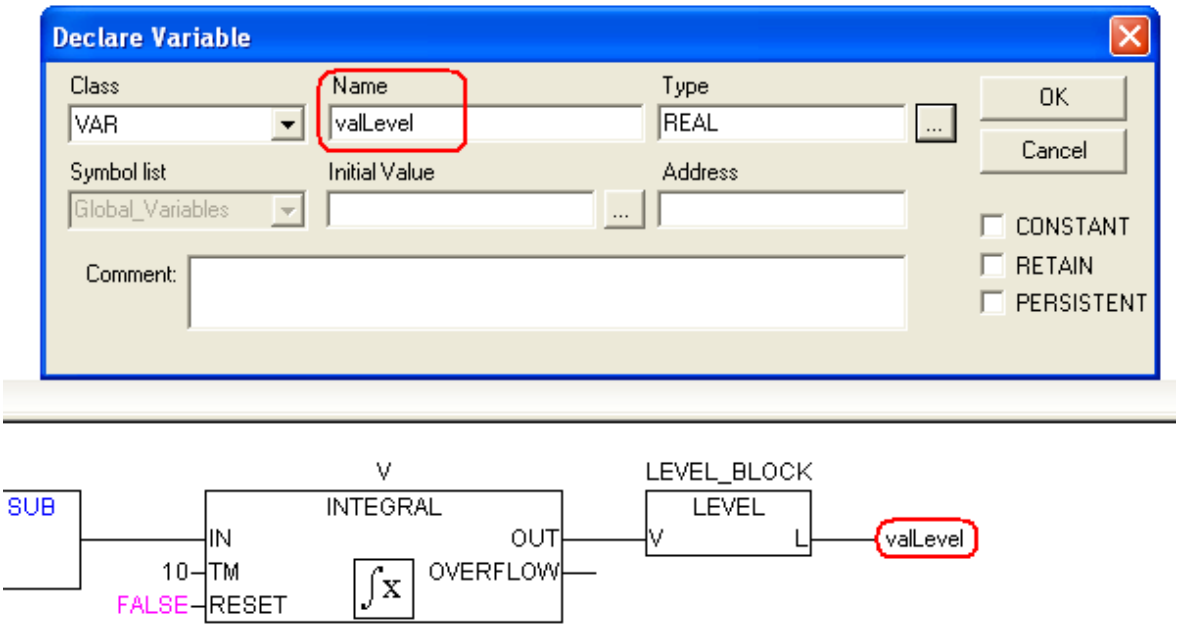


Рис. 9. Автоматическое объявление переменной

- возможность представления раздела объявлений переменных в различных формах (текстовое, табличное), как показано на рис. 10;

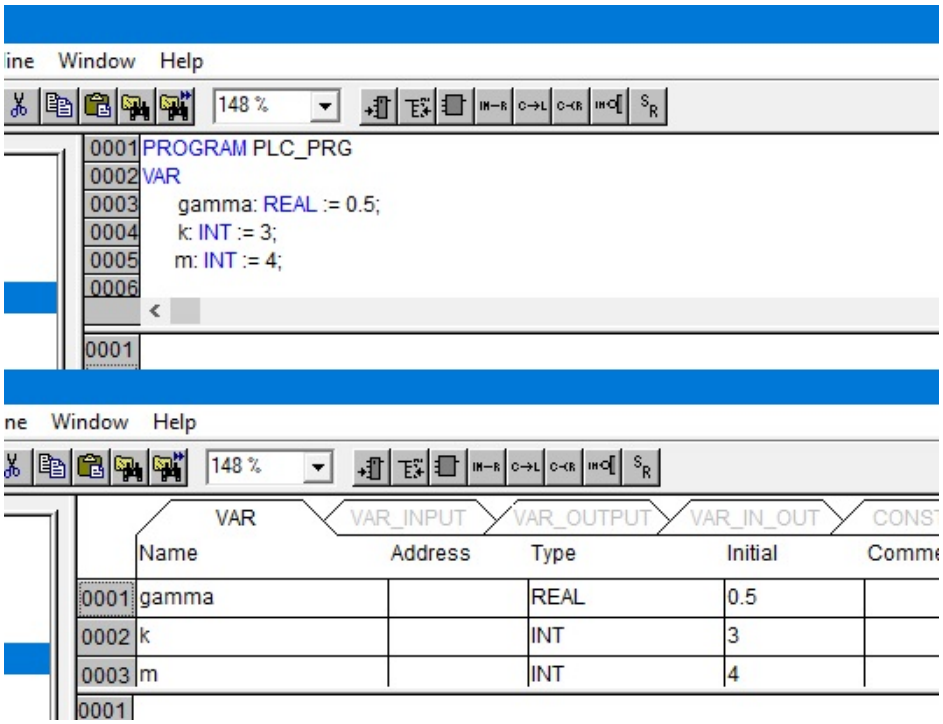


Рис. 10. Текстовое и табличное представление раздела объявления переменных

- автоматическое форматирование вводимого кода и проверка его синтаксиса. Текстовый редактор выделяет код различными цветами резервированные слова, команды, комментарии и т. д.;
- нумерация строк в автоматическом режиме.

9.1.3. Графический редактор

Аналогичным образом для графических языков стандарта МЭК 61131-3 существует графический редактор, который обладает рядом свойств, направленных на увеличение скорости и комфорт работы разработчика:

- автоматическая трассировка соединительных линий блоков программы;
- автоматическое расположение компонентов программы. Это справедливо для программ, разрабатываемых на языках LD, FBD и SFC, а для языка CFC CoDeSys разработчик сам определяет местоположение компонента в программе, тем самым определяя очередность исполнения этих компонентов;
- нумерация алгоритмических цепей в автоматическом режиме;
- возможность копирования, вставки и перемещения выделенных графических элементов программы;
- масштабирование изображения программного кода на экране для его наилучшего представления или детального просмотра;

- в режиме онлайн активные цепи и переменные выделены другой толщиной и подсвечиваются другим цветом. На рис. 11 переменная VALVE_STATE отображается синим цветом, это говорит о том, что в текущий момент она имеет значение True (Истина);
- в режиме онлайн на блоках программы отображаются входные значения, которые можно переписать (в случае если входной параметр блока задан переменной), на рис. 11 показано, что на входе функционального блока ПИД-регулятора отображаются значения входных параметров, двойной клик по переменной MAX_LEVEL привел к появлению окна перезаписи ее значения.

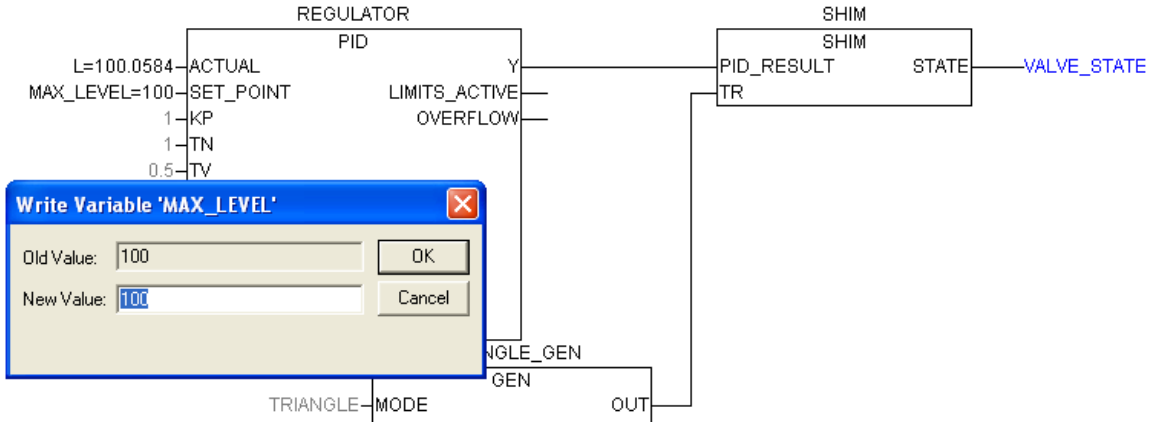


Рис. 11. Запись нового значения переменной MAX_LEVEL

9.2. Средства отладки проекта

Средства отладки проекта инструментальной среды программирования ПЛК обладают следующими функциями:

- механизм установления соединения компьютера и ПЛК (вместо ПЛК может выступать эмулятор). Интерфейс соединения и его характеристики не имеют значения и могут влиять только на следующие физические характеристики их совместной работы, в основном связанные с физической реализацией;
- загрузка кода в память ПЛК;
- автоматический контроль версий загружаемого кода (в случае первичной загрузки ПО CoDeSys информирует об отсутствии ПО в ПЛК (рис. 12), а в противном случае – о том, что загружаемое ПО изменено по сравнению с версией кода, находящейся в ПЛК (рис. 13));
- возможность работы с ПО загруженным в ПЛК в режиме реального времени;
- при останове работы программы прекращается исполнение кода программ ПЛК, однако сохраняется опрос входов и доступна запись в каналы модулей вывода;

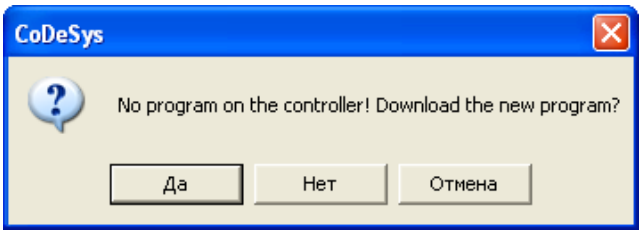


Рис. 12. Загрузка ПО в ПЛК

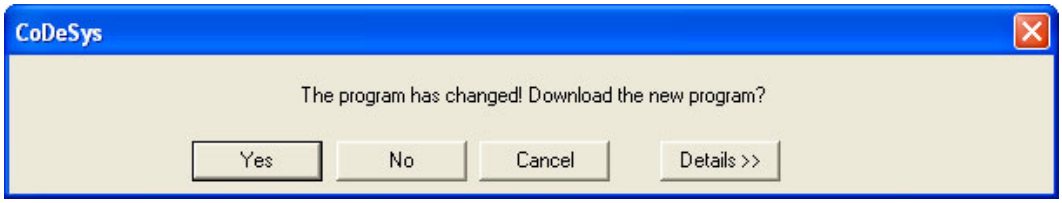


Рис. 13. Перезапись ПО в ПЛК

- сброс данных, хранящихся в ПЛК. Возможно осуществление сброса нескольких видов:
  - «горячий» сброс – программы переводятся в исходное состояние, переменные приобретают начальные значения;
  - «холодный» сброс – программы переводятся в исходное состояние, переменные приобретают начальные значения, переменные энергонезависимой памяти также получают начальные значения;
  - заводской сброс – удаление программ из памяти ПЛК и возврат настроек ПЛК к заводским;

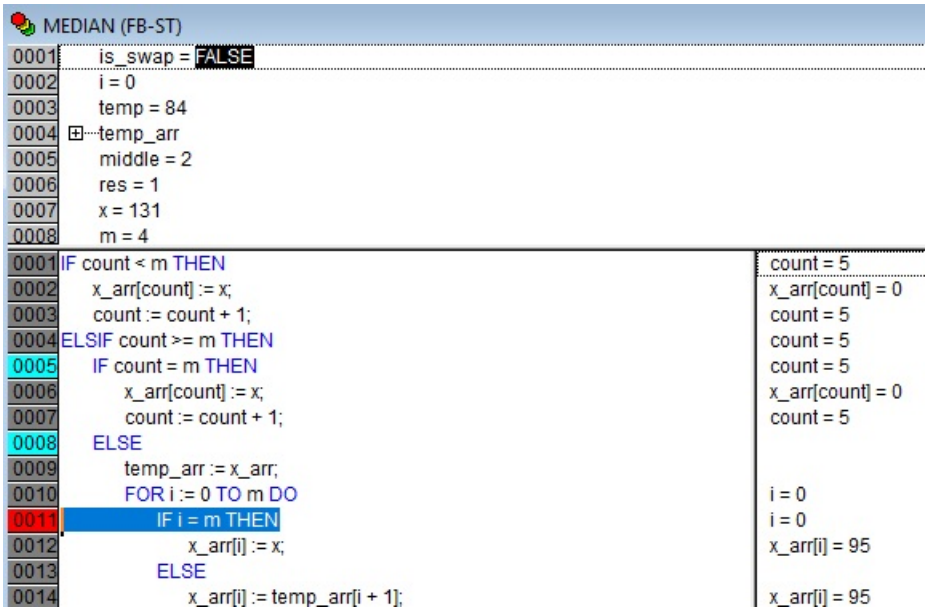


Рис. 14. Пошаговая отладка кода функционального блока

- ввод, изменение и мониторинг мгновенных значений переменных проекта;

- имитация входных и выходных значений параметров системы управления;
- пошаговое выполнение управляющей программы (построчно, по одному рабочему циклу) с возможностью задания точек останова (рис. 14);
- отслеживание перекрестных ссылок переменных управляющих программ.

9.3. Средства управления проектом

Инструментальная среда программных комплексов ПЛК содержит встроенные средства управления проектом – менеджер, который решает следующие задачи:

- представление дерева проекта, содержащего все компоненты разработанного проекта;
- работа с компонентами проекта (создание, переименование, перемещение, копирование, удаление);
- настройка ресурсов проекта;
- управление библиотеками проекта (подключение существующих, разработка и добавление собственных библиотек);
- формирование файлов документальной версии проекта, содержащей:
  - текстовое описание компонентов POU;
  - ресурсы разработанного проекта – конфигурация ПЛК, перечень и описание задач проекта, глобальных переменных и подключенных библиотек;
  - перечень перекрестных ссылок переменных проекта.

9.4. Средства восстановления проекта

В процессе эксплуатации внедренной АСУ так или иначе возникает потребность в корректировке исходного кода. Однако не стоит исключать возможности утраты исходных файлов внедренного проекта. Такая задача решается различными способами. Наиболее распространен подход хранения всех исходных кодов в памяти ПЛК. Как правило, файлы сжимаются, архивируются и хранятся на карте памяти. Наряду с этим способом существует и подход декомпиляции кода, преобразования исполняемого кода в МЭК-программы. Однако такой подход более сложный по реализации, поэтому и встречается крайне редко.

9.5. Средства обеспечения безопасности

Доступ к изменениям и модификации программного кода обычно закрывается паролем, которым владеет администратор АСУ. Ограниченный доступ повышает уровень безопасности АСУ и исключает возможность доступа с правами записи постороннего человека.

В некоторых случаях для ограничения доступа к системе используется аппаратный ключ.

Контрольные вопросы и задания

1. Приведите примеры инструментальных программных комплексов, предназначенных для ПЛК.
2. Дайте определение понятию «Встроенный редактор».
3. Перечислите свойства текстового редактора.
4. Перечислите свойства графического редактора.
5. Какими средствами отладки проекта обладает среда программирования?
6. Какие задачи решает инструментальная среда программных комплексов ПЛК?

10. СТРУКТУРА ПО ПЛК

Инструментальная среда для разработки кода ПЛК имеет четкую структуру. В менеджере проекта при его разработке отображается дерево, или структура, проекта. На рис. 15 представлен пример структуры проекта.

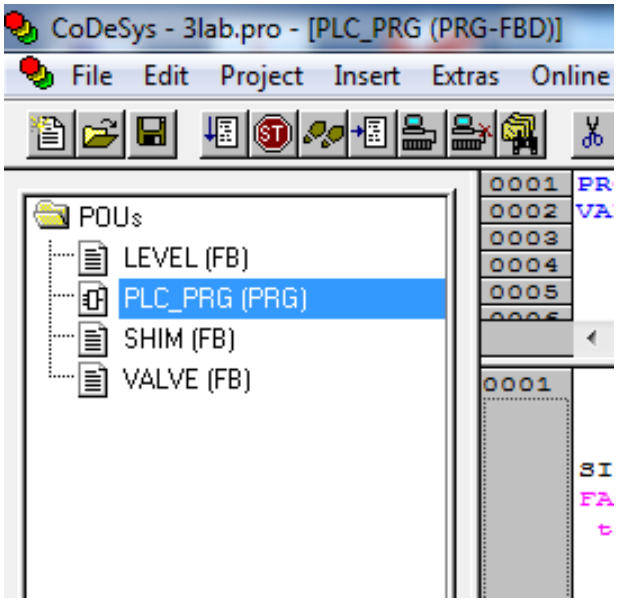


Рис. 15. Дерево проекта

В показанной на рис. 15 структуре представлены программа с именем PLC_PRG и функциональные блоки LEVEL, SHIM, VALVE. В скобках отображается тип элемента: программа, функциональный блок, функция. Управление исполнением программ определяется на уровне задач.

10.1. Задачи

Управление работой программ проекта обеспечивает задача. Каждая задача проекта имеет уникальное имя и обладает определенными характеристиками. Отдельная задача может управлять не только одной, но и целым рядом программ, которые все выполняются в каждом рабочем цикле ПЛК. В составе одного проекта возможно создание нескольких задач как одинаковых, так и различных типов.

Существуют задачи различных типов, например циклические, независимые (со свободным ходом), разовые (рис. 16). Разовые задачи выполняются по какому-либо событию, которое определяется передним фронтом триггерной переменной, соответствующей этому событию. Циклические задачи выполняются через определенные разработчиком проекта интервалы времени. Независимые задачи со свободным ходом исполняются также циклически, однако не имеют фиксированных интервалов времени

между исполнениями задачи и приступают к новому $n+1$ выполнению по завершению n-й итерации.

Рис. 16. Атрибуты задачи

В различных инструментальных средах программирования ПЛК описание задач осуществляется собственными способами (текстовое или графическое представление). В программной среде CoDeSys вкладка «Ресурсы» содержит менеджер задач, в котором создаются задачи и определяются их атрибуты (рис. 17).

Для исполнения программ ПЛК в любом проекте должна существовать как минимум одна задача. В программной среде CoDeSys по умолчанию существует циклическая задача, в состав которой добавлена программа с именем PLC_PRG (по умолчанию).

Исполнительная система управляет работой задач. Для того чтобы управлять исполнением задач, им присваиваются приоритеты. Приоритет задачи определяется назначенным ей числом в диапазоне от 0 до 31 (рис. 16).

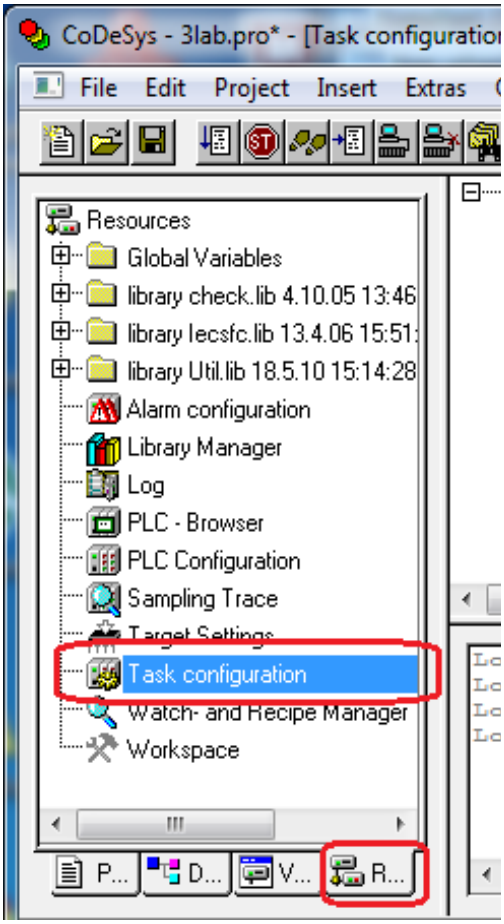


Рис. 17. Менеджер задач

Чем меньше назначаемое число, тем более высоким приоритетом обладает исполняемая задача. Если на исполнение приходят несколько задач, то первой выполняется задача с более высоким приоритетом. В том случае если в очереди на исполнение находятся две задачи с одинаковым приоритетом, то в первую очередь исполняется та, которая имеет большее время ожидания. Если две циклические задачи с различными приоритетами синхронизировано приходят на исполнение, то задача с меньшим приоритетом не будет исполняться вовсе.

В исполнительной системе ПО ПЛК может быть реализована как невытесняющая многозадачность, так и вытесняющая. В случае невытесняющей многозадачности активная задача выполняется до тех пор, пока сама не передаст управление планировщику, в вытесняющей – решение о переключении на другую задачу принимается самим планировщиком.

10.2. Ресурсы

Ресурс по отношению к задаче является более глобальным элементом. Он включает в себя процессор и собственную систему исполнения. Производители программного обеспечения для ПЛК используют термин «проект». В состав проекта входит все ПО, обеспечивающее работу определенного приложения, а его ресурсы содержат аппаратно-зависимые элементы проекта (рис. 18).

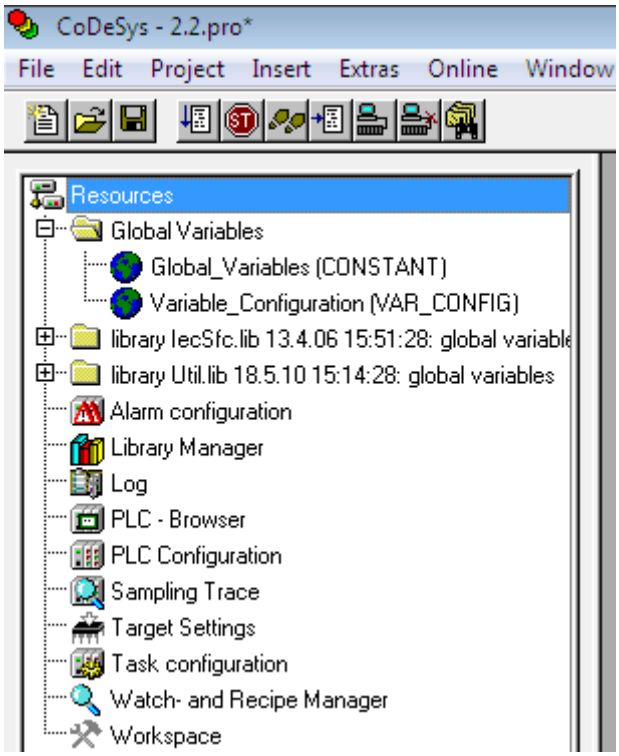


Рис. 18. Ресурсы проекта

В ресурсы входят следующие элементы:

- раздел объявления прямо адресуемых и глобальных переменных;
- менеджер библиотек;
- конфигуратор ПЛК;
- целевые параметры настройки системы исполнения: тип процессора, распределение памяти, параметры сети и др.;
- менеджер задач.

Также различные производители ПО ПЛК в состав ресурсов добавляют различные фирменные инструменты, упрощающие работу с инструментальной средой, такие как конфигуратор сети, терминал обработки команд для ядра ПЛК, бортовые журналы и др.

10.3. Конфигурация

Стандарт МЭК 61131 содержит описание понятия «конфигурация». Конфигурация – это набор ресурсов, работающих и взаимодействующих по определенным установленным правилам. Каждый из ресурсов, входящих в конфигурацию, обладает собственным процессором, памятью и системой исполнения и имеют доступ к некой аппаратной базе, работая и взаимодействуя через глобальные переменные проекта (или мультипроекта).

В программной среде CoDeSys понятие «конфигурация» не реализовано. Однако в Step7 есть возможность создавать мультипроекты, объединяя работу нескольких ПЛК с возможностью передачи данных между ними.

Понятие «конфигурация» наиболее часто используется в распределенных АСУ для решения более сложных специфических задач.

Контрольные вопросы и задания

1. Назовите типы задач.
2. Каким образом в задаче определяется очередность исполнения программ?
3. В чем разница между невытесняющей и вытесняющей многозадачностью?
4. Перечислите основные элементы ресурсов.
5. Дайте определение понятию «Конфигурация».

11. КОМПОНЕНТЫ ОРГАНИЗАЦИИ ПРОГРАММ

11.1. Определение компонента

Базовые элементы программной среды будем называть компонентами. Каждый компонент обладает собственным интерфейсом, наименованием и описан на одном из языков стандарта МЭК 61131-3.

Понятие компонента было затронуто в разделе 9 и определено, что к базовым компонентам программ стандарта МЭК относятся функциональные блоки, функции, программы.

Эти компоненты являются базовыми, и из них строится весь код проекта. Каждый компонент обладает собственным уникальным идентификатором (именем), определенным интерфейсом и описывается на одном из языков стандарта МЭК 61131-3.

Вызов компонентов может быть организован различными способами. Такой компонент, как программа, может вызываться на исполнение определенной задачей, а также другим компонентом, исполняемым программой. Такие компоненты как функции и функциональные блоки вызываются на исполнение только программами. Отличительной особенностью ПО ПЛК является отсутствие возможности организации рекурсии, т. к. она запрещена стандартом МЭК.

В ПО ПЛК широко применяется свойство инкапсуляции, т. е. компоненты создаются по принципу «черного ящика»: его детали скрыты от пользователя, а для работы достаточно обладать знаниями о входах и выходах и решаемой задаче этим компонентом. Знания внутреннего устройства компонента не являются важными. В графических языках компонент представляет собой прямоугольник с входами слева, выходами справа и именем сверху (рис. 19). Локальные же данные недоступны извне.

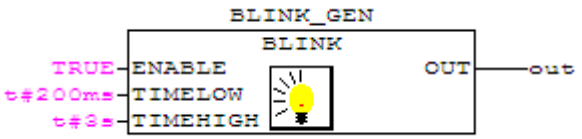


Рис. 19. Вид компонента на языке FBD

Использование свойства инкапсуляции компонентов в проекте решает задачу структурной декомпозиции. Проект строится от большего к меньшему компоненту. В ресурсах проекта разработчик определяет настройки целевой платформы, задает тип процессора, конфигурирует ПЛК, создает задачи, подключает требуемые библиотеки, определяет набор глобальных переменных и т. д. Далее в созданных задачах разработ-

чик осуществляет вызов программ, которые, в свою очередь, могут вызывать другие программы, функции, функциональные блоки. Таким образом, на каждом уровне создания проекта решается определенный круг задач без детализации, а для того чтобы обратиться к деталям, необходимо рассмотреть компонент, который стоит рангом ниже.

Структурная декомпозиция решает задачу локализации имен переменных, что позволяет использовать повторяющиеся имена, т. к. область их видимости – только те компоненты, внутри которых они объявлены. Это позволяет существенно упростить структуру проекта, сократить раздел объявлений перечень глобальных переменных.

11.2. Формальные и актуальные параметры

Входные и выходные переменные, а также заданное имя компонента определяют его интерфейс. Входные параметры компонента являются формальными, при вызове компонента на исполнение им передаются значения внешних актуальных параметров, подключенных к ним, тем самым приобретая актуальные значения. Далее происходят исполнение кода компонента с актуальными значениями формальных параметров и формирование значений выходов компонента, записываемых в актуальные выходные параметры.

В качестве примера рассмотрим таймера-пульса (рис. 20). При вызове экземпляра таймера Timer 1 актуальные значения параметров In x и Time p передаются формальным параметрам IN и PT функционального блока TP. Далее выполняется код экземпляра функционального блока, формируются значения выходов и записываются в переменную out.

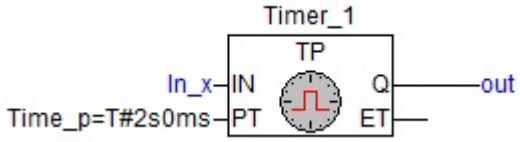


Рис. 20. Актуальные и формальные параметры

11.3. Параметры и переменные компонента

Раздел объявлений переменных компонента содержит несколько типов переменных: VAR_INPUT, VAR_OUTPUT, VAR, VAR_IN_OUT (рис. 21).

Актуальные значения поступают в формальные входные параметры VAR_INPUT путем копирования. Типы параметров должны совпадать. На каждом такте работы ПЛК в компонент будет поступать актуальное значение внешнее по отношению к этому компоненту.

Значения формальных выходных параметров VAR_OUTPUT формируются за счет работы компонента, реализованном в нем алгоритме. Аналогичным образом путем копирования данные записываются в актуальные параметры.

Переменные из раздела VAR являются локальными и используются только внутри компонента.

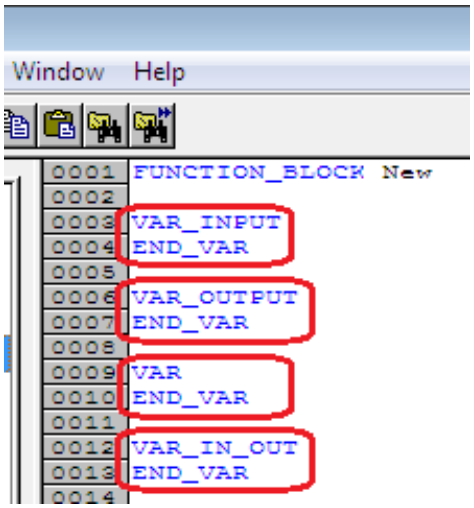


Рис. 21. Раздел объявлений переменных компонента

Переменные из раздела VAR_IN_OUT являются одновременно и входами и выходами. Передача значений переменной экземпляру компонента осуществляется по ссылке. Это означает, что переменная, являющаяся внешней по отношению к компоненту, работает внутри него на правах внутренней переменной. При организации блока с переменной, объявленной в разделе VAR_IN_OUT, в компонент осуществляется только передача адреса расположения значения этой переменной. Существует ряд ограничений для переменной раздела VAR_IN_OUT:

- запрещено использование в функциях;
- запрещено присваивание начальных значений;
- невозможно использование как элемента структуры данных, обращение через точку;
- присваивание переменной возможно только при вызове блоке.

11.4. Функции

Функцией называют программный компонент, который отображает множество входных значений на выход. Функция всегда возвращает только одно значение (которое может состоять из нескольких элементов, если это битовое поле или структура). Имя функции является своего рода выходной переменной, в которую записываются результаты расчетов.

При объявлении функции указывается тип возвращаемого значения, имя функции и входные параметры.

Компонент «функция» реализован без наличия внутренней памяти. Функция может иметь локальные переменные, однако при окончании своей работы они выгружаются из памяти. Отсутствие памяти означает, что функция всегда возвращает одно и то же значение при одинаковых значениях входных параметров.

Тип возвращаемого функцией значения может быть любым из предлагаемого программной средой перечня типов данных, а также созданных пользователем. Тело функции может быть описано на любом из языков МЭК-стандарта за исключением языка SFC, т. к. он не содержит как такого синтаксиса, а лишь определяет структуру и очередность исполнения кода. В теле функции можно осуществлять вызов других библиотечных или пользовательских функций, но нельзя осуществлять вызов функциональных блоков и программ.

На рис. 22 приведен пример функции умножения переменных *a* и *b*.

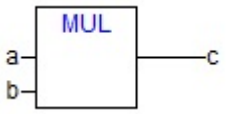


Рис. 22. Функция умножения

11.5. Функциональные блоки

Функциональным блоком является программный компонент, который принимает, обрабатывает и возвращает произвольное количество значений. В коде программ вызывается экземпляр функционального блока. В отличие от функций функциональные блоки не формируют возвращаемые значения. После исполнения экземпляра функционального блока все значения его переменных сохраняются до следующего вызова. В силу того, что функциональные блоки имеют внутреннюю память и хранят в ней значения переменных, вызываемые экземпляры с одинаковыми входными данными, могут иметь различные выходные данные.

На рис. 23 представлен экземпляр функционального блока RS-триггера с именем RS_1.

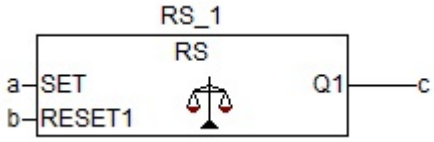


Рис. 23. Экземпляр функционального блока RS-триггера

При обращении к функциональному блоку извне доступны только его входы и выходы, доступ к внутренним переменным функционального блока отсутствует.

11.6. Программы

Программа является глобальным программным компонентом, которая способна формировать произвольные значения во время производимых вычислений. Значения всех переменных сохраняются после окончания работы программы и хранятся до следующего ее вызова.

Исполнение программ определяется на уровне ресурсов проекта через менеджер задач. Однако вызов программы на исполнение можно осуществить в другой программе.

Контрольные вопросы и задания

1. Дайте определение понятию «Формальный параметр».
2. Дайте определение понятию «Актуальный параметр».
3. Назовите типы переменных в разделе их объявления.
4. Какие ограничения имеют переменные раздела VAR_IN_OUT?
5. Сформулируйте ключевое отличие компонентов – функция и функциональный блок.
6. Какие компоненты программной среды доступны на уровне задач проекта?

12. ЯЗЫКИ СТАНДАРТА МЭК 61131-3

При рассмотрении стандарта МЭК 61131 упоминалось о том, что в состав стандарта входят пять языков программирования ПЛК: IL, LD, FBD, ST и SFC. Кроме указанных языков пакет CoDeSys содержит еще один язык – CFC, модернизированный графический язык, очередность исполнения алгоритмических блоков которого определяется расположением на экране. Далее в разделе будут рассмотрены языки программирования ПЛК стандарта МЭК.

12.1. Язык линейных инструкций (Instruction list, IL)

IL (Instruction list) – это низкоуровневый язык, содержащий перечень стандартизированных инструкций, не зависимых от целевой платформы программной среды. Выполнение операций инструкций осуществляется через аккумулятор, а приоритет исполнения инструкций может быть изменен применением переходов на метки.

Инструкция, она же команда, может содержать четыре поля: метка, оператор, операнд и комментарий. Поля, оператор и операнд являются обязательными и должны размещаться в одной строке. Поля, метка и комментарий не являются обязательными. Метка может быть размещена в отдельной строке и заканчивается двоеточием. Метка используется в связке с оператором перехода. Комментарий также может размещаться в отдельной строке (или строках).

Расстановка операторов и операндов, выделение и выравнивание текста кода программы осуществляются автоматически редактором программной среды.

12.1.1. Аккумулятор

Язык IL реализован таким образом, что выполнение операций осуществляется над своего рода хранилищем, в котором и помещается результат выполнения операции.

Например, необходимо сложить два числа. В языке IL необходимо сначала поместить (загрузить) в аккумулятор первое число, а затем выполнить операцию добавления второго числа к первому. В результате выполнения операции в аккумуляторе будет находиться результат от сложения чисел, который можно выгрузить в переменную, соответствующего типа данных.

Аккумулятор способен принимать значения любых типов данных. Однако разработчик кода самостоятельно отслеживает соответствие типов данных при выполнении различных операций. Несоответствие типов данных при компиляции кода покажет ошибку. Аккумулятор не требует

ручной процедуры очистки от данных, при загрузке нового значения в аккумулятор произойдет перезапись значения.

12.1.2. Переход на метку

Логика исполнения кода в языке IL определяется размещением команд сверху вниз. В том случае, когда необходимо изменить приоритет исполнения может быть использован переход на метку, например, для многократного исполнения части кода при организации цикла или для условного перехода на метку требуемого при выполнении какого-то условия. Для реализации таких переходов используются специальные операторы перехода на метку: JMP – безусловный переход и JMPC – условный переход. Разница между этими операторами заключается в том, что при выполнении условного оператора проверяется наличие значения TRUE в аккумуляторе и переход состоится только в том случае, если TRUE имеется в аккумуляторе, в противном случае исполняется следующая строка кода на языке IL. Следует отметить, что переход на метку возможен только в пределах одного компонента.

Рассмотрим пример реализации на языке IL цикла на заданное количество итераций.

```
LD 1 (*загрузка начального значения счетчика в аккумулятор*)
ST Counter (*выгрузка начального значения счетчика в переменную счетчика из аккумулятора*)
loop1: LD Counter (*загрузка значения переменной счетчика Counter в аккумулятор*)
ADD 1 (*добавление к значению в аккумуляторе 1*)
ST Counter (*выгрузка значения аккумулятора в Counter *)
LE 7 (*проверка условия: значение аккумулятора <= 7*)
JMPC loop1 (*переход на метку при условии, что значение аккумулятора TRUE*)
```

В представленном примере реализована организация цикла на 7 итераций. Таким образом, переход на метку изменяет приоритет исполнения кода сверху вниз до тех пор, пока значение переменной Counter не более 7.

12.1.3. Скобки

Помимо перехода на метку порядок исполнения кода может осуществлять специальный модификатор – скобки. Скобки используются в паре: открывающую скобку размещают между оператором и операндом, а закрывающую – в отдельной пустой строке. Команды, размещенные между скобками, выполняются в первую очередь, а результат от их выполнения помещается во временный дополнительный аккумулятор.

Рассмотрим применение модификатора «скобки» на примере фрагмента кода *res*=10*(exp(2)–1) = 18:

```
LD 10
MUL (2
EXP
SUB 1
)
ST res (* res=63,89*)
```

Скобки могут содержать другие вложенные скобки, а результат, формируемый от операций внутри их, также записывается в отдельный временный аккумулятор.

12.1.4. Модификаторы

Один из модификаторов языка IL – скобки – позволяет изменять порядок выполнения кода за счет размещения результата выполненных операций во временный аккумулятор. Наряду со скобками в языке IL имеются модификаторы, позволяющие изменить суть самой функции, вызванной на исполнение. Технически это осуществляется добавлением к оператору модификатора «C» или «N». Такие модификаторы могут употребляться далеко не со всеми операторами, в табл. 2 приведен перечень операторов с указанием допустимого использования модификаторов.

Добавление модификатора «C» (condition) означает, что команда, к которой добавлен модификатор исполнится в том случае, если в аккумуляторе находится логическая единица TRUE.

Использование модификатора «N» (negation) приводит к инверсии значения операнда.

Совместное использование модификаторов «C» и «N» влечет к исполнению команды при наличии логического нуля FALSE в аккумуляторе.

12.1.5. Операторы

В табл. 2 приведены стандартные операторы языка IL, их описание и допустимые модификаторы.

Таблица 2

Перечень операторов языка IL

Оператор	Модификатор	Описание
LD	N	Загрузка значения операнда в аккумулятор
ST	N	Присвоение значения аккумулятора операнду
JMP	CN	Переход на метку

Окончание табл. 2

Оператор	Модификатор	Описание
CAL	CN	Вызов функционального блока
RET	CN	Выход из компонента и возврат в вызывающую программу
S		Установка значения операнда в ИСТИНА, если значение аккумулятора ИСТИНА
R		Установка значения операнда в ЛОЖЬ, если значение аккумулятора ИСТИНА
AND	N, (	Поразрядное И
OR	N, (	Поразрядное ИЛИ
XOR	N, (	Поразрядное ИЛИ
NOT		Поразрядная инверсия аккумулятора
ADD	(	Сложение
SUB	(	Вычитание
MUL	(	Умножение
DIV	(	Деление
MOD	(	Деление по модулю
GT	(	Больше
GE	(	Больше или равно
QE	(	Равно
NE	(	Неравно
LE	(	Меньше или равно
LT	(	Меньше

12.1.6. Вызов функциональных блоков и программ

Вызов функционального блока может осуществляться двумя способами. Далее рассмотрим примеры вызова блока таймер-пульса различными способами.

Пример 1:

```
CAL TP_1(IN:=a, PT:=T#3s)
LD TP_1.Q
ST b
```

В примере 1 показаны вызов функционального блока «таймер-пульс» и присваивание выхода Q переменной b. В свою очередь, вызов функционального блока может быть реализован иначе. Сначала описаны входы функционального блока, а потом осуществлен вызов.

Пример 2:

```
LD a
ST TP_1.IN
```

```
LD T#3S
ST TP_1.PT
CAL TP_1
LD TP_1.Q
ST b
```

12.1.7. Вызов функции

Вызов функций на языке IL не требует использования оператора CAL. Отличительной особенностью вызова функции является необходимость загрузки первого параметра функции в аккумулятор. Рассмотрим пример вызова мультиплексора-селектора, имеющего 9 входных параметров:

```
LD k
MUX a0,a1,a2,a3,a4,a5,a6,a7
ST z
```

12.1.8. Комментирование

Для внедрения в код комментариев используются специальные символы: (* – открытие комментария, *) – закрытие комментария. Таким образом, комментарий помещается между специальными символами, а программная среда выделяет его цветом.

12.2. Структурированный текст (Structured Text, ST)

Язык ST – это текстовый высокоуровневый язык, по структуре и синтаксису напоминающий язык Паскаль и адаптированный под программирование промышленных контроллеров.

12.2.1. Выражения

Язык ST в своей основе содержит различные синтаксические конструкции (условия, циклы, множественный выбор и др.) и выражения, определяющие значения переменных, присвоение которых осуществляется через символы «:=». Выражение может размещаться в нескольких строках, но обязательно должно заканчиваться точкой с запятой. Также и в одной строке может быть несколько выражений, но они должны быть отделены точкой с запятой.

В языке ST при создании выражений математические операции описываются стандартными символами: «+», «–», «*», «/», а также операции сравнения «>»,«<»,«>=»,«<=»,«=»,«<>».

Приведем пример выражения на языке ST:

```
x:=a – 12 * (b + c) + sin(d);
```

12.2.2. Порядок вычислений выражений

При вызове программного компонента, реализованного на языке ST, на исполнение – код исполняется слева направо и сверху вниз. Выражение, содержащее несколько операций, также исполняется в соответствии с установленными приоритетами в программной среде.

Операции в порядке понижения приоритета выполняются следующих образом:

- выражение в скобках;
- вызов функции;
- EXPT;
- смена знака;
- отрицание;
- умножение;
- деление;
- сложение и вычитание;
- сравнение;
- равенство;
- неравенство;
- логические операции.

12.2.3. Пустое выражение

На языке ST допускается использование пустых строк, заканчивающихся точкой с запятой. Компиляция такой программы не выдает ошибки.

Такой подход может быть использован при построении структуры программы и отсутствии на текущей стадии определенных данных разработки проекта. В частности, пустое выражение может быть использовано для описания целого компонента либо определенной конструкции, например IF – THEN – ELSIF – THEN – ELSE.

12.2.4. Оператор IF

При помощи оператора IF может быть реализована конструкция выбора групп действий в зависимости от выполнения различных условий, описываемых логическими выражениями.

Рассмотрим синтаксическую конструкцию оператора IF.

IF *логическое выражение 1*
THEN *перечень действий при выполнении выражения 1*;
[

ELSIF *логическое выражение 2*
THEN
перечень действий при выполнении выражения 2;
…
ELSIF *логическое выражение n*
THEN
перечень действий при выполнении выражения n;
ELSE
перечень действий при невыполнении условий 1..n;
]
END_IF

Код, размещенный в квадратных скобках [ ], не является обязательной частью синтаксической конструкции оператора IF.

Рассмотрим пример реализации сигнализации состояния уровня жидкости в резервуаре:

```
IF L < 10 THEN
   X := ‘НИЖЕ НОРМЫ’;
 ELSIF L > 85 THEN
   X := ‘ВЫШЕ НОРМЫ’;
 ELSE
   X := ‘НОРМА’;
END_IF
```

12.2.5. Оператор множественного выбора CASE

Оператор CASE в назначении схож с оператором IF, но разница заключается в том, что при использовании CASE выбирается группа действий по значению целочисленной переменной (или выражения), а IF использует логические выражения.

Рассмотрим синтаксическую конструкцию оператора CASE:

CASE *целочисленная переменная (или выражение)* OF

значение 1 целочисленной переменной (или выражения):
перечень действий при значении 1;

значение 2 целочисленной переменной (или выражения):
перечень действий при значении 2;

…

значение n целочисленной переменной (или выражения):
перечень действий при значении n;

[
ELSE

перечень действий при ином отличном значении от 1 до n;

]
END_CASE

Код, размещенный в квадратных скобках [ ], не является обязательной частью синтаксической конструкции оператора CASE.

Пример:

```
CASE Counter OF
0..5:          X := ‘НИЖЕ НОРМЫ’;
6..15:         X := ‘НОРМА’;
16..20: X := ‘ВЫШЕ НОРМЫ’;
21:            X := ‘МАКСИМУМ’;
ELSE           X := ‘ПЕРЕПОЛНЕНИЕ’;
END_CASE
```

12.2.6. Циклы While и Repeat

При необходимости обеспечения многократного повторения части кода компонента при наличии выполняющихся определенных условий могут быть использованы циклы WHILE и REPEAT, особенность которых заключается в том, что количество итераций в явном виде неизвестно и выход из цикла осуществляется по значению логического выражения.

Синтаксическая конструкция цикла WHILE:

WHILE *логическое выражение* DO
  тело цикла – перечень действий
END_WHILE

Проверка логического условия осуществляется до входа в цикл WHILE. Если условие не выполняется, то цикл не исполняется вовсе.

Пример цикла WHILE:

```
err := 0,01;
WHILE x > err DO
 x := (a + b)/2;
 counter := counter + 1;
END_WHILE
```

Синтаксическая конструкция цикла REPEAT:

REPEAT
  тело цикла – перечень действий
UNTIL *логическое выражение*
END_REPEAT

Проверка логического условия цикла REPEAT осуществляется после исполнения цикла, поэтому даже если условие цикла не соблюдается, цикл выполнится минимум один раз.

Реализуем тот же самый пример с применением цикла REPEAT:

```
err := 0,01;
REPEAT
  x := (a + b)/2;
  counter := counter + 1;
UNTIL x > err
END_REPEAT
```

При использовании циклов WHILE или REPEAT возможны зацикливание кода и бесконечное исполнение тела цикла. Как правило, это связано с тем, что переменные, задействованные в теле цикла, не связаны с логическим условием.

12.2.7. Цикл FOR

Цикл FOR в отличие от WHILE и REPEAT обеспечивает заданное количество итераций кода, размещенного в теле цикла.

Синтаксическая конструкция цикла FOR:

FOR *целочисленный счетчик := начальное значение* TO
конечное значение [BY *шаг*] DO
тело цикла – перечень действий
END_FOR

Пример:

```
FOR i := 1 TO 10 BY 1
  x := (a + b)/2;
  counter := counter + 1;
END_FOR
```

Таким образом, тело цикла повторяется

(конечное значение – начальное значение)/шаг + 1

количество раз. Счетчик итераций с каждым выполнением тела цикла увеличивается на единицу.

12.2.8. Прерывание итераций операторами EXIT и RETURN

Оператор EXIT требуется для обеспечения выхода из циклов, без проверки каких-либо дополнительных условий. В том случае, когда EXIT применяется во вложенном цикле, то выход осуществляется только из внутреннего цикла.

При решении различных задач оператор EXIT применяется в синтаксической конструкции IF – THEN с целью выхода из цикла по причине достижения каких-либо заданных условий и отсутствия необходимости выполнения последующих итераций тела цикла.

Доработаем пример, рассмотренный в разделе 11.2.7:

```
FOR i := 1 TO 10 BY 1
  x := (a + b)/2;
   IF x < 0,01 THEN
     EXIT;
   END_IF
  counter := counter + 1;
END_FOR
```

В приведенном примере рассмотрен цикл на 10 итераций, но в случае выполнения условия x < 0,01 будет осуществлен досрочный выход из цикла.

Оператор RETURN используется для выхода из исполняемого компонента и передачи управления системе исполнения проекта. RETURN как и EXIT не требуют дополнительных проверок условий.

Рассмотрим пример. Предположим, что в проекте вызов программы Program_1 осуществляется в программе PLC_PRG и по событию Stop необходимо немедленно вернуться к исполнению программы PLC_PRG, это позволит сделать фрагмент кода программы Program_1:

```
IF Stop THEN
    RETURN;
END_IF
```

12.3. Релейные диаграммы LD

Язык релейных диаграмм, или релейно-контактных схем (Ladder Diagram, LD), – это графический язык стандарта МЭК 61131, код которого состоит из построчно организованных электрических цепей, содержащих определенным образом соединенных контактов различного типа и событий.

Считается, что язык LD ориентирован на специалистов с электротехническим образованием, однако иногда выбирается программистами исходя из удобства решения конкретной задачи на данном языке.

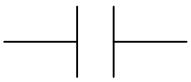
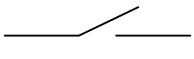
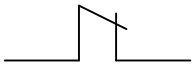
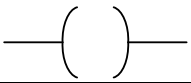
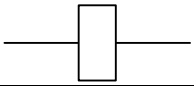
Графически код, разработанный на языке LD, представляет собой набор цепей, которые начинаются с последовательно или параллельно соединенных нормально замкнутых или разомкнутых контактов, а заканчиваются событием. По своей сути, контакты и события – это булевы переменные и в различных цепях они могут быть использованы как события,

так и контакты. Это означает, что в одном случае осуществляется чтение, а в другом – запись.

Элементная база языка приведена в табл. 3.

Таблица 3

Базовые элементы языка LD

LD	ЕСКД	Обозначение
		Нормально разомкнутый (открытый) контакт
		Нормально замкнутый (закрытый) контакт
		Обмотка реле

12.3.1. Реле с фиксацией

В языке LD обмотка реле, или событие, может иметь модификации, аналогичные работе поляризованного реле. Такое реле имеет две обмотки, переключающие его из одного положения в другое. Переключение производится импульсами. Примером использования такого реле является пускатель насоса.

В языке LD такое реле технически реализуется при помощи событий, содержащих модификаторы SET и RESET. SET обозначаются буквой S внутри круглых скобок (S), а RESET обозначаются буквой R. Если в переменную, соответствующую событию (S), записывается значение TRUE, то она сохраняет это значение до тех пор, пока не получит значение TRUE, соответствующее событию (R) для этой же переменной. На рис. 24 представлен их внешний вид. Аналогично работает элемент – RS–триггер, описание работы которого будет представлено в разделе «Стандартные компоненты».

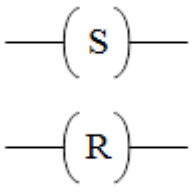


Рис. 24. События Set и Reset

12.3.2. Порядок выполнения и обратные связи

В языке LD код исполняется слева направо сверху вниз. Все цепи однократно исполняются за один рабочий цикл контроллера.

Порядок исполнения кода может быть изменен за счет применения перехода на метку.

Метка указывается над цепью, а правило их именования аналогичны правилам именования переменных. Очевидно, что в одной строке (цепи) может быть только одна метка и один переход на метку. Таким образом, используя переход на метку, можно изменить очередность исполнения цепей кода, реализованного на языке LD. Переход на метку может быть организован как вверх по строкам, так и вниз. Пропущенные цепи сохраняют значения предыдущего цикла. Применение перехода на метку позволяет создавать циклы на языке LD.

12.3.3. Расширение возможностей LD

Как было отмечено ранее, основными элементами языка LD являются контакты и события. Однако можно в коде, реализованном на LD, можно использовать функциональные блоки и функции. Для их внедрения в LD-цепи компоненты должны иметь специальный булевый вход EN, которым компонент подключается к цепи, а в случае необходимости продолжения цепи такой компонент должен иметь и булевый выход ENO, после которого в цепь могут быть добавлены другие контакты (или компоненты) и событие.

На рис. 25 представлен вызов функционального блока Obrabotka на языке LD. Интерфейс функционального блока Obrabotka, являющегося функциональным блоком пользователя, содержит булевый вход EN и выход ENO соответственно для вызова экземпляра этого блока в LD-диаграмме.

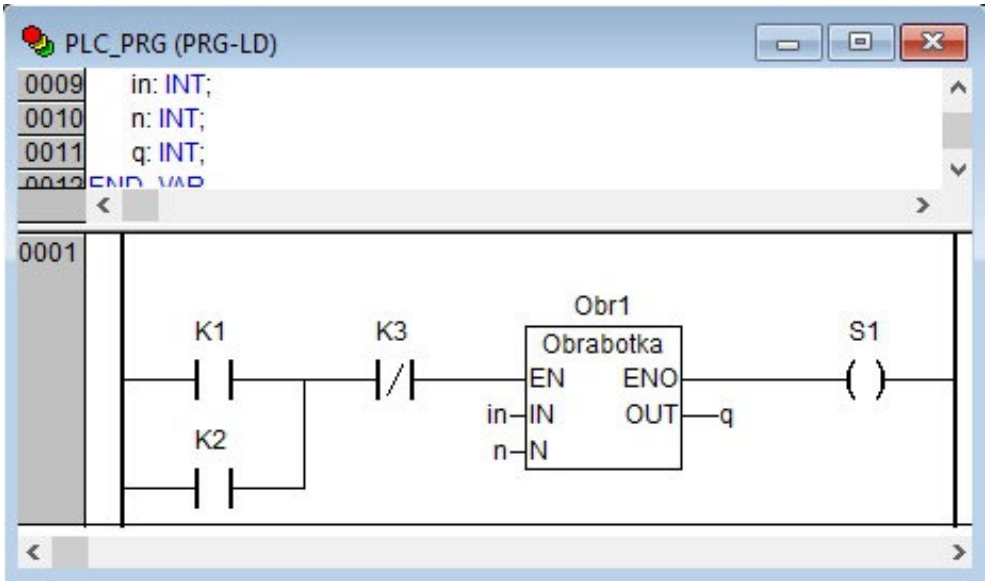


Рис. 25. Вызов функционального блока

12.4. Функциональные диаграммы блоков (FBD)

Язык функциональных диаграмм блоков (Function Block Diagram, FBD) – это графический язык программирования стандарта МЭК 61131, состоящий из построчно организованных цепей и строящийся посредством компонентов (блоков) и соединительных линий.

Графически на языке FBD компонент представляется в виде прямоугольника, у которого слева изображены входы, а справа – выходы. Внутри компонента указываются имена его формальных параметров, а также его имя. Размеры прямоугольника не являются фиксированными и зависят от количества входов и выходов.

12.4.1. Соединительные линии

Передача информации от компонента к компоненту осуществляется слева направо и обеспечивается посредством применения соединительных линий.

Вход компонента может быть соединен с константой, переменной или с выходом другого компонента, размещенного слева, при соблюдении типов данных.

Во многих программных средах соединительные линии строятся автоматически при добавлении компонента.

12.4.2. Порядок выполнения FBD

Программы, разработанные на языке FBD, исполняются слева направо и сверху вниз, построчно. При разработке кода на языке FBD размещение блоков осуществляется программной средой автоматически.

При исполнении кода программы блок получает актуальные входные значения, выполняется код блока и осуществляется запись значений в выходные параметры. Далее полученные данные передаются по соединительным линиям к другим блокам или записываются в переменные. Программа исполняется за один рабочий цикл.

Логика исполнения программы может быть изменена, детальнее это вопрос будет рассмотрен 12.4.4.

12.4.3. Инверсия логических сигналов

Интерфейс программной среды языка FBD позволяет инвертировать входы и выходы блоков типа BOOL, BYTE, WORD и DWORD.

Инверсия является независимым от блока действием, осуществляемым над входным или выходным параметром. Графически инвертированный вход или выход имеет обозначение в виде окружности, расположенной вместе соединения блока и соединительной линии. На рис. 26 продемонстрирована инверсия входа и выхода блока AND.

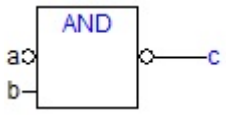


Рис. 26. Инверсия входа и выхода блока

12.4.4. Соединители и обратные связи

В целях сокращения длины цепи на языке FBD могут использоваться соединители, по своей сути – промежуточные переменные. Такой прием позволяет не только сократить длину цепи, сделать ее более читаемой, но и при необходимости изменить логику исполнения, разместив такую переменную-соединитель через определенное количество цепей ниже или выше по коду программы.

В некоторых программных средах использование обратной связи допустимо. Значение по обратной связи принимается в работу на следующем рабочем цикле контроллера. Однако в программной среде CoDeSys на языке FBD построение обратных связей соединительными линиями недопустимо, но в качестве обратной связи можно использовать переменную.

12.4.5. Метки, переходы и возврат

Как уже упоминалось ранее, код, разработанный на языке FBD, исполняется построчно – слева направо и сверху вниз. Как и в других языках программирования МЭК 61131 эта логика может быть изменена посредством перехода на метку.

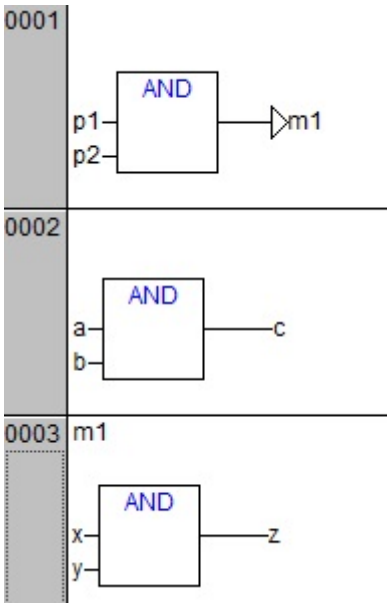


Рис. 27. Переход на метку

В редакторе языка имеется специальный графический элемент – Jump, соответствующий функции перехода на метку. Каждая цепь языка может иметь только одну метку с уникальным идентификатором (именем), подчиняющимся правилам именования переменных. Переход на метку может состояться как вниз, так и вверх по коду.

В качестве примера рассмотрим фрагмент программы, содержащей переход на метку (рис. 27). В том случае, если переменные *p1* и *p2* имеют значения ИСТИНА, то осуществляется переход на цепь с меткой *m1*, т. е. цепь на 0003, пропуская цепь 0002. В противном случае, если хотя бы одна из двух переменных *p1* и *p2* имеет значение ЛОЖЬ, все цепи исполняются последовательно.

Переход на метку состоится только в том случае, если на переходе Jmp формируется логическое значение TRUE, для безусловного перехода на метку необходимо константу TRUE напрямую связать с переходом.

Оператор RETURN используется для выхода из исполняемого компонента и передачи управления системе исполнения проекта.

12.5. Последовательные функциональные схемы (SFC)

В семействе МЭК-языков SFC (Sequential Function Chart) диаграммы стоят выше по отношению к остальным четырем языкам. Диаграммы SFC являются высокоуровневым графическим инструментом.

12.5.1. Шаги

Любая SFC-схема составляется из элементов, представляющих шаги и условия переходов. Шаги в SFC-программах показываются прямоугольниками. Действия шага описываются на одном из других языков стандарта МЭК 61131: IL, LD, FBD, ST. Однако шаг может содержать и вложенные программы, реализованные на языке SFC, но шаги таких вложенных программ в конечном итоге должны будут описаны на одном из языков IL, LD, FBD, ST. Код шага инкапсулирован и не отражается на SFC-схеме. О назначении шага SFC говорит только его название, а также комментарий, которым он может сопровождаться.

Шаг может быть пустым. Это не вызывает ошибки при компиляции проекта, пока не задан язык описания его действий. Если же язык все-таки задан, то допустимо сделать очистку шага или предпринять иные действия. В частности, при выбранном языке ST для компиляции проекта достаточно внутри шага поставить точку с запятой.

12.5.2. Переходы

После шага на соединительной линии размещается переход, графически выполненный в виде горизонтальной черты.

Условием перехода могут 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 ИСТИНА.

Простые условия отображаются непосредственно на диаграмме справа от черты, обозначающий переход.

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.

Переменные или прямые адреса используются в условии перехода только для чтения. В условном выражении перехода нельзя вызывать экземпляры функциональных блоков и использовать операцию присваивания.

Признаком того, что идентификатор перехода на диаграмме является отдельно реализованным условием, а не простой логической переменной, служит закрашенный угол перехода, как показано на рис. 28 для перехода *check*.

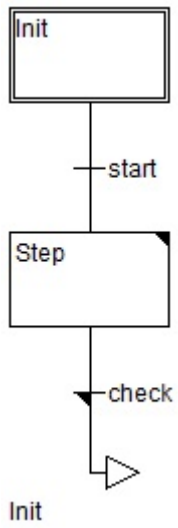


Рис. 28. SFC-схема с различными идентификаторами перехода

В качестве условия перехода может быть использована константа. Если задано значение ИСТИНА, то шаг будет выполнен однократно, за один рабочий цикл. Далее управление перейдет к следующему шагу. Если задано условие ЛОЖЬ, то шаг будет выполняться бесконечно.

12.5.3. Начальный шаг

Каждая SFC-схема начинается с начального шага, графически выполненного в виде прямоугольника с двойными вертикальными линиями по всему периметру. Наименование начального шага по умолчанию Init, но может быть изменено. Наличие начального шага обязательно, хотя он может быть и пустым.

12.5.4. Параллельные ветви

Несколько ветвей SFC могут быть параллельными. Признаком параллельных ветвей на схеме является двойная горизонтальная линия (рис. 29). Каждая параллельная ветвь начинается и заканчивается шагом. Другими словами, условие входа в параллельность всегда одно, условие выхода

тоже одно на всех. В примере, приведенном на рис. 29, условием входа в параллельные ветви является значение ИСТИНА переменной *start*.

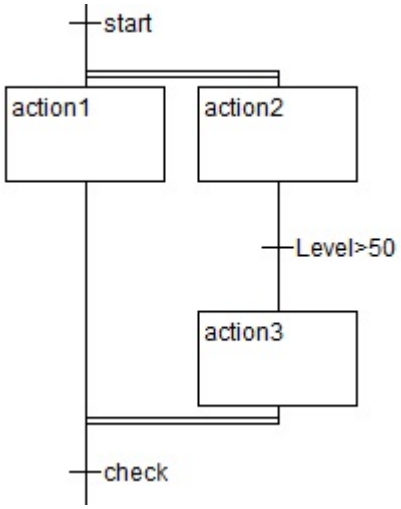


Рис. 29. Параллельные ветви

Параллельные ветви выполняются в одном рабочем цикле с приоритетом исполнения слева направо. Условие перехода, завершающее параллельность, проверяется только в случае, если в каждой параллельной ветви активны последние шаги *action1* и *action3*. Иными словами, в приведенном примере шаги *action1* и *action2* будут выполняться до тех пор, пока не выполнится условие Level > 50, после чего будут выполняться шаги *action1* и *action3* до наступления значения ИСТИНА переменной *check*.

12.5.6. Альтернативные ветви

Помимо параллельных ветвей в SFC-схеме могут быть использованы альтернативные ветви. Графически альтернативные ветви на схеме представляются одинарными горизонтальными линиями (рис. 30). У каждой альтернативной ветви имеются свои входные и выходные условия (переходы). Проверка входных условий альтернативных ветвей осуществляется слева направо. Как только условие перехода в одной из ветвей найдено, прочие оставшиеся ветви не рассматриваются. В альтернативных ветвях всегда работает только одна из ветвей, поэтому ее окончание и будет означать переход к следующему за альтернативной группой шагу.

В предложенном примере сначала проверяется условие перехода *cond1*. И только если оно не выполняется, осуществляется проверка условия *cond3*.

В языке SFC шаги могут содержать входные и выходные действия, а также к шагам могут подключаться действия с различными классификаторами по времени и типу исполнений действия (задержка, количество

раз исполнений и т. д.). Авторами данного пособия намеренно не приводится этот раздел, т. к. целевая аудитория, обучающаяся по данному пособию, не является профессиональными программистами.

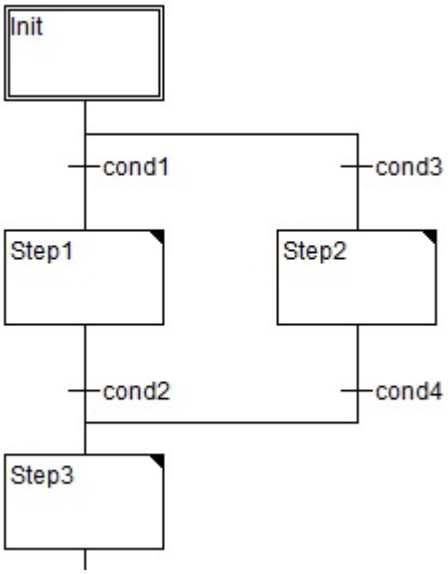


Рис. 30. Альтернативные ветви

Контрольные вопросы и задания

1. Сформулируйте основное назначение аккумулятора (result) языка IL.
2. Что такое «модификатор на языке IL»?
3. Назовите математические операции в порядке понижения приоритета на языке ST.
4. С помощью какого оператора на языке ST осуществляется выход из цикла без проверки каких-либо дополнительных условий?
5. Перечислите основные компоненты языка LD.
6. Допустим ли вызов компонентов (функции, функциональные блоки) на языке LD? Если допустим, каким образом осуществляется их подключение к релейно-контактным схемам?
7. Какими способами возможна передача информации между компонентами на языке FBD?
8. Каким образом исполняется код программы на языке LD и возможно ли изменение порядка исполнения кода?
9. При каких условиях осуществляется переход от шага к шагу на языке SFC?
10. Сформулируйте отличия в исполнении альтернативных и параллельных ветвей на языке SFC.

13. СТАНДАРТНЫЕ КОМПОНЕНТЫ

13.1. Арифметические операторы

Арифметические операторы имеют символьную форму для записи в выражениях языка ST (как показано в табл. 4). В других языках МЭК используются вызовы операторов в виде функции. В табл. 4 представлен перечень арифметических операторов.

Таблица 4

Арифметические операторы

Оператор	Символ	Действие	Типы параметров
ADD	+	Сложение	ANY_NUM, TIME
SUB	–	Вычитание	ANY_NUM, TIME
MUL	*	Умножение	ANY_NUM, TIME
DIV	/	Деление	ANY_NUM, TIME
MOD	MOD	Остаток от деления	ANY_INT
EXPT	EXPT	Возведение в степень	IN1 – ANY_NUM, IN2 – ANY_INT
MOVE	:=	Присваивание	ANY

Арифметические операторы являются перегружаемыми: тип результата операции определяется типом операндов.

В графических языках блоки MUL и ADD можно расширять, т. е. добавлять входы блоков.

Рассмотрим реализацию операции умножения на различных языках программирования стандарта МЭК 61131-3. Пусть необходимо перемножить три числа – 3, 4 и 5.

На языке IL:

```
LD 3
MUL 4,5
ST res1
```

На языке ST:

```
Res1:=3*4*5;
```

На языке FBD:

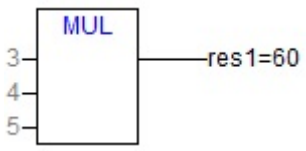


Рис. 31. Реализация примера умножения чисел на языке FBD

Язык LD также может быть применен для реализации заданного примера. Может быть разработан пользовательский функциональный блок с интерфейсными булевыми входом и выходом для вызова блока в цепи (рис. 32). В данном случае в блоке добавлены вход EN и выход ENO.

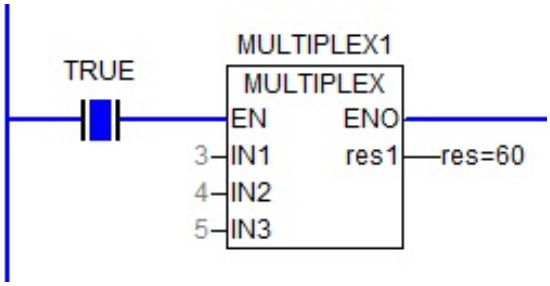


Рис. 32. Реализация примера умножения чисел на языке LD

Переменные типа TIME можно складывать между собой и вычитать. Одну переменную типа TIME можно умножать и делить на число. Результат во всех случаях будет иметь тип TIME.

Операция MOD применима только на множестве целых чисел.

Операция MOVE может иметь только один параметр совместимого типа. В явном виде MOVE встречается только в графических языках. В IL присваивание значения одной переменной или константы другой переменной выполняется парой инструкций LD, ST.

13.2. Операторы битового сдвига

Операторы сдвига применимы для типов ANY_BIT. В табл. 5 приведен перечень операторов битового сдвига.

Таблица 5

Операторы битового сдвига

Оператор	Действие
SHL	Побитный сдвиг операнда IN влево на N бит, с дополнением нулями справа
SHR	Побитный сдвиг операнда IN вправо на N бит, с дополнением нулями слева
ROR	Циклический сдвиг операнда IN вправо на N бит, старшие биты замещаются младшими
ROL	Циклический сдвиг операнда IN влево на N бит, младшие биты замещаются старшими

Рассмотрим примеры битового сдвига. Пусть задана переменная *in* типа BYTE со значением 1 (2#0000 0001), а параметр *n*, указывающий

на какое количество бит осуществлять сдвиг, равен 2, тогда результатом операции битового сдвига влево SHL получим бит в третьем разряде, что соответствует значению 4 (2#0000 0100), а циклического сдвига – вправо 64 (2#0100 0000). Реализация примеров проиллюстрирована на рис. 33.

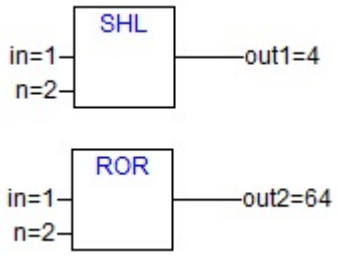


Рис. 33. Битовый сдвиг в байте

Операторы сдвига применимы для типов ANY_BIT.

13.3. Логические битовые операторы

В табл. 6 приведен перечень логических битовых операторов.

Таблица 6

Логические битовые операторы

Оператор	**Действие**
AND	Побитное И
OR	Побитное ИЛИ
XOR	Побитное исключающее ИЛИ
NOT	Побитное НЕ

Оператор NOT имеет только один входной параметр.

В FBD блоки AND, OR и XOR можно расширять, т. е. добавлять произвольное число входных параметров. В качестве примера приведем работу блока XOR с расширенным количеством входов на языке FBD (рис. 34).

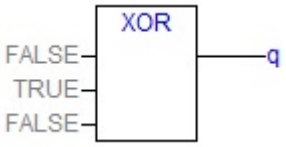


Рис. 34. Исключающее ИЛИ

13.4. Операторы выбора и ограничения

Операторы выбора и ограничения представлены в табл. 7.

В качестве примера приведем работу мультиплексора-селектора на языке FBD (рис. 35). По умолчанию блок MUX содержит три входа.

Первый вход указывает, какой из имеющихся входов передавать на выход. В рассматриваемом примере у блока добавлены еще два входа. Настроечный параметр, указывающий, значение какого входа по номеру передавать на выход, равен двум. Поэтому с учетом того, что нумерация входов начинается с нуля, на выход будет передан третий вход.

Таблица 7

Операторы выбора и ограничения

Текстовый формат	**Действие**	**Типы параметров**
OUT:= SEL(G, IN0, IN1)	Если G = FALSE, то OUT = IN0, иначе OUT = IN1	IN0, IN1: ANY, G: BOOL
OUT:= MAX(INO, INI)	Выбирается большее из значений	ANY
OUT:= MIN(INO, INI)	Выбирается меньшее из значений	ANY
OUT:= LIMIT(Min, IN, Max)	Значение, поступающее на IN, ограничивается значениями Min или Max, если IN<Min или IN>Max соответственно. OUT = IN при Min<IN<Max	ANY
OUT:= MUX(K, IN_0,..,$IN_{(K-1)}$)	OUT = IN_K по установленному значению K	IN: ANY K: ANY_INT

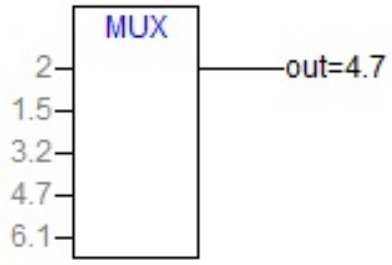


Рис. 35. Мультиплексор-селектор на языке FBD

13.5. Операторы сравнения

Следующие операторы, представленные в табл. 8, реализуют операции сравнения.

Функции имеют два входных аргумента, которые сравниваются. Если условие, определяемое логикой функции, выполняется, то возвращается значение ИСТИНА.

Таблица 8

Операторы сравнения

Оператор	Символ	Действие
GT	>	Больше
GE	>=	Больше или равно
EQ	=	Равно
LE	<=	Меньше или равно
LT	<	Меньше
NE	>	Не равно

Приведем примеры использования функции сравнения на различных языках. Сравним два числа – 4.6 и 2.7. Поскольку 4.6 больше 2.7, то функция вернет значение ИСТИНА.

На языке FBD:

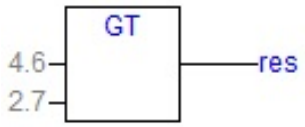


Рис. 36. Сравнение чисел

На языке ST:

```
res := 4.6 > 2.7;
```

На языке IL:

```
LD 4.6
GT 2.7
ST res
```

13.6. Математические функции

Стандартные математические функции представлены приведенными в табл. 9 операторами.

Математические функции, приведенные в табл. 9, возвращают значение типа REAL. Каждая из функций имеет один аргумент, за исключением функции EXPT. Рассмотрим функции на примере. Пусть необходимо вычислить cos значения 0.5.

На языке FBD:

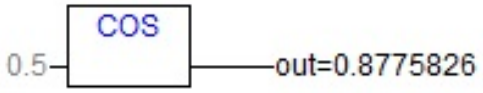


Рис. 37. Вычисление cos

Таблица 9

Математические функции

Оператор	Действие	Типы параметров
ABS	Абсолютное значение числа	BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT
SQRT	Квадратный корень числа	
LN	Натуральный логарифм числа	
LOG	Десятичный логарифм числа	
EXP	Экспонента	
SIN	Синус	
COS	Косинус	
TAN	Тангенс	
ASIN	Арксинус	
ACOS	Арккосинус	
ATAN	Арктангенс	
EXPT	Возведение в степень	

На языке ST:

```
out := cos(0.5);
```

На языке IL:

```
LD 0.5
cos
ST out
```

Для возведения числа в степень воспользуемся функцией expt. Пусть необходимо решить задачу: $1.5^{3.2}$.

На языке FBD:

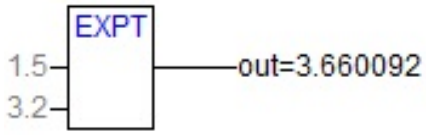


Рис. 38. Возведение в степень числа

На языке ST:

```
out := expt(1.5,3.2);
```

На языке IL:

```
LD 1.5
expt 3.2
ST out
```

13.7. Строковые функции

Строковые функции представлены приведенными в табл. 10 инструкциями.

Таблица 10

Строковые функции

INT:= LEN(STR) Возвращает длину строки
STR:= LEFT(STRING STR, INT SIZE) Возвращает левую часть STR размером SIZE
STR:= RIGHT(STRING STR, INT SIZE) Возвращает правую часть STR размером SIZE
STR:= DELETE(STRING STR,INT LEN,INT POS) Возвращает STR, удалив LEN символов с позиции POS
STR**:=** MID(STRING STR, INT LEN, INT POS) Возвращает часть STR с позиции POS длиной LEN
STR:= CONCAT(STRING STR1, STRING STR2) Возвращает конкатенацию строк STR:= STR1 + STR2
STR:= INSERT(STRING STR1, STRING STR2, INT POS) Возвращает STR1 со вставленной STR2 в позицию POS
STR:= REPLACE(STR1, STRING STR2, INT LEN, INT POS) Возвращает STR1, заменив LEN символов, с позиции POS на STR2
INT:= FIND(STRING STR1, STRING STR2) Возвращает позицию STR2 в строке STR1. Если STR2 не найдена, возвращает 0

Приведем ряд примеров, демонстрирующих работу со строковыми функциями. Пусть задана переменная типа STRING str:= 'Controller is running'. Вернем правую часть строки ‘is running’, используя различные языки программирования.

На языке ST:

```
str1 := right(10);
```

На языке FBD:

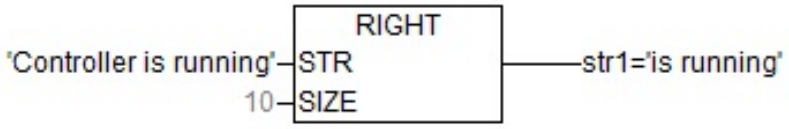


Рис. 39. Возврат правой части строки

На языке IL:

```
LD str
Right 10
ST str1
```

Рассмотрим другой пример. Пусть необходимо из строки 'Controller is running' получить строку 'Controller is not running'. Для решения задачи воспользуемся функцией Insert.

На языке FBD:

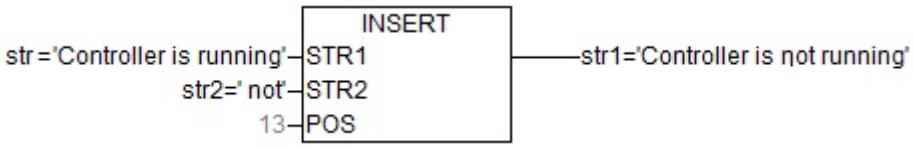


Рис. 40. Вставка строки в строку

На языке ST:

```
str1 := insert(str, str2, 13);
```

На языке IL:

```
LD str
INSERT str2, 13
ST str1
```

13.8. Таймеры

В пособии рассматриваются четыре вида таймеров на примере программной среды CoDeSys: таймер-пульс TP, таймеры с задержкой включения и выключения (TON, TOF), часы реального времени RTC.

Для правильной работы таймеров необходима аппаратная поддержка. Все экземпляры функциональных блоков таймеров «засекают» время (в CoDeSys во внутренней локальной переменной StartTime), пользуясь общими часами. При проектировании ПЛК достаточно иметь один аппаратный таймер-счетчик, увеличивающийся с постоянной частотой. Аппаратный счетчик должен иметь достаточную разрядность, чтобы исключить возможность переполнения за один рабочий цикл ПЛК.

13.8.1. Таймер-пульс TP

В табл. 11 представлены интерфейс блока TP (таймера-пульса), описание типов входных и выходных переменных.

Запуск таймера происходит по переднему фронту импульса, поступающего на вход IN. Вход PT задает длительность формируемого импульса. После запуска таймер не реагирует на изменение значения входа

IN. Выход ET отсчитывает прошедшее время. При достижении ET значения PT счетчик останавливается, и выход Q сбрасывается в 0. Рассмотрим работу таймера-пульса на примере диаграммы работы блока, приведенной на рис. 41.

Таблица 11

Интерфейс таймера-импульса TP

TP				
IN	BOOL		BOOL	Q
PT	TIME		TIME	ET

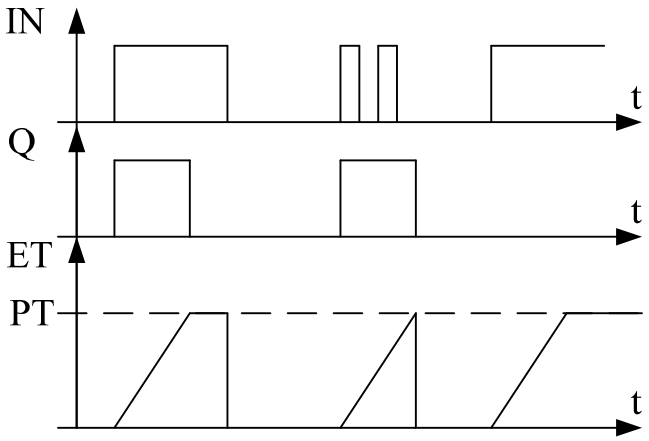


Рис. 41. Диаграмма работы блока TP

13.8.2. Таймер с задержкой выключения TOF

В табл. 12 представлен интерфейс блока TOF (таймера с задержкой выключения TOF), описание типов входных и выходных переменных.

Таблица 12

Интерфейс таймера с задержкой выключения TOF

TOF				
IN	BOOL		BOOL	Q
PT	TIME		TIME	ET

По переднему фронту импульса, поступающего на вход IN, выход Q устанавливается в TRUE. Сброс счетчика ET и начало отсчета времени происходит по каждому спаду импульсов на входе IN. Выход Q сбрасывается в FALSE через установленное время PT после спада входного сигнала IN. Если во время отсчета ET вход IN будет установлен в TRUE,

отсчет сбрасывается. Таким образом, выход Q включается по переднему фронту, а выключается после отключения IN с задержкой по времени, установленной на входе PT.

Более детально разберем работу таймера с задержкой выключения на примере диаграммы, представленной на рис. 42.

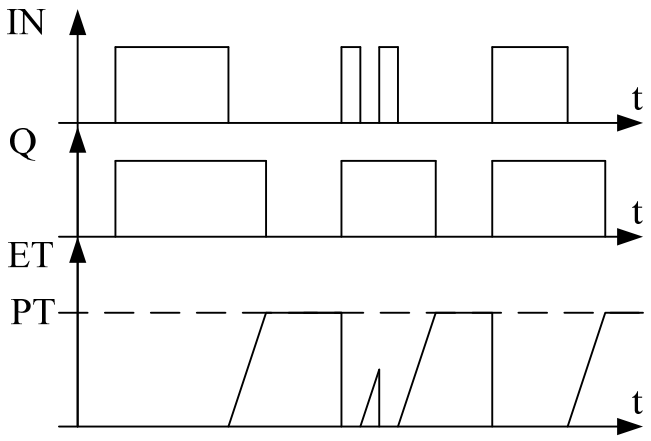


Рис. 42. Диаграмма работы блока TOF

13.8.3. Таймер с задержкой включения TON

В табл. 13 представлены интерфейс блока TON (таймера с задержкой включения), описание типов входных и выходных переменных.

Таблица 13

Интерфейс таймера с задержкой включения TON

TON				
IN	BOOL		BOOL	Q
PT	TIME		TIME	ET

По переднему фронту импульса, поступающего на вход IN, выполняется отсчет времени длительностью, установленной значением на входе PT. Выход Q будет установлен в TRUE через заданное время на входе PT, если IN будет продолжать оставаться в состоянии TRUE. Задний фронт импульса на входе IN сбрасывает отсчет времени и выход Q в FALSE. Таким образом, выход Q включается после временной задержки длительностью, установленной значением на входе PT, отсчитываемой от момента возникновения переднего фронта импульса на входе IN, а выключается по заднему фронту импульса на входе IN. Пример работы таймера с задержкой включения приведен на рис. 43.

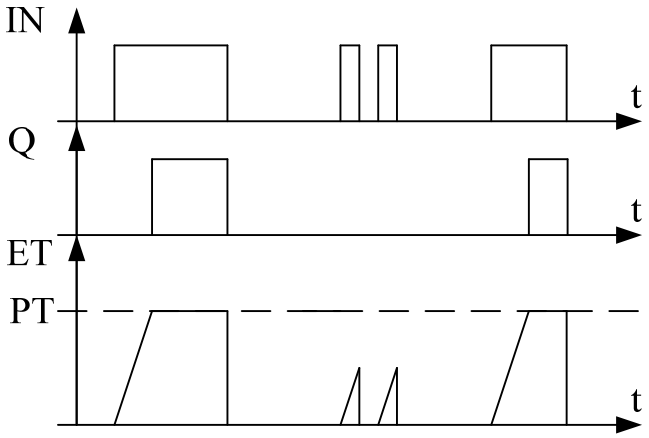


Рис. 43. Диаграмма работы блока TON

13.8.4. Часы реального времени RTC

В табл. 14 представлены интерфейс блока RTC (часы реального времени), описание типов входных и выходных переменных.

Таблица 14

Интерфейс блока RTC

RTC				
EN	BOOL		BOOL	Q
PDT	D_A_T		D_A_T	CDT

При вызове экземпляра блока RTC, пока вход EN равен FALSE, выход Q равен FALSE, а выход CDT равен DT#1970-01-01-00-00:00:00. По переднему фронту сигнала, поступающего на вход EN, в часы загружаются начальная дата и время, заданные на входе PDT. Далее начинается отсчет даты и времени на выходе CDT. Пока часы работают, выход Q = TRUE. Если EN перейдет в FALSE, CDT сбросится в начальное значение DT#1970-01-01-00-00:00:00.

Блок RTC имеет специфическую реализацию. Работа блока описанным выше способом влечет за собой ряд сложностей. В частности, в случае останова работы блока RTC на выходе CDT дата и время сбрасываются на начальное значение DT#1970-01-01-00-00:00:00 и при повторном запуске блока RTC требуется актуальное значение даты и времени на входе PDT.

13.9. Триггеры

Работа триггеров SR и RS аналогична работе электрических устройств (например, электрический пускатель с кнопками без фиксации). Для переключения используются две кнопки – «ПУСК» и «СТОП».

Кнопки не имеют механической фиксации, переключение выполняется коротким нажатием кнопок. Пускатель сам фиксирует свое состояние.

Аналогичным образом работают триггеры SR и RS. Поведение триггеров SR и RS отличается только при одновременном нажатии обеих кнопок. В блоке с доминантной включения SR побеждает «ПУСК», в блоке с доминантой выключения RS – «СТОП».

Триггеры SR и RS являются функциональными блоками, т. е. блоками с наличием внутренней памяти. Приведем пример вызова триггера RS на различных языках программирования.

На языке FBD:

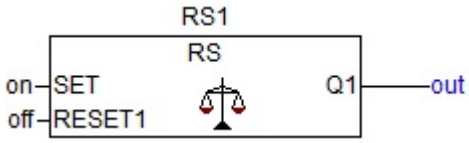


Рис. 44. Вызов триггера RS

На языке ST:

```
RS1(SET:= on , RESET1:=off );
out := RS1.Q1;
```

На языке IL:

```
CAL RS1(SET:= on , RESET1:=off )
LD RS1.Q1
ST out
```

13.9.1. Триггер с доминантой включения SR

В табл. 15 приведены интерфейс блока SR (триггера с доминантой включения), описание типов входных и выходной переменных.

Таблица 15

Интерфейс блока SR

SR				
SET1	BOOL		BOOL	Q1
RESET	BOOL			

Выход блока SR может быть в двух устойчивых состояниях: Q1 = TRUE и Q1 = FALSE. На языке ST работа блока описывается выражением

Q1 = (NOT RESET AND Q1) OR SET1.

Импульс, поступающий на вход SET1, включает выход Q1, а импульс на вход RESET – выключает. При одновременном воздействии

на входы триггера Q1 = TRUE, т. к. вход SET1 является доминантным. Рассмотрим пример: на рис. 45 представлена диаграмма работы триггера с доминантой включения SR.

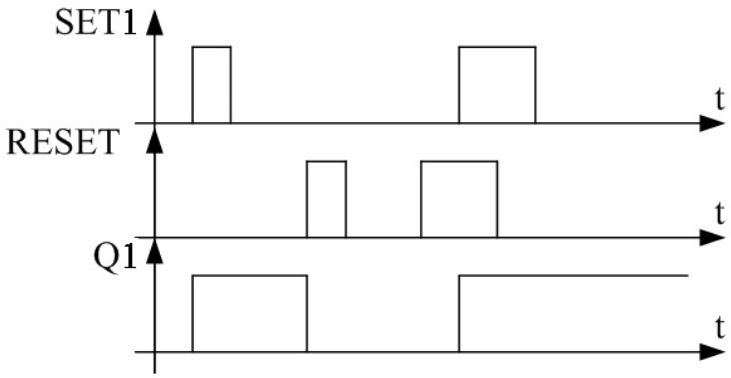


Рис. 45. Диаграмма работы триггера SR

Из диаграммы работы триггера SR видно, что выходное значение блока Q1 формируется по уровню входных значений, а не по их фронтам.

13.9.2. Триггер с доминантой выключения RS

В табл. 16 показаны интерфейс блока RS (триггера с доминантой выключения), описание типов входных и выходной переменных.

Таблица 16

Интерфейс блока RS

RS				
SET	BOOL		BOOL	Q1
RESET1	BOOL			

Как отмечалось ранее, работа триггеров RS и SR аналогична. Отличие в работе этих триггеров проявляется при одновременном воздействии на входы блоков SET и RESET1.

На языке ST работа блока описывается выражением

Q1 = NOT RESET1 AND (Q1 OR SET).

Импульс, поданный на вход SET, включает выход Q1, а на вход RESET1 – выключает. При одновременном воздействии на входы SET и RESET1 выход Q1 = FALSE, т. к. вход RESET1 является доминантным. Рассмотрим пример на рис. 46 приведем диаграмму, отражающую работу триггера с доминантой выключения RS.

Из диаграммы работы триггера RS видно, что выходное значение блока Q1 формируется по уровню входных значений, а не по их фронтам.

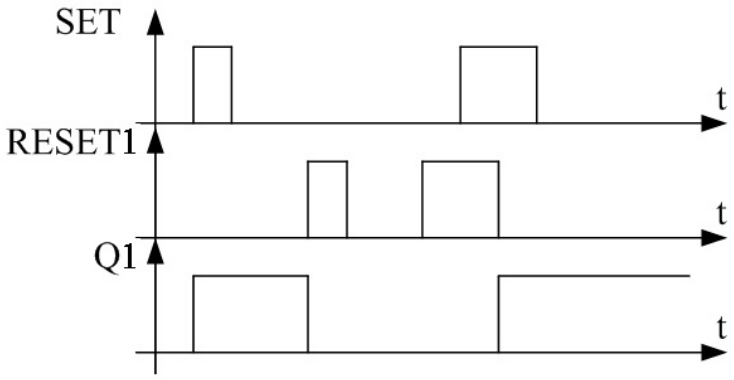


Рис. 46. Диаграмма работы триггера RS

13.10. Детектор импульсов

Детекторы переднего и заднего фронтов импульсов предназначены для фиксирования изменения состояния булевых сигналов – переход из состояния FALSE в TRUE и наоборот.

Детекторы переднего и заднего фронтов являются функциональными блоками. Рассмотрим пример вызова функционального блока R_TRIG на различных языках программирования.

На языке FBD:

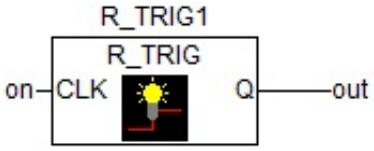


Рис. 47. Вызов детектора переднего фронта импульса

На языке ST:

```
R_TRIG1(CLK:= on);
out := R_TRIG1.Q;
```

На языке IL:

```
CAL R_TRIG1(CLK:= on)
LD R_TRIG1.Q
ST out
```

13.10.1. Детектор переднего фронта R_TRIG

В табл. 17 представлен интерфейс функционального блока R_TRIG – детектора переднего фронта.

Функциональный блок R_TRIG генерирует единичный импульс по переднему фронту входного сигнала, поступающего на вход CLK. Рассмотрим на примере диаграмму работы блока R_TRIG, представленную на рис. 48.

Таблица 17

Интерфейс блока R_TRIG

R_TRIG				
CLK	BOOL		BOOL	Q

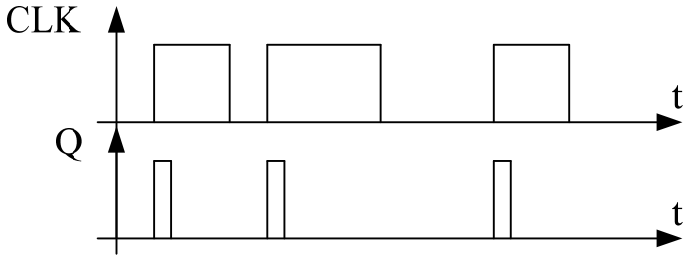


Рис. 48. Диаграмма работы блока R_TRIG

13.10.2. Детектор заднего фронта F_TRIG

В табл. 18 представлен интерфейс F_TRIG детектора заднего фронта.

Таблица 18

Интерфейс блока R_TRIG

F_TRIG				
CLK	BOOL		BOOL	Q

Функциональный блок F_TRIG генерирует единичный импульс по заднему фронту входного сигнала. Пример, отражающий диаграмму работы функционального блока FBD, приведен на рис. 49.

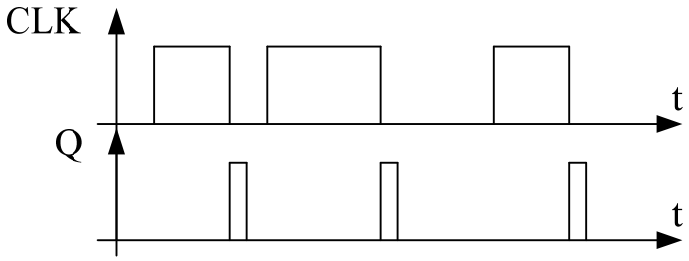


Рис. 49. Диаграмма работы блока F_TRIG

13.11. Счетчики

Существуют несколько различных типов счетчиков: инкрементный, декрементный, инкрементно-декрементный (комбинированный). Инкрементный счетчик призван считать входные импульсы, увеличивая при этом выходное значения счетчика, декрементный – уменьшая, а инкрементно-декрементный сочетает в себе обе функции.

Приведем пример вызова функционального блока CTUD – инкрементно-декрементного счетчика на языках программирования МЭК 61131-3.

На языке FBD:

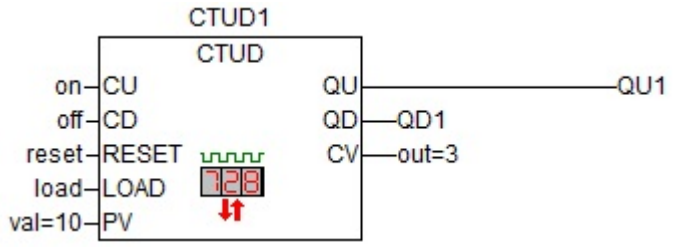


Рис. 50. Вызов инкрементно-декрементного счетчика

На языке ST:

```
CTUD1(CU:= on, CD:=off, RESET:=reset, LOAD:=load,
        PV:=val);
QU1:= CTUD1.QU;
QD1:= CTUD1.QD;
out := CTUD1.CV;
```

На языке IL:

```
CAL CTUD1(CU:= on, CD:=off, RESET:=reset, LOAD:=load,
          PV:=val)
LD CTUD1.QU
ST QU1
LD CTUD1.QD
ST QD1
LD CTUD1.CV
ST out
```

13.11.1. Инкрементный счетчик CTU

В табл. 19 представлен интерфейс инкрементного счетчика CTU.

Таблица 19

Интерфейс блока CTU

CTU				
CU	BOOL		BOOL	Q
RESET	BOOL			
PV	WORD		WORD	CV

По каждому переднему фронту импульсов, поступающих на вход CU, значение счетчика, формируемое на выходе блока CV, увеличивается на 1. Значение на выходе Q устанавливается в TRUE, когда значение

счетчика на выходе CV будет не меньше значения константы, установленной на входе блока PV. Подача импульса на вход блока RESET обнуляет выходное значение счетчика CV.

13.11.2. Декрементный счетчик CTD

В табл. 20 показан интерфейс декрементного счетчика CTD.

По каждому переднему фронту импульса, поступающего на вход счетчика CD, выходное значение CV уменьшается на 1. Значение выхода Q устанавливается в TRUE в том случае, когда выходное значение счетчика будет равным нулю. Выходное значение счетчика CV приобретает значение, установленное на входе PV, по импульсу, поступающему на вход LOAD.

Таблица 20

Интерфейс блока CTD

CTD				
CD	BOOL		BOOL	Q
LOAD	BOOL			
PV	WORD		WORD	CV

Стоит отметить, что выходное значение CV не может быть отрицательным, т. к. его тип – WORD, т. е. при поступлении импульсов на вход CD значение выхода CV не уменьшается, если уже равно нулю.

13.11.3. Инкрементно-декрементный счетчик CTUD

В табл. 21 представлен интерфейс комбинированного счетчика CTUD.

Таблица 21

Интерфейс блока CTUD

CTUD				
CU	BOOL		BOOL	QU
CD	BOOL		BOOL	QD
RESET	BOOL			
LOAD	BOOL			
PV	WORD		WORD	CV

По переднему фронту импульса, поступающего на вход RESET, обнуляется значение выхода CV, а по переднему фронту импульса, поступающего на вход LOAD, значение, установленное на входе PV, загружается в CV.

По каждому переднему фронту импульсов, поступающих на вход CU, значение счетчика CV увеличивается на 1, а по переднему фронту импульсов, поступающих на вход CD, – уменьшается на 1 до достижения 0.

Значение выхода QU устанавливается в TRUE при выполнении условия CV >= PV, иначе – FALSE.

Значение выхода QD устанавливается в TRUE при выполнении условия CV = 0, иначе – FALSE.

13.12. Побитовый доступ к целым

Для побитового доступа к целым используются следующие инструкции: чтение и запись битов, упаковка и распаковка байт.

13.12.1. Чтение бита

На табл. 22 представлен интерфейс функции чтения бита в двойном слове EXTRACT.

Функция EXTRACT типа BOOL имеет два входных параметра: DWORD X и BYTE N. EXTRACT возвращает TRUE, если бит номер N в числе X равен 1, в противном случае – FALSE. Нумерация бит начинается с 0.

Таблица 22

Интерфейс функции EXTRACT

EXTRACT			
X	DWORD		BOOL
N	BYTE		

Рассмотрим пример. Пусть на вход блока X поступает переменная *val* со значение 4 (2#0100). Необходимо узнать, какое значение во втором бите (переменная *read*, нумерация с 0). Представим реализацию на различных языках программирования.

На языке FBD:

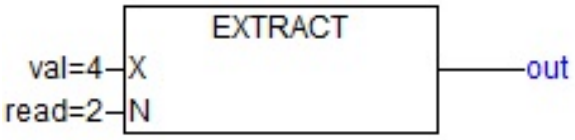


Рис. 51. Чтение бита

На языке ST:

```
Out:= EXTRACT(val,read);
```

На языке IL:

```
LD val
EXTRACT read
ST out
```

13.12.2. Запись бита

В табл. 23 приведен интерфейс функции записи бита в двойное слово.

Функция PUTBIT возвращает значение типа DWORD имеет 3 входных параметра: DWORD X, BYTE N и BOOL B. Функция возвращает значение X с установленным битом B в бит N.

Таблица 23

Интерфейс функции PUTBIT

PUTBIT			
X	DWORD		DWORD
N	BYTE		
B	BOOL		

Рассмотрим пример записи бита в двойное слово. Запишем в X = 4 (2#0100) в нулевой бит (N = 0) бит B = TRUE. Таким образом, функция вернет значение 5 (2#0101). Далее приведем реализацию на различных языках программирования.

На FBD:

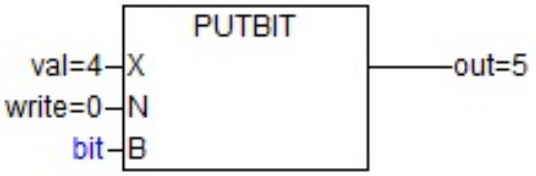


Рис. 52. Запись бита в двойное слово

На языке ST:

```
Out:= PUTBIT(val, write, bit);
```

На языке IL:

```
LD val
PUTBIT write, bit
ST out
```

13.12.3. Упаковка в байт

В табл. 24 представлен интерфейс функции упаковывания битов в байт.

Значения восьми логических переменных упаковываются в байт. Функция PACK получает восемь параметров B0, B1, …, B7 типа BOOL и возвращает значение типа BYTE, содержащее побитно значения входных параметров B0, B1, …, B7.

Таблица 24

Интерфейс функции PACK

PACK			
B0	BOOL		BYTE
B1	BOOL		
B2	BOOL		
B3	BOOL		
B4	BOOL		
B5	BOOL		
B6	BOOL		
B7	BOOL		

Рассмотрим применение функции на практике, используя различные языки программирования. Упакуем в байт 8 бит. В нулевом и втором бите запишем логические единицы. Это означает, что при упаковывании в байт на выходе сформируется значение 5 (2#0101).

На языке FBD:

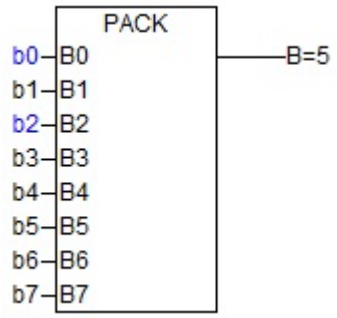


Рис. 53. Упаковка в байт

На языке ST:

```
Out:= PACK(b0, b1, b2, b3, b4, b5, b6, b7);
```

На языке IL:

```
LD b0
PACK b1, b2, b3, b4, b5, b6, b7
ST B
```

13.12.4. Распаковка байта

В табл. 25 представлен интерфейс функции распаковывания байта в биты.

В отличие от функции PACK (упаковывания в байт) распаковка байта UNPACK является функциональным блоком.

Таблица 25

Интерфейс функционального блока UNPACK

UNPACK				
B	BYTE		BOOL	B0
			BOOL	B1
			BOOL	B2
			BOOL	B3
			BOOL	B4
			BOOL	B5
			BOOL	B6
			BOOL	B7

Функциональный блок UNPACK имеет один вход типа BYTE и восемь выходов типа BOOL, выполняет обратную по отношению к функции PACK распаковку.

Приведем пример распаковывания байта с использованием функционального блока UNPACK, используя различные языки программирования стандарта МЭК 61131-3. Пусть задан входной параметр B типа BYTE со значением 5 (2#0101), осуществим распаковку на биты.

На языке FBD:

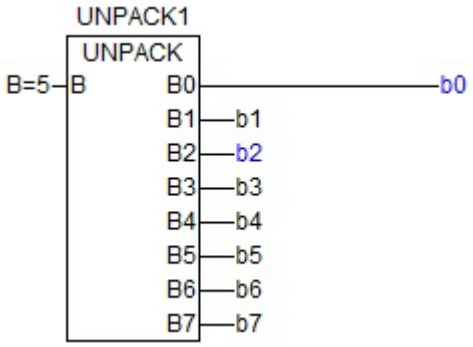


Рис. 54. Расковка байта

На языке ST:

```
UNPACK1(B:=B);
b0:= UNPACK1.B0;
b1:= UNPACK1.B1;
b2:= UNPACK1.B2;
b3:= UNPACK1.B3;
b4:= UNPACK1.B4;
b5:= UNPACK1.B5;
b6:= UNPACK1.B6;
b7:= UNPACK1.B7;
```

На языке IL:

```
CAL UNPACK1(B:=B)
LD UNPACK1.B0
ST b0
LD UNPACK1.B1
ST b1
LD UNPACK1.B2
ST b2
LD UNPACK1.B3
ST b3
LD UNPACK1.B4
ST b4
LD UNPACK1.B5
ST b5
LD UNPACK1.B6
ST b6
LD UNPACK1.B7
ST b7
```

13.13. HYSTERESIS

Интерфейс блока HYSTERESIS представлен в табл. 26.

Функциональный блок HYSTERESIS анализирует направление изменения значений сигнала, сравнивает текущее значение сигнала с заданными пороговыми значениями и вырабатывает на выходе сигнал типа BOOL.

Таблица 26

Интерфейс блока HYSTERESIS

HYSTERESIS				
IN	INT		BOOL	OUT
HIGH	INT			
LOW	INT			

Если значение сигнала IN больше верхнего порогового значения HIGH, то на выходе OUT формируется значение FALSE, а если меньше нижнего порогового значения LOW, то на выходе – TRUE. Состояние выхода OUT при значении IN в диапазоне между значениями порогов LOW и HIGH может быть как TRUE, так и FALSE и зависит от направления входа в зону, ограниченную пороговыми значениями. Если до входа в диапазон значений, ограниченных значениями LOW и HIGH, значение IN было меньше LOW, то OUT = TRUE, а если IN было больше

HIGH, то OUT = FALSE. На диаграмме, представленной на рис. 55, представлена логика работы функционального блока HYSTERESIS.

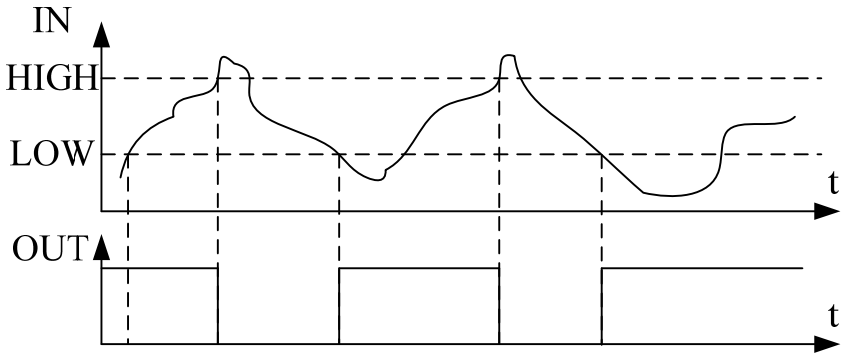


Рис. 55. Диаграмма работы блока HYSTERESIS

Рассмотрим процедуру вызова блока HYSTERESIS на различных языках программирования стандарта МЭК 61131-3.

На языке FBD:

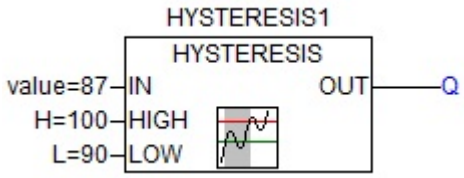


Рис. 56. Вызов экземпляра функционального блока HYSTERESIS

На языке ST:

```
HYSTERESIS1(IN:= value, HIGH:= H, LOW:= L);
Q:= HYSTERESIS1.OUT;
```

На языке IL:

```
CAL HYSTERESIS1(IN:= value, HIGH:= H, LOW:= L)
LD HYSTERESIS1.OUT
ST Q
```

13.14. Пороговый сигнализатор

Интерфейс порогового сигнализатора отражает табл. 27.

Функциональный блок LIMITALARM отслеживает соответствие значения входа IN заданному диапазону. Результат формируется с помощью логических выходов – меньше, больше, норма. Величина входного сигнала IN сравнивается с верхним и нижним пороговыми значениями HIGH и LOW соответственно. Все входные переменные IN, HIGH и LOW –

типа INT. Все три выхода O, U и IL являются логическими и информируют нас о следующем:

- O = TRUE, если IN > HIGH, иначе O = FALSE;
- U = TRUE, если IN < LOW, иначе U = FALSE;
- IL = TRUE, если LOW<= IN >= HIGH, иначе IL = FALSE).

Таблица 27

Интерфейс блока порогового сигнализатора

LIMITALARM				
IN	INT		BOOL	O
HIGH	INT		BOOL	U
LOW	INT		BOOL	IL

Рассмотрим диаграмму работу функционального блока LIMITALARM, представленную на рис. 57.

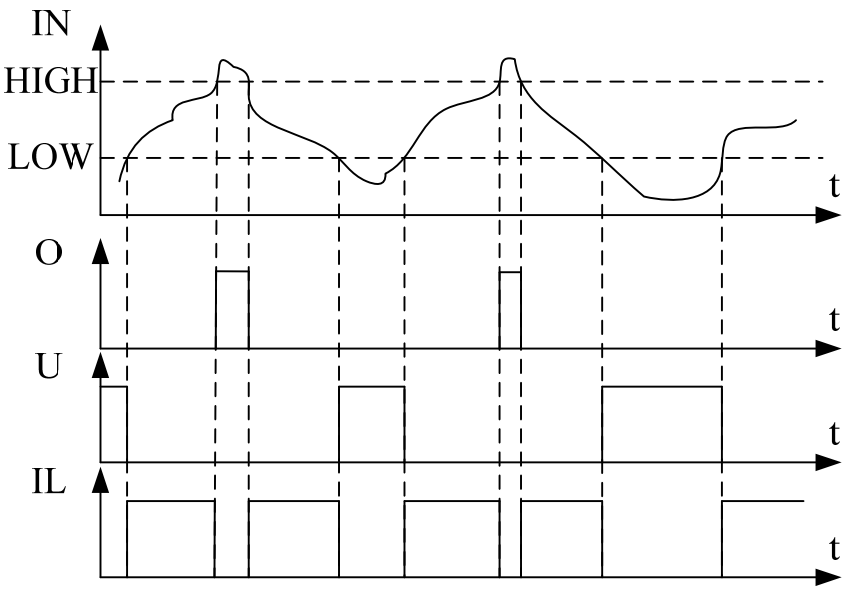


Рис. 57. Диаграмма работы порогового сигнализатора

Приведем пример вызова функционального блока LIMITALARM на различных языках программирования.

На языке FBD:

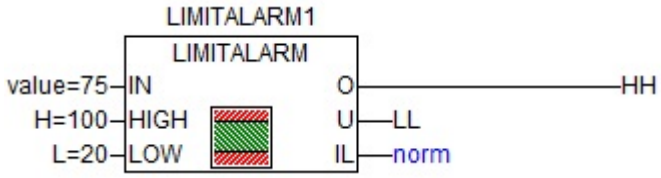


Рис. 58. Вызов экземпляра функционального блока LIMITALARM

На языке ST:

```
LIMITALARM1(IN:= value, HIGH:= H, LOW:= L);
HH:= LIMITALARM1.O;
LL:= LIMITALARM1.U;
norm:= LIMITALARM1.IL;
```

На языке IL:

```
CAL LIMITALARM1(IN:= value, HIGH:= H, LOW:= L)
LD LIMITALARM1.O
ST HH
LD LIMITALARM1.U
ST LL
LD LIMITALARM1.IL
ST norm
```

13.15. Ограничение скорости изменения сигнала

В табл. 28 приведен интерфейс функционального блока ограничения скорости изменения сигнала RAMP.

Таблица 28

Интерфейс блока RAMP

RAMP				
IN	INT, REAL		INT, REAL	OUT
ASCEND	INT, REAL			
DESCEND	INT, REAL			
TIMEBASE	TIME			
RESET	BOOL			

Функциональный блок RAMP присутствует в библиотеках CoDeSys в двух исполнениях: для целочисленных значений параметров RAMP_INT и действительных – RAMP_REAL.

Блок RAMP обеспечивает ограничение скорости изменения значений сигнала, поступающего на вход IN. Правила ограничения изменения скорости значений сигналов определяются значениями входных настроечных параметров ASCEND, DESCEND и TIMEBASE.

Формирование значения выходного параметра OUT осуществляется на основании сравнения текущего и предыдущего значений входного параметра IN.

Если значение, поступающее на вход IN, на текущем такте работы контроллера отличается от предыдущего меньше, чем величины ASCEND/TIMEBASE и DESCEND/TIMEBASE, то оно передается на выход

блока OUT, в противном случае – ограничивается. Если разность между текущим и предыдущим значениями больше, чем ASCEND/TIMEBASE или DESCEND/TIMEBASE, то на выход OUT передается ограниченное значение:

IN + ASCEND/TIMEBASE или IN – DESCEND/TIMEBASE.

Если TIMEBASE = t#0s, за интервал времени TIMEBASE берется время рабочего цикла.

При RESET = TRUE выход OUT замораживает значение на текущем такте и не изменяет его до наступления условия RESET = FALSE – после чего начнется отслеживание изменений значения с текущего такта.

Рассмотрим пример вызова функционального блока RAMP_INT на различных языках программирования.

На языке FBD:

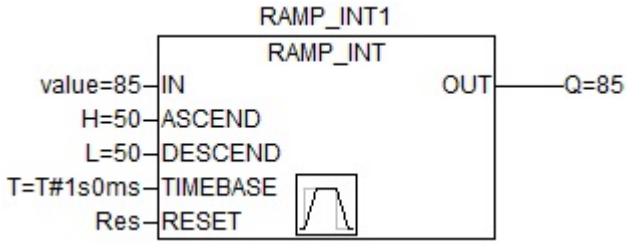


Рис. 59. Вызов экземпляра функционального блока RAMP_INT

На языке ST:

```
    RAMP_INT1(IN:= value, ASCEND:= H, DESCEND:= L,
TIMEBASE:= T, RESET:= Res);
    Q:= RAMP_INT1.OUT;
```

На языке IL:

```
     CAL RAMP_INT1(IN:= value, ASCEND:= H, DESCEND:= L,
     TIMEBASE:= T, RESET:= Res)
    LD RAMP_INT1.OUT
    ST Q
```

13.16. Интерполяция зависимостей

В табл. 29 приведен интерфейс блока интерполяции зависимостей, позволяющего вычислять значения функции между узлами.

Функциональный блок CHARCURVE выполняет кусочно-линейную интерполяцию зависимостей, заданных вектором значений узловых точек. Значение функции между узлами вычисляется по линейному закону изменения:

$$y = Y_i + \frac{Y_i - Y_{i-1}}{X_i - X_{i-1}}\left(X_i - x\right),$$

где y – вычисляемое значение функции для аргумента x, а (X_{i-1}, Y_{i-1}) и (X_i, Y_i) – узлы, между которыми находится вычисляемое значение.

Таблица 29

Интерфейс блока CHARCURVE

CHARCURVE				
IN	INT		INT	OUT
N	BYTE		BYTE	ERR
P	ARRAY			

Входной параметр P – вектор узловых значений – представляется в виде массива точек POINT, состоящего из двух переменных *X* и *Y* типа INT. Узлы в массиве должны быть представлены в порядке возрастания аргумента *X*, а их максимальное количество не может превышать 11 и быть не менее 2.

Блок CHARCURVE имеет выходной параметр ERR типа BYTE, позволяющий идентифицировать возникающие ошибки. Далее в табл. 30 приведены коды ошибок и их расшифровка.

Таблица 30

Кодификатор ошибок

Код ошибки, значение ERR	Расшифровка
0	Ошибок нет
1	Массив точек отсортирован неверно
2	Входное значение лежит за пределом области определения: IN < P[0].X или IN > P[N-1].X
4	Недопустимое значение N: N < 2 или N > 11

Рассмотрим пример использования блока интерполяции. Пусть функция представлена следующими узловыми значениями: (0,0), (50,100), (100,150), (150,250), (200,270), (250,300), (300,315), (350,320), (400,325), (450,328), (500,30). Вычислим значение для аргумента 275.

На языке FBD:

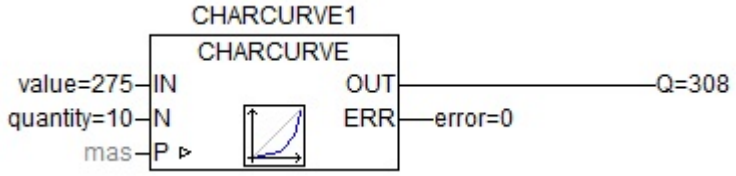


Рис. 60. Интерполяция

На языке ST:

```
CHARCURVE1(IN:=value, N:=quantity, P:=mas);
Q:=CHARCURVE1.OUT;
error:=CHARCURVE1.ERR;
```

На языке IL:

```
 CAL CHARCURVE1(IN:=value, N:=quantity, P:=mas)
LD CHARCURVE1.OUT
ST Q
LD CHARCURVE1.ERR
ST error
```

13.17. Дифференцирование

Для произведения операции дифференцирования численным методом используется функциональный блок DERIVATIVE. В табл. 31 представлен его интерфейс.

Таблица 31

Интерфейс блока DERIVATIVE

DERIVATIVE				
IN	REAL		REAL	OUT
TM	DWORD			
RESET	BOOL			

Существует достаточно большое количество алгоритмов численного дифференцирования сигналов. В частности, дифференцирование сигнала в CoDeSys осуществляется по 4 значениям сигнала: текущему и трем предыдущим:

$$\mathrm{Out} = \frac{3\left(\mathrm{IN}_0 - \mathrm{IN}_3\right) + \mathrm{IN}_1 - \mathrm{IN}_2}{10\,\mathrm{TM}},$$

где IN_i – значения сигнала, поступающие на вход IN; минимальный индекс соответствует текущему значению, максимальный – наиболее устаревшему значению; TM – шаг дискретизации времени.

Значение TRUE на входе RESET обнуляет выходное значение OUT.

Рассмотрим пример реализации дифференцирования сигнала. Установим время цикла задачи контроллера T#1s и зададим постоянную времени TM, измеряемую в миллисекундах, также равной 1 секунде, т. е. 1000 миллисекунд. Пусть на вход IN поступили следующие значения: $\mathrm{IN}_0 = 3$, $\mathrm{IN}_1 = 5$, $\mathrm{IN}_2 = 4$, $\mathrm{IN}_3 = 1$, тогда на выходе блока дифференцирования сформируется значение 0,7 с учетом приведенной выше формулы.

Приведем реализацию примера на различных языках программирования.

На языке FBD:

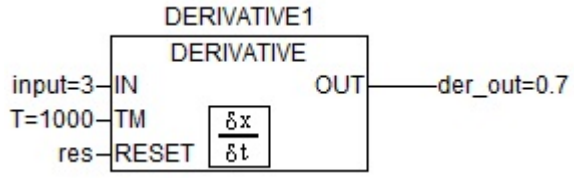


Рис. 61. Дифференцирование сигнала

На языке ST:

```
DERIVATIVE1(IN:=input, TM:=T, RESET:=res);
der_out:= DERIVATIVE1.OUT;
```

На языке IL:

```
CAL DERIVATIVE1(IN:=input, TM:=T, RESET:=res)
LD DERIVATIVE1.OUT
ST der_out
```

13.18. Интегрирование

Для осуществления операции интегрирования численным методом в CoDeSys может быть использован функциональный блок INTEGRAL. В табл. 32 представлен его интерфейс.

Таблица 32

Интерфейс блока INTEGRAL

INTEGRAL				
IN	REAL		REAL	OUT
TM	DWORD		BOOL	OVERFLOW
RESET	BOOL			

Функциональный блок INTEGRAL имеет три входа и два выхода. На вход IN подается сигнал, подлежащий интегрированию, а на входе TM задается приращение времени. На выходе OUT накапливается значение интеграла. В случае переполнения значения выхода OUT на выходе OVERFLOW формируется логическое значение TRUE, сигнализирующее о возникшем переполнении. Логическое значение TRUE, поданное на вход RESET, обнуляет выходное значение OUT.

Вычисление выходного значения OUT блока интегрирования осуществляется по формуле прямоугольников

$$OUT_i = OUT_{i-1} + TM \cdot IN_i.$$

Рассмотрим пример реализации интегрирования сигнала. Установим время цикла задачи контроллера T#1s и зададим постоянную времени TM, измеряемую в миллисекундах, также равной 1 секунде, т. е. 1000 миллисекунд. Пусть на вход IN поступили следующие значения: $IN_0 = 3$, $IN_1 = 5$, $IN_2 = 4$, $IN_3 = 1$, тогда на выходе блока интегрирования сформируется значение 13 с учетом приведенной выше формулы.

Приведем реализацию примера на различных языках программирования.

На языке FBD:

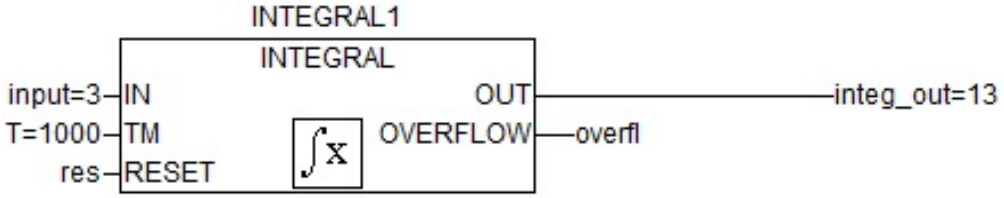


Рис. 62. Интегрирование сигнала

На языке ST:

```
INTEGRAL1(IN:=input, TM:=T, RESET:=res);
integ_out:= INTEGRAL1.OUT;
overfl:= INTEGRAL1.OVERFLOW;
```

На языке IL:

```
CAL INTEGRAL1(IN:=input, TM:=T, RESET:=res)
LD INTEGRAL1.OUT
ST integ_out
LD INTEGRAL1.OVERFLOW
ST overfl
```

13.19. ПИД-регулятор

13.19.1. Описание алгоритма регулирования

Один из наиболее часто применяемых регуляторов на производстве – ПИД-регулятор. Как правило, любая программная среда, предназначенная для программирования контроллеров, содержит в библиотеке элементов функциональный блок PID. Не является исключения и CoDeSys.

Интерфейс функционального блока PID представлен в табл. 33.

Таблица 33

Интерфейс блока PID

PID				
ACTUAL	REAL		REAL	Y
SET_POINT	REAL		BOOL	LIMITS_ACTIVE
KP	REAL		BOOL	OVERFLOW
TN	REAL			
TV	REAL			
Y_MANUAL	REAL			
Y_OFFSET	REAL			
Y_MIN	REAL			
Y_MAX	REAL			
MANUAL	BOOL			
RESET	BOOL			

В табл. 34 и 35 представлено описание входных и выходных переменных ПИД-регулятора.

Таблица 34

Описание входов блока

Вход	Описание
ACTUAL	Текущее значение регулируемого параметра
SET_POINT	Уставка (задающее воздействие)
KP	Коэффициент пропорциональности
TN	Коэффициент интегрирования
TV	Коэффициент дифференцирования
Y_MANUAL	Задание, передаваемое на выход в ручном режиме
Y_OFFSET	Начальное значение
Y_MIN	Минимальное допустимое значение
Y_MAX	Максимальное допустимое значение
MANUAL	Режим работы регулятора
RESET	Сброс

Таблица 35

Описание выходов блока

Y	Управляющий сигнал
LIMITS_ACTIVE	Индикатор достижения Y пороговых значений Y_MIN или Y_MAX (Y≤ Y_MIN или Y≥Y_MAX)
OVERFLOW	Индикатор переполнения выходного значения Y

Выходное значение ПИД-регулятора с зависимыми настройками Y формируется по следующей формуле:

$$Y = Y\_OFFSET + KP\left(e + \frac{1}{TN}\int edt + TV\frac{de}{dt}\right),$$

где e – рассогласование между текущим значением регулируемого параметра ACTUAL и задающим воздействием SET_POINT; значение Y ограничивается пороговыми Y_MIN и Y_MAX.

ПИД-регулятор помимо автоматического режима работы (формирование значения выхода Y по вышеуказанной формуле) имеет ручной режим. Перевод регулятора в ручной режим работы осуществляется логической единицей (TRUE), поданной на вход MANUAL. При переводе регулятора в ручной режим выходу Y присваивается значение входа Y_MANUAL.

Рассмотрим пример вызова экземпляра функционального блока ПИД-регулирования на различных языках программирования.

На языке FBD:

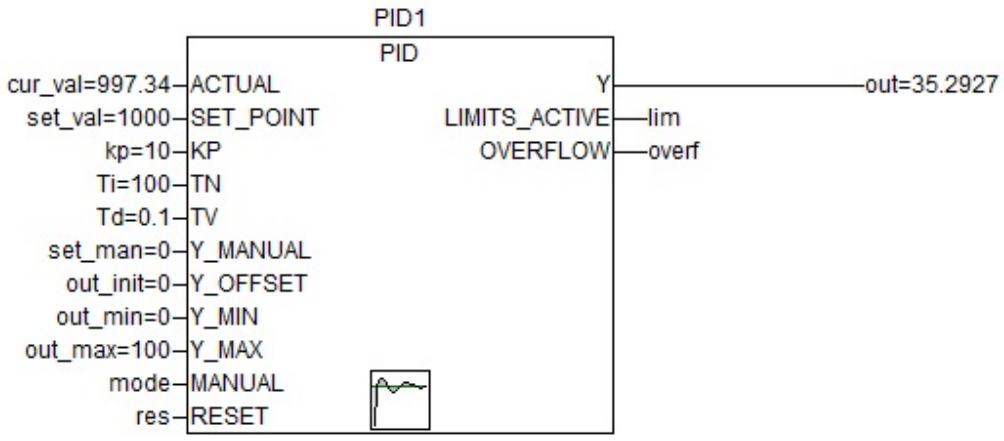


Рис. 63. ПИД-регулирование

На языке ST:

```
PID1(ACTUAL:=cur_val, SET_POINT:=set_val, KP:= kp,
TN:= Ti, TV:= Td, Y_MANUAL:= set_man,
Y_OFFSET:= out_init, Y_MIN:= out_min, Y_MAX:= out_max,
RESET:=res);
out:= PID1.Y;
lim:= PID1.LIMITS_ACTIVE;
overf:= PID1.OVERFLOW;
```

На языке IL:

```
CAL PID1(ACTUAL:=cur_val, SET_POINT:=set_val, KP:= kp,
TN:= Ti, TV:= Td, Y_MANUAL:= set_man,
Y_OFFSET:= out_init, Y_MIN:= out_min, Y_MAX:= out_max,
RESET:=res)
```

```
LD PID1.Y
ST out
LD PID1.LIMITS_ACTIVE
ST lim
LD PID1.OVERFLOW
ST overf
```

13.19.2. Настройка ПИД-регулятора

Существуют различные подходы к настройке ПИД-регуляторов: метод Циглера–Никольса, метод Шубладзе, метод Шеделя, метод Куна и др.

Наиболее популярным методом настройки является метод Циглера–Никольса. Описание метода опубликовано в далеком в 1942 г. Алгоритм метода включает в себя следующие шаги:

- в работающей системе отключаются интегральная и дифференциальная составляющие ПИД-регулятора (т. е. $k_{\text{и}} = \infty$, $k_{\text{д}} = 0$). Иными словами, из ПИД-регулятора формируют П-регулятор;
- постепенно увеличивая $k_{\text{п}}$ с одновременной подачей незначительного ступенчатого сигнала задания добиваются возникновения в контуре управления незатухающих колебаний с периодом $T_{\text{кр}}$. Это соответствует выведению контура управления на границу устойчивости. При возникновении данного режима работы фиксируются значения критического коэффициента усиления регулятора $k_{\text{кр}}$ и периода критических колебаний в системе $T_{\text{кр}}$;
- далее рассчитываются и устанавливаются настроечные параметры ПИД-регулятора по формулам: $k_{\text{п}} = 0,6k_{\text{кр}}$; $k_{\text{и}} = \frac{T_{\text{кр}}}{2}$; $k_{\text{д}} = \frac{T_{\text{кр}}}{8}$.

Данный метод удобно использовать тогда, когда достаточно обеспечить довольно посредственные динамические качества замкнутой системы управления устойчивыми объектами, параметры которых точно измерить не удается. Большинство специалистов называют главными недостатками метода большое перерегулирование и выведение системы на границу устойчивости.

Главным достоинством этого метода является простота его выполнения, поскольку нет необходимости в размыкании контура системы управления и дополнительном исследовании его объекта. Однако по условиям функционирования далеко не каждый объект можно выводить на границу устойчивости. Зачастую это является невыполнимым условием. Заметим, что система с регулятором, настроенным этим методом, обладает большим перерегулированием.

Общим недостатком упомянутых методов является аппроксимация объектов управления моделями, которые не содержат ни нулей, ни звеньев запаздывания, эти методы являются инженерными и довольно просты в применении.

В настоящее время производители программного обеспечения для программирования логических контроллеров нередко разрабатывают отдельный программный модуль по настройке ПИД-регуляторов. Чаще всего это платный программный модуль. Как правило, его работа основана на подборе значений коэффициентов регулятора на основе информации, полученной от объекта управления путем оценки его реакции на незначительные ступенчатые воздействия в процессе эксплуатации.

Контрольные вопросы и задания

1. Сформулируйте ключевое отличие операторов битового сдвига SHL и ROL.
2. Сравните работу операторов OR и XOR.
3. Каким оператором ограничивается минимальное допустимое и максимальное допустимое значение входного параметра?
4. Какого типа выходной параметр операторов сравнения?
5. При помощи какого оператора осуществляется возведение в степень числа?
6. При помощи какого оператора можно объединить строки?
7. Есть ли задержка отключения булевого выходного параметра у таймера TON? Если есть, то какова ее длительность?
8. Сформулируйте отличия в логике работы триггеров SR и RS.
9. Какова длительность импульса, формируемого детекторами переднего и заднего фронта?
10. Может ли сформироваться отрицательное значение на выходе декрементного счетчика?
11. Какого типа значение возвращает функция записи бита PUTBIT?
12. Анализируется ли блоком HYSTERESIS направление изменения значений входного параметра?
13. Какие состояния сигнала отслеживаются пороговым сигнализатором?
14. В каком случае входное значение сигнала блока ограничения скорости изменения сигнала передается на выход?
15. Какой закон используется для вычисления межузловых значений функции при использовании функционального блока интерполяции зависимостей CHARCURVE?
16. Сколько значений функции используется для вычисления ее производной при использовании блока DERIVATIVE?
17. По какой формуле рассчитывается выходное значение блока интегрирования INTEGRAL?
18. По какому закону формируется выходное значение блока PID?
19. Приведите примеры алгоритмов настройки ПИД-регулятора.

14. ИНТЕРФЕЙС СИСТЕМЫ УПРАВЛЕНИЯ

Функционирование автоматизированных систем управления подразумевает участие человека. Интерфейс, обеспечивающий взаимодействие автоматизированной системы и человека, называется человеко-машинным интерфейсом. В литературе встречаются различные аббревиатуры, подразумевающие под собой человеко-машинный интерфейс:

- ЧМИ – человеко-машинный интерфейс;
- MMI – Man-Machinery Interface;
- HMI – Human-Machinery Interface.

Для интерфейса управления технологическим процессом введен термин «SCADA-система» – Supervisory Control And Data Acquisition, буквально означающий диспетчерское управление и сбор данных. Однако на практике SCADA-система включает более широкий функционал и решает большее количество задач.

14.1. Функции SCADA

Вопросы функциональных возможностей уже затрагивались в разрезе функций АСУ и ОС (разделы 4, 5). Рассмотрим детальнее функциональные возможности SCADA-систем.

В литературе функциональные возможности SCADA-систем классифицируют по различным критериям и признакам и объединяют в различные группы.

По решаемым задачам функциональные возможности можно выделить в следующие группы:

- представление информации о технологическом процессе различными средствами визуализации;
- выработка управляющих воздействий на технологическое оборудование в автоматическом режиме;
- ручное дистанционное управление;
- регистрация информации и хранение данных;
- безопасное управление технологическим процессом;
- общесистемные функции.

Несомненно, ключевая особенность SCADA-системы – это наличие инструментария для разработки человеко-машинного интерфейса, т. к. многие из вышеперечисленных функций могут выполнять другие технические устройства АСУ ТП, в частности, ПЛК или специализированные серверы.

SCADA-система по своей сути состоит из двух частей: среда разработки и система исполнения (Runtime). Именно среда разработки, наличие различных библиотек, поддержка языков программирования, поддержка COM, DCOM, OLE-объектов, ActiveX-элементов, механизмы

безопасного функционирования, наличие различных фирменных сервисов определяют удобство, качество и продвинутость интерфейса автоматизированной системы управления.

SCADA-система – это система, работающая в реальном времени и реагирующая на изменения как стороны технологического процесса, так и оператора. Изменения в технологическом процессе тесно связаны с такими понятиями, как аларм и событие.

Под событием понимаются некие изменения в системе – включение или отключение какого-либо оборудования, открытие или закрытие арматуры и т. д. Событие не требует незамедлительного вмешательства оператора, а лишь информирует о произошедших изменениях в системе.

Аларм, в отличие от события, – это сигнал тревоги в силу происходящих событий в технологическом процессе. Требует принятия либо со стороны оператора, либо со стороны системы управления технологическим процессом в автоматическом режиме.

В качестве примеров, описывающих понятие «аларм», могут быть приведены высокая температура, повышенное давление трубопровода, низкий уровень масла в маслобаке, высокая концентрация СН-веществ и т. д.

Существует ряд классификаций алармов, в основе которых лежат различные критерии.

По стадии отклонения параметра от нормы алармы делятся на предаварийные и аварийные. Предаварийные алармы сигнализируют, что значение достигло некого порога и приближается к критическому. Предаварийные алармы не влекут за собой действий со стороны системы управления, скорее адресованы оператору технологического процесса, которому необходимо предпринять комплекс мер по стабилизации технологического процесса. Аварийные алармы, как правило, влекут за собой действия со стороны системы управления или системы противоаварийной защиты, например останов двигателей, включение приточно-вытяжной вентиляции, открытие или закрытие арматуры и т. д.

По статусу алармы делятся на подтвержденные или неподтвержденные (квитированные и не квитированные). Возникновение аларма влечет за собой срабатывание сигнализции, а со стороны оператора квитирования (подтверждения) – сообщения о возникновении аларма. По квитированию отключается звук сигнализации, но сохраняется подсветка параметра до его стабилизации.

На рис. 64 представлена графическая интерпретация формирования аларма с выделением зон: «Норма», «Предупреждение» и «Авария».

Условия формирования предаварийного аларма:

- нарастание сигнала и $y(a) < y(t) < y(b)$;
- спад сигнала и $y(e) < y(t) < y(f)$.

Условия формирования аварийного аларма:

- нарастание сигнала и $y(b) < y(t)$;
- спад сигнала и $y(f) > y(t)$.

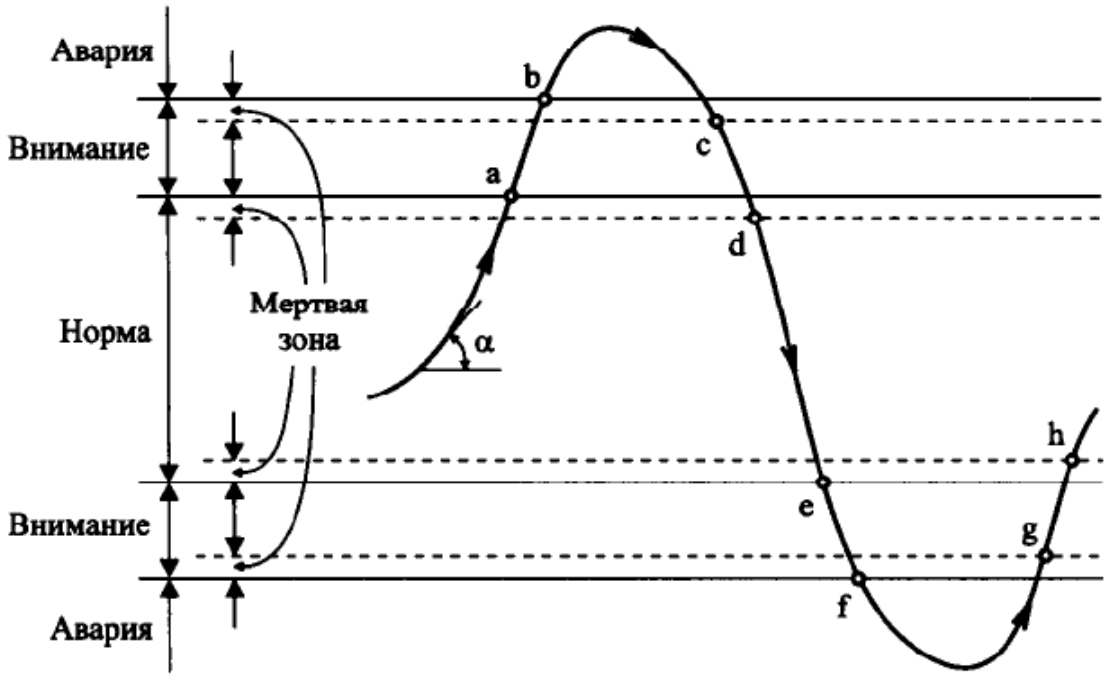


Рис. 64. Реализация срабатывания алармов

У каждого порогового значения «Внимание» или «Авария» имеется зона нечувствительности с целью исключения ложных срабатываний алармов. Зоны нечувствительности располагаются по направлениям формирования алармов. Снятие алармов осуществляется после выхода из зоны нечувствительности по направлению к зоне «Норма».

14.2. Человеко-машинный интерфейс

Ключевая задача любого SCADA-пакета – разработка человеко-машинного интерфейса АСУ ТП. Во многом скорость и качество получения результат зависит от инструментария среды разработки: наличия богатых элементной базой библиотек, возможностью использования языков программирования, поддержкой COM-объектов и ActiveX-элементов, наличия различных сервисных функций и т. д.

Разработка интерфейса включает в себя:

- разработку графической части интерфейса технологического процесса и системы управления (мнемосхема, графики, тренды, рапорты, отчеты, таблицы, кнопки, сообщения, меню, окна и т. д.);
- формирование связи элементов графики интерфейса с тегами;

- разработку алгоритмов и программ (в случае поддержки языков программирования);
- коммуникационную настройку связи с OPC-серверами, базами данных, смежными системами;
- тестирование и отладку (в некоторых случаях с использованием имеющихся генераторов сигналов и эмуляторов).

14.3. Диспетчерское управление

Любая современная SCADA-система способна выполнять функции диспетчерского управления и решать такие задачи, как:

- прием и обработка команд оператора;
- формирование подсказок для оператора при управлении технологическим процессом;
- представление информационных сообщений;
- ведение журнала действий оператора и событий;
- предаварийная и аварийная сигнализация;
- извлечение из архива регистрируемой информации в требуемых разрезах и заданной глубины;
- формирование отчетов и рапортов по требованию оператора;
- ведение различной статистики, связанной с оборудованием и технологией получения продукции и т. д.

14.4. Функции системы автоматического управления

В АСУ ТП с иерархической структурой, как правило, функции управления, контуры управления и блокировки, выполняет ПЛК. Это связано с рядом факторов:

- прямое назначение устройства;
- наличие операционной системы реального времени;
- быстродействие;
- имеющиеся функциональные возможности (наличие готовых к использованию алгоритмов управления, в частности функциональный блок ПИД-регулирования).

Однако в некоторых случаях функции автоматического управления может брать на себя SCADA-система и решать следующие задачи:

- блокировка работы технологического оборудования при определенных заданных условиях;
- автоматическое регулирование параметров;
- формирование перечня управляющих действий по сценарию и др.

В этом случае SCADA-система должна обладать соответствующими техническими и инструментальными возможностями, а решение вышеуказанных задач позволяет в некоторых случаях отказаться от использования ПЛК.

14.5. Хранение истории процесса

Хранение истории процесса является важной задачей SCADA-системы и позволяет решать следующие задачи:

- анализ причин в различных внештатных ситуациях – аварийный останов системы, ошибочные действия персонала, получение бракованной продукции, выявление первопричины;
- установление причинно-следственной связи действий оператора и работоспособности оборудования;
- ведение статистики, в частности время работы на отказ и прочее;
- представление информации в различных разрезах по параметрам и времени;
- исследование и модернизация технологического процесса.

Как правило, хранение истории обеспечивают по наиболее важным параметрам. Это связано с выделяемыми мощностями под хранение информации. Глубина архива истории процесса также может быть различной и согласовывается с Заказчиком на стадии формирования технического задания.

14.6. Безопасность SCADA

В последнее время вопросам безопасности SCADA-систем уделяется гораздо большее внимание. Вполне возможно толчком этому послужили включение станций оператора в глобальную сеть, интеграция и глобализация систем управления. Все большую популярность набирает удаленный доступ, облачные сервисы и хранилища. Перечисленные факторы ведут к снижению уровня безопасности системы и принятия различных мер по ее повышению. В качестве таких мер могут быть:

- гибкая настройка прав доступа к системе в зависимости от функциональных задач, решаемых специалистом;
- защита информации за счет использования специальных протоколов, шифрования данных;
- применение межсетевых экранов и брандмауэров;
- использование специализированного программного обеспечения для защиты от кибератак;
- применение аппаратных ключей доступа.

14.7. Свойства SCADA

Выбор SCADA-пакета при разработке АСУ ТП осуществляется исходя из различных критериев, особенностей и свойств. Эти свойства можно разделить на следующие группы:

- инструментальные;
- эксплуатационные;

- открытость;
- рентабельность.

Далее детально рассмотрим группы перечисленных свойств.

14.7.1. Инструментальные свойства

Инструментальные свойства SCADA-пакета – это свойства, связанные с доступными инструментами среды разработки SCADA, фирменными сервисами, дружественностью интерфейса SCADA и др., что, в свою очередь, определяет:

- скорость разработки проекта;
- доступность освоения;
- наличие функциональных возможностей для обработки данных различной степени сложности;
- поддержку языков программирования стандарта МЭК 61131;
- поддержку COM, DCOM, OLE – объектов, ActiveX – элементов;
- наличие графического редактора с широкой элементной базой и графическими примитивами;
- наличие внутреннего генератора сигналов и эмулятора, позволяющих оперативно осуществить отладку разработанной SCADA;
- наличие технической документации по SCADA на различных языках;
- техническое сопровождение разработчика SCADA и др.

14.7.2. Эксплуатационные свойства

Эксплуатационные свойства SCADA – это свойства, определяющие качество работы SCADA в процессе ее функционирования. Выделяют следующие свойства:

- надежность;
- устойчивость к сбоям и ошибкам;
- самовосстанавливаемость;
- защищенность информации;
- наличие средств автоматического сохранения информации при сбоях и отсутствия питания;
- наличие возможности подхвата последних актуальных значений в системе при восстановлении после сбоя;
- возможность реализации резервирования SCADA;
- поддержка многоэкранного режима работы SCADA и др.

14.7.3. Открытость SCADA-систем

Рентабельность SCADA-системы во многом зависит от ее способности интегрироваться в иные разноуровневые автоматизированные системы управления различных производителей. Степень открытости

SCADA-системы определяется способностью обмена информацией с различными программными модулями, разработанными пользователями или другими производителями.

Как правило, это обеспечивается путем разработки SCADA с поддержкой COM-, DCOM-, OLE-объектов, ActiveX-элементов. Технология совместимости SCADA-систем, ПЛК и БД различных серверов определяется стандартом OPC, поддержкой языков программирования стандарта МЭК 61131, а также применением интерфейса ODBC и/или OLE DB.

14.7.4. Рентабельность SCADA-систем

Рентабельность SCADA-системы определяет степень ее экономической эффективности – экономический эффект от внедрения к сумме затрат на внедрение и поддержание SCADA-системы в рабочем состоянии.

К свойствам SCADA, определяющим ее рентабельность, можно отнести следующие:

- доступная масштабируемость – возможное приобретение SCADA на различное количество тегов (параметров), зависящее от объекта автоматизации и решаемых задач;
- проектная компонуемость SCADA-системы. Заказчику доступен выбор приобретаемых модулей системы, в конечном счете определяющий стоимость всей системы;
- ценовая политика сопровождения SCADA-систем;
- обновляемость версий SCADA;
- стоимость обучения клиентов;
- политика и стоимость технической поддержки;
- репутация разработчика SCADA;
- политика ценообразования SCADA.

Зачастую универсальность SCADA-системы ведет к снижению степени ее эффективности.

Контрольные вопросы и задания

1. Перечислите основные функции SCADA-систем.
2. Какие действия включает в себя разработка человеко-машинного интерфейса?
3. Какие задачи решаются SCADA в рамках диспетчерского управления?
4. Перечислите меры, повышающие безопасность SCADA.
5. Перечислите свойства SCADA и их характеристики.

15. OPC-СЕРВЕР

Для обеспечения совместной работы средств автоматизации различных производителей международной организацией OPC Foundation разработан стандарт OPC. В состав этой международной организации входит более 400 организаций, работающих в соответствующем секторе экономики, а в числе основателей числятся такие, компании как Fisher-Rosemount, Rockwell Software и др.

15.1. Обзор стандарта OPC

Основной целью разработки стандарта OPC является обеспечение условий совместной работы оборудования, устройств и элементов различных производителей, работающих на различных программно-аппаратных платформах, находящихся в разных сетях.

Решение такой задачи позволяет избежать индивидуальной адаптации SCADA к остальному оборудованию – ПЛК, платы расширения, интеллектуальные датчики, исполнительные устройства, иные элементы АСУ ТП.

Поддержка стандарта OPC подразумевает работу в клиент-серверном режиме. На станции оператора разворачивается OPC-сервер и по протоколам обмена данными устройствами АСУ ТП (SCADA, ПЛК и др.) осуществляется чтение и запись информации.

После внедрения стандарта OPC, как правило, производители аппаратной части АСУ ТП начали сопровождать производимое оборудование OPC-сервером.

Стандарт OPC регламентирует только интерфейс, предоставляемый сервером для клиентов, а методы взаимодействия стандартом OPC не регламентированы, разработчики определяют их самостоятельно.

OPC стандарт включает в себя несколько частей и описывает:

- OPC DA (OPC Data Access) – спецификация для обмена данными между клиентом (в частности, SCADA) и аппаратной частью (ПЛК, платы расширения, модули ввода-ввода и др.) в реальном времени;
- OPC Alarms & Events (A&E) – спецификация для уведомления клиента о событиях и сигналах тревоги, отправляемых клиенту в случае их возникновения. OPC сервер отправляет алармы, действия оператора, сообщения, результаты контроля состояния системы;
- OPC HDA (Historical Data Access) – спецификация для доступа к данным, сохраненным в архиве;
- OPC Batch – спецификация для особых физико-химических технологических процессов обработки материалов, не относящихся к непрерывным. OPC-сервер обеспечивает обмен между клиентом и сервером

рецептами, характеристиками технологического оборудования, условиями и результатами обработки;

- OPC Data eXchange – спецификация для обмена данными между двумя OPC DA-серверами через сеть Ethernet;
- OPC Security – спецификация, регламентирующая методы доступа клиентов к серверу, обеспечивающие защиту информации от ее несанкционированного изменения;
- OPC XML-DA – спецификация, регламентирующая представление данных с помощью языка XML, веб-технологий и сообщений SOAP;
- OPC Complex Data – дополнительные спецификации к OPC DA и XML-DA, описывающие работу серверов со сложными типами данных, такими как бинарные структуры и XML-документы;
- OPC Commands – спецификация, описывающая команды, позволяющие клиентам и серверам идентифицировать, посылать и контролировать команды, реализуемые в аппаратной части системы управления;
- OPC Unified Architecture – модернизированная спецификация, описывающая передачу данных в промышленных сетях и не использующая технологию COM/DCOM.

Наиболее часто применяемые в АСУ ТП – OPC DA и OPC HDA.

15.2. OPC DA-сервер

OPC DA-сервер является наиболее часто используемым в АСУ ТП. Он организует обмен данными между сервером и клиентами, данные имеют следующие поля:

- значение;
- качество;
- временная метка.

Существуют четыре стандартных режима чтения данных из OPC-сервера:

1) синхронный режим: клиент посылает запрос серверу и ждет от него ответ;

2) асинхронный режим: клиент отправляет запрос и сразу же переходит к выполнению других задач, а по наличию уведомления от сервера забирает запрошенные данные;

3) режим подписки: клиент сообщает серверу список тегов, значения которых сервер должен отправлять клиенту только в случае их изменения (с учетом зоны нечувствительности);

4) режим обновления данных: клиент вызывает одновременное чтение всех активных тегов.

Чтение данных может осуществляться как из кэш OPC-сервера, так и из самих устройств. Очевидно, что скорость чтения из кэш выше, но он

может содержать устаревшие значения параметров, полученные при последнем обновлении данных сервера. С целью разгрузки процессора выделяют различные группы параметров и устанавливают разное время обновления значений в зависимости от назначения и степени их важности.

Запись данных в устройствах осуществляется либо асинхронным, либо синхронным методом напрямую в устройство без буферизации. При синхронном режиме записи данных она продолжается до тех пор, пока от устройства не поступит уведомление о том, что запись выполнена. Все это время клиент находится в режиме ожидания. При асинхронной записи клиент отправляет данные устройству и переключается на другие задачи, получая позже от сервера уведомление о записи данных.

Следует отметить, что OPC-сервер и OPC-клиенты могут работать только на ОС и ПЛК с операционными системами, поддерживающими технологию DCOM. Клиентская программа и OPC-сервер могут быть установлены как на одном компьютере, так и на разных компьютерах, находящихся в сети. При этом каждая из ОС сети может иметь OPC-сервер и подключенные к ней устройства.

В такой системе любой OPC-клиент с любого компьютера может обращаться к любому OPC-серверу, включая к расположенному на другом компьютере сети, что достигается благодаря технологии DCOM.

При использовании оборудования разных производителей на компьютере могут быть установлены несколько OPC-серверов, но следует отметить, что каждый из серверов требует свой COM-порт, поэтому число портов должны быть не менее числа серверов.

15.3. OPC HDA-сервер

Основное назначение OPC HDA-серверов – это предоставление информации клиентам, находящейся в хранилище данных. Стандарт распространяется только на интерфейсы для взаимодействия HDA-сервера с клиентскими программами и не регламентирует механизмы получения или хранения данных.

Спецификация OPC HDA устанавливает стандарт на интерфейсы COM-объекта и методы его использования. Структура сервера и методы взаимодействия с клиентами полностью аналогичны общей идеологии OPC и описанному ранее OPC DA. Например, OPC клиент может подсоединяться к нескольким OPC HDA-серверам разных производителей и быть установлен на разных компьютерах в сети Ethernet.

Существует два типа HDA-серверов:

- простой сервер данных предыстории для построения графиков (трендов);
- сервер для хранения данных в упакованном виде с возможностью их обработки и анализа.

15.4. OPC UA-сервер

Технология OPC не лишена недостатков:

- доступность только на операционных системах семейства Microsoft Windows;
- связь с технологией DCOM, исходные коды которой являются закрытыми;
- бывают проблемы конфигурирования, связанные с DCOM;
- неточные сообщения DCOM о прерываниях связи;
- неприспособленность DCOM для обмена данными через Интернет;
- неприспособленность DCOM для обеспечения информационной безопасности.

В этой связи была разработана стандартная спецификация для обмена данными в системах автоматизации под названием «OPC Unified Architecture – «OPC с унифицированной архитектурой».

Стандарт OPC UA устанавливает методы обмена сообщениями между OPC-сервером и клиентом, которые не зависят от программно-аппаратной платформы, от типа взаимодействующих систем и сетей. OPC UA обеспечивает надежную и безопасную коммуникацию, противодействие вирусным атакам, гарантирует идентичность информации клиента и сервера.

В этом стандарте используется понятие объекта, под которым понимается физический или абстрактный элемент системы. Примерами объектов могут быть физические устройства, включающие их системы и подсистемы. Датчик температуры, к примеру, может быть представлен как объект, который включает в себя значение температуры, набор параметров сигнализаций и границы их срабатывания. Объект по аналогии с объектно ориентированным программированием определяется как экземпляр класса, а класс рассматривается как тип данных. Объекты включают в себя переменные, события и методы.

OPC UA использует несколько различных форматов данных, основными из которых являются бинарные структуры и XML-документы. Формат данных может быть определен поставщиком OPC-сервера или стандартом. Для работы с произвольными форматами клиент может запросить у сервера информацию об описании этого формата. Во многих случаях используется автоматическое распознавание формата данных во время их передачи.

OPC UA обладает высокой робастностью данных и уведомлений о событиях. Робастность обеспечивается механизмом быстрого обнаружения ошибок коммуникации и восстановления данных.

Серверы могут иметь доступ как к текущим, так и архивированным данным, к событиям и аварийным сигналам. OPC UA может быть внедрен в различные коммуникационные протоколы, а данные могут быть закодированы способами, оптимальными по соотношению эффективности и переносимости на другие платформы.

Контрольные вопросы и задания

1. Назовите наиболее часто используемые OPC-сервера, применяемые в АСУ ТП.
2. Перечислите режимы чтения данных из OPC DA-серверов.
3. Какие поля имеют данные, передаваемые между OPC DA-сервером и клиентами?
4. Сформулируйте основное назначение OPC HDA-серверов.
5. Какие основные недостатки имеют OPC UA-серверов?

ПРАКТИЧЕСКИЕ ЗАДАНИЯ

Задание 1. Первичная обработка сигналов

Цель работы: разработка программных компонентов, обеспечивающих фильтрацию сигналов; проведение сравнительного анализа сигналов, полученных после их обработки.

Задачи:

1) формирование зашумленного сигнала;

2) разработка функциональных блоков, реализующих фильтрацию сигналов алгоритмами экспоненциального сглаживания, скользящего среднего и медианного фильтра;

3) изучение функционального блока RAMP – ограничителя скорости изменения сигнала;

4) разработка мнемосхемы, демонстрирующей работу фильтров;

5) сравнение сигналов после их фильтрации.

1. *Формирование зашумленного сигнала.* В качестве генератора сигналов может быть использован функциональный блок GEN (библиотека Util). Требуется сгенерировать сигнал, содержащий полезный сигнал, высокочастотный шум и всплески.

2. *Разработка функциональных блоков.* Каждый из функциональных блоков должен одержать два входа и как минимум один выход. Входными сигналами для каждого из блоков выступают зашумленный сигнал и настроечный параметр фильтра, а выходным – обработанный фильтром сигнал.

2.1. Фильтр скользящего среднего. Расчет значения сигнала осуществляется по формуле

$$Y[n] = \frac{\sum_{i=n+1-N}^{n} X[i]}{N},$$

где $X[i]$ – зашумленный сигнал на i-м такте работы; $Y[n]$ – отфильтрованный сигнал на n-м такте; N – настроечный параметр фильтра.

2.2. Фильтр экспоненциального сглаживания. Вычисление отфильтрованного значения осуществляется по формуле

$$Y[n] = \gamma X[n] + (1-\gamma) Y[n-1],$$

где $X[n]$ – зашумленный сигнал на n-м такте работы; $Y[n]$ – отфильтрованный сигнал на n-м такте; γ – настроечный параметр фильтра.

2.3. Медианный фильтр. Вычисление отфильтрованного значения сигнала осуществляется по формуле

$$Y[n] = med\left(sort\left(X\left[(n+1-M)\ldots n\right]\right)\right),$$

где $X\left[(n+1-M)\ldots n\right]$ – массив значений зашумленного сигнала; $sort\left(X\left[(n+1-M)\ldots n\right]\right)$ – отсортированный массив значений зашумленного сигнала; $med\left(sort\left(X\left[(n+1-M)\ldots n\right]\right)\right)$ – значение зашумленного сигнала, находящееся на центральной позиции в отсортированном массиве (если количество значений в массиве четное, то полусумма средних); $Y[n]$ – отфильтрованный сигнал на n-м такте; M – настроечный параметр фильтра.

3. *Изучение функционального блока RAMP.* Необходимо из библиотеки Util вызвать функциональный блок RAMP и подключить к нему зашумленный сигнал. Исходя из характера зашумленного сигнала, настроить пороговые значения, ограничивающие скорость нарастания и спада сигнала, а также базовое значение времени.

4. *Разработка мнемосхемы, демонстрирующей работу фильтров.* Средствами визуализации программного пакета CoDeSys представить в одних осях координат попарно отфильтрованные сигналы с зашумленным. Также с целью настройки фильтров необходимо вывести на мнемосхему настроечные параметры с возможностью редактирования их значений в режиме исполнения кода (онлайн).

5. *Сравнение сигналов после их фильтрации.* Выбрать критерии и произвести сравнительный анализ результатов фильтрации сигналов в соответствии с выбранными критериями.

Задание 2. Дифференцирование сигналов

Цель работы: разработка программных компонентов, обеспечивающих дифференцирование сигналов; проведение сравнительного анализа сигналов, полученных после их дифференцирования.

Задачи:

1) формирование сигнала;
2) разработка функциональных блоков, реализующих дифференцирование сигналов алгоритмами: одностороннего дифференцирования, двухстороннего дифференцирования;
3) изучение функционального блока DERIVATIVE – блока дифференцирования сигналов;
4) разработка мнемосхемы, демонстрирующей результаты работы;
5) сравнение сигналов после их дифференцирования.

1. *Формирование сигнала.* В качестве генератора сигналов может быть использован функциональный блок GEN (библиотека Util). Требуется сгенерировать рабочий сигнал, подлежащий дифференцированию.

2. *Разработка функциональных блоков.* Каждый из функциональных блоков должен содержать интерфейсные входы и выход, на котором формируется сигнал после дифференцирования.

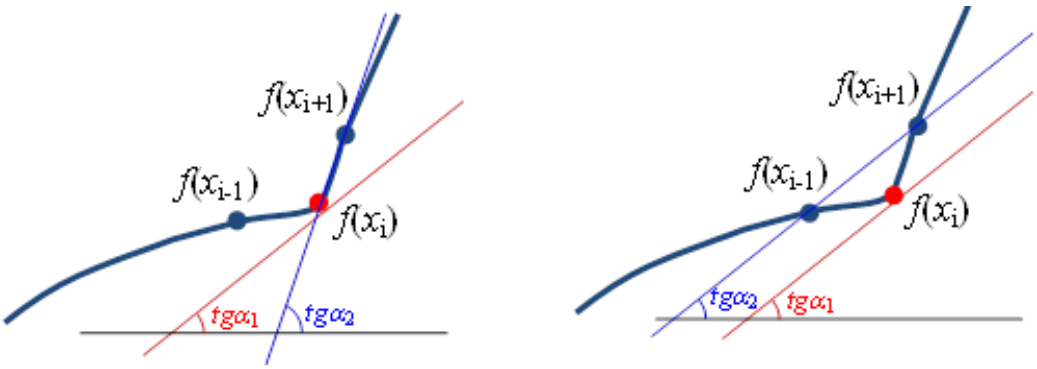
2.1. Одностороннее дифференцирование. Расчет значения сигналов осуществляется по формулам:

$$f'(x) = \frac{f(x+\Delta x) - f(x)}{\Delta x};$$

$$f'(x) = \frac{f(x) - f(x-\Delta x)}{\Delta x}.$$

2.2. Двухстороннее дифференцирование. Вычисление значения сигнала производной осуществляется по формуле

$$f'(x) = \frac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x}.$$



а *б*

Рис. 65. Графическая интерпретация дифференцирования сигналов: а – одностороннее дифференцирование; б – двухстороннее дифференцирование

3. *Изучение функционального блока DERIVATIVE.* Необходимо из библиотеки Util вызвать функциональный блок DERIVATIVE и подключить к нему сгенерированный сигнал. Изучить интерфейс функционального блока.

4. *Разработка мнемосхемы, демонстрирующей результаты работы.* Средствами визуализации программного пакета CoDeSys представить графическую интерпретацию полученных результатов работы. При необходимости вынести на мнемосхему настроечные параметры.

5. *Сравнение сигналов после дифференцирования.* Сравнить полученные результаты. Сделать выводы о полученных результатах.

Задание 3. Интегрирование сигналов

Цель работы: разработка программных компонентов, обеспечивающих интегрирование сигналов; проведение сравнительного анализа сигналов, полученных после их интегрирования.

Задачи:

1) формирование сигнала;
2) разработка функциональных блоков, реализующих интегрирование сигналов алгоритмами: левых прямоугольников, трапеций, парабол;
3) изучение функционального блока INTEGRAL – блока интегрирования сигналов;
4) разработка мнемосхемы, демонстрирующей результаты работы;
5) сравнение сигналов после их интегрирования.

1. *Формирование сигнала.* В качестве генератора сигналов может быть использован функциональный блок GEN (библиотека Util). Требуется сгенерировать рабочий сигнал, подлежащий интегрированию.

2. *Разработка функциональных блоков.* Каждый из функциональных блоков должен содержать интерфейсные входы и выход, на котором формируется сигнал после интегрирования.

2.1. Интегрирование методом левых прямоугольников. Расчет значения интегрированного сигнала осуществляется по формуле

$$\int_a^b f(x)dx \approx h\sum_{i=0}^{n-1} f_i = h\left(f_0 + f_1 + \cdots + f_{n-1}\right).$$

2.2. Интегрирование методом трапеций. Вычисление значения интегрированного сигнала осуществляется по формуле

$$\int_a^b f(x)dx \approx h\sum_{i=1}^{n} \frac{f_i - f_{i-1}}{2}.$$

2.3. Интегрирование методом парабол (Симпсона). Вычисление значения интегрированного сигнала осуществляется по формуле

$$\int_a^b f(x)dx \approx \int_a^b p_2(x)dx = \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right).$$

3. *Изучение функционального блока INTEGRAL.* Необходимо из библиотеки Util вызвать функциональный блок INTEGRAL и подключить к нему сгенерированный сигнал. Изучить интерфейс функционального блока.

4. *Разработка мнемосхемы, демонстрирующей результаты работы.* Средствами визуализации программного пакета CoDeSys представить графическую интерпретацию полученных результатов работы. При необходимости вынести на мнемосхему настроечные параметры.

5. *Сравнение сигналов после интегрирования.* Сравнить полученные результаты. Сделать выводы о полученных результатах.

Задание 4. Разработка функционального блока ПИД-регулятора

Цель работы: разработка программного компонента, обеспечивающего выработку управляющего воздействия в соответствии с алгоритмом ПИД-регулятора; проведение сравнительного результатов работы разработанного функционального блока и стандартного библиотечного функционального блока PID.

Задачи:

1) формирование сигнала;
2) разработка функционального блока, реализующего выработку управляющего воздействия согласно алгоритму ПИД-регулирования с зависимыми настройками;
3) изучение функционального ПИД-блока, вырабатывающего управляющее воздействие согласно законам ПИД-регулирования;
4) разработка мнемосхемы, демонстрирующей результаты работы;
5) сравнение результатов регулирования.

1. *Формирование сигнала.* В качестве генератора сигналов может быть использован функциональный блок GEN (библиотека Util). Требуется сгенерировать рабочий сигнал, подлежащий регулированию.

2. *Разработка функционального блока ПИД-регулирования.* При разработке функционального блока ПИД-регулятора предусмотреть следующие интерфейсные входы:

- актуальное значение регулируемой величины;
- задающее воздействие;
- настроечные параметры регулятора;
- режим работы регулятора;
- ручное задание выходной величины;
- начальное значение;
- масштаб выходного сигнала блока.

В качестве составляющих регулятора: интегральная и дифференциальная составляющие обоснованно выбрать алгоритмы численного интегрирования и дифференцирования.

3. *Изучение функционального блока PID.* Необходимо из библиотеки Util вызвать функциональный блок PID и подключить к нему сгенерированный сигнал. Изучить интерфейс функционального блока.

4. *Разработка мнемосхемы, демонстрирующей результаты работы.* Средствами визуализации программного пакета CoDeSys представить графическую интерпретацию полученных результатов работы. Вынести на мнемосхему настроечные параметры регуляторов, а также режим работы регуляторов.

5. *Сравнение результатов регулирования.* Сравнить полученные результаты. Сделать выводы о полученных результатах.

Задание 5. Регулирование уровня жидкости в емкости

Цель работы: управление уровнем жидкости в резервуаре.

Задачи:

1) разработка модели объекта управления, содержащего резервуар, трубопровод на входе, задвижка на сливе резервуара;
2) разработка блока ШИМ для управления задвижкой;
3) настройка параметров регулятора;
4) разработка мнемосхемы, демонстрирующей управление уровнем жидкости в резервуаре.

1. *Разработка модели системы управления.* На рис. 66 представлена модель объекта управления. В резервуар подается жидкость с входным потоком F_1. При неуправляемом изменяющемся входном потоке F_1 обеспечить стабилизацию уровня L в резервуаре путем управления потоком F_2 через степень открытия задвижки a. Расчет текущего уровня жидкости осуществляется по формуле

$$L = \frac{\int (F_1 - F_2)dt}{S},$$

где S – площадь поперечного сечения резервуара.

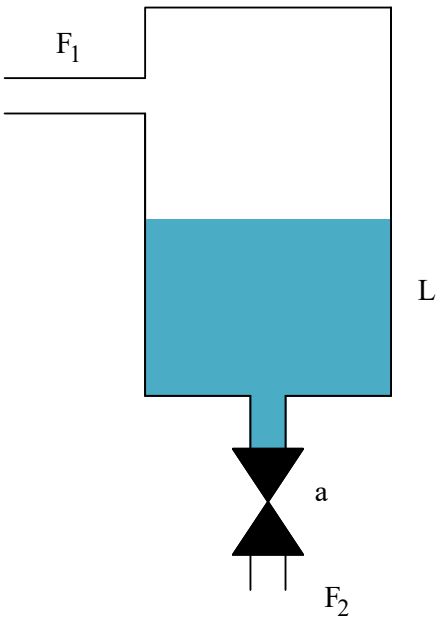


Рис. 66. Объект управления

Геометрические размеры резервуара предусмотреть самостоятельно. Диаметр трубопровода на выходе резервуара в 1,5 раза больше, чем на входе.

Для выработки управляющих воздействий на исполнительный механизм задвижки необходимо использовать ПИД-регулятор. В свою очередь, исполнительный механизм задвижки не имеет аналогового интерфейса управления. Управление осуществляется через блок ШИМ.

2. *Разработка блока ШИМ.* Используя аналоговый выходной сигнал ПИД-регулятора, получить дискретные управляющие сигналы для управления исполнительным механизмом задвижки через ШИМ (рис. 67).

3. *Настройка параметров регулятора.* Обеспечить настройку параметров регулятора, гарантирующих перерегулирование переходного

процесса уровня в резервуаре жидкости не более 10 %. Для настройки параметров регулятора может быть использован любой известный алгоритм, а также предложен собственный.

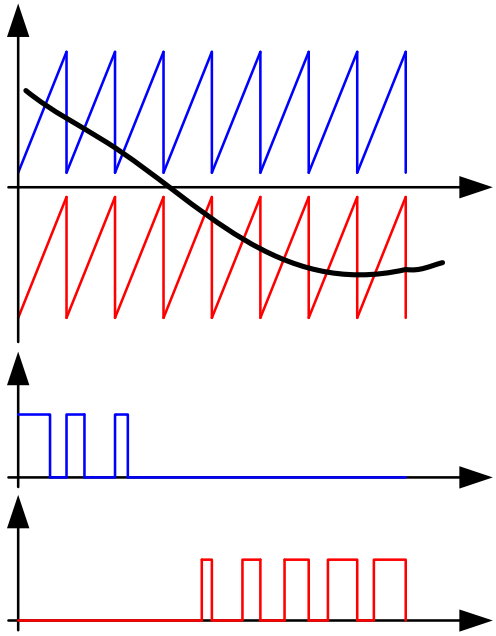


Рис. 67. Широтно-импульсная модуляция

4. *Разработка мнемосхемы.* На мнемосхеме представить визуализацию технологического процесса стабилизации уровня. Предусмотреть возможность изменения с мнемосхемы:

- входного потока жидкости в резервуар;
- режима работы регулятора;
- настроечных параметров регулятора;
- задания уровня жидкости в резервуаре.

Уровень воды в резервуаре должен динамически изменяться, а задвижка динамически изменять цвет исходя из ее состояния: крайние положения (открыто и закрыто), промежуточное положение, рядом с задвижкой должна быть цифровая интерпретация положения задвижки.

Представить тренд, на котором показаны задающее воздействие и текущий уровень жидкости в резервуаре.

Задание 6. Разработка интерфейса системы управления в MasterSCADA

Целью работы является разработка мнемосхемы, отражающей технологический процесс поддержания заданного уровня жидкости в резервуаре сливным дискретным клапаном при постоянной подаче жидкости в резервуар (рис. 68).

Описание требований к интерфейсу

Задание требуемого уровня осуществляется с мнемосхемы. Текущий уровень жидкости в резервуаре и состояние клапана должны динамически изменяться: заливка резервуара и цвет клапана (зеленый – открыт, красный – закрыт).

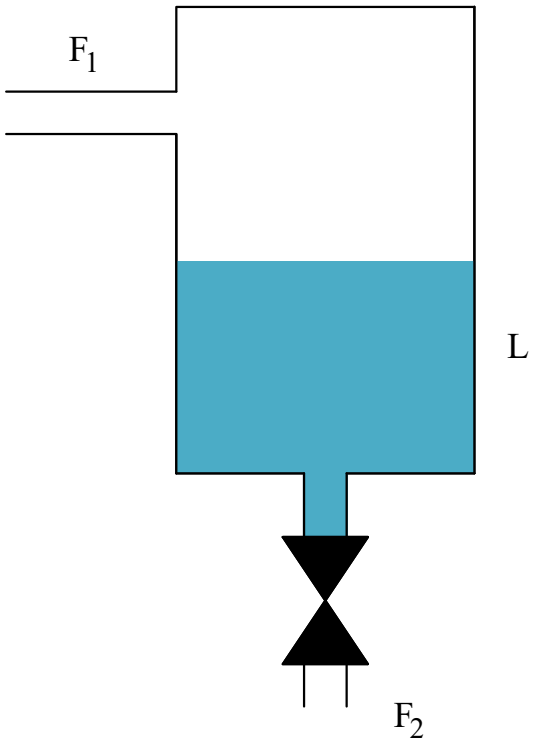


Рис. 68. Объект управления

На мнемосхеме представить мнемонические индикаторы состояния уровня жидкости в резервуаре – НИЖЕ НОРМЫ, НОРМА, ВЫШЕ НОРМЫ. В случае состояния ВЫШЕ НОРМЫ клапан на сливе должен быть открыт, а задание уровня сброшено на значение 0.

Также на мнемосхеме требуется представить график изменения текущего уровня в резервуаре и заданное значение уровня.

Стандартными средствами разработать тренд для параметров – уровень жидкости в резервуаре, состояние сливного клапана.

Разработать элементы управления SCADA-системой: печать мнемосхемы, останов системы, смена оператора, вызов тренда, вызов журнала событий, переход в навигатор проекта.

В разрабатываемой SCADA-системе создать не менее двух учетных записей с разными правами доступа к системе.

На текущей мнемосхеме отразить рабочую информацию: текущую дату и время, учетную запись активного оператора.

Задание 7. Регулирование уровня жидкости в емкости

Цель работы: организация передачи данных между программным обеспечением среднего и верхнего уровней АСУ ТП*.

Задачи:

1) разработка модели объекта управления, содержащего резервуар, трубопровод на входе, задвижку на сливе резервуара;

2) разработка блока ШИМ для управления задвижкой;

3) настройка параметров регулятора;

4) разработка мнемосхемы SCADA-системы, демонстрирующей управление уровнем жидкости в резервуаре.

1. *Разработка модели системы управления.* На рис. 69 представлена модель объекта управления. В резервуар подается жидкость с входным потоком F_1. При неуправляемом изменяющемся входном потоке F_1 обеспечить стабилизацию уровня L в резервуаре путем управления потоком F_2 через степень открытия задвижки a. Расчет текущего уровня жидкости осуществляется по формуле

$$L = \frac{\int (F_1 - F_2)dt}{S},$$

где S – площадь поперечного сечения резервуара. Геометрические размеры резервуара предусмотреть самостоятельно. Диаметр трубопровода на выходе резервуара в 1,5 раза больше, чем на входе.

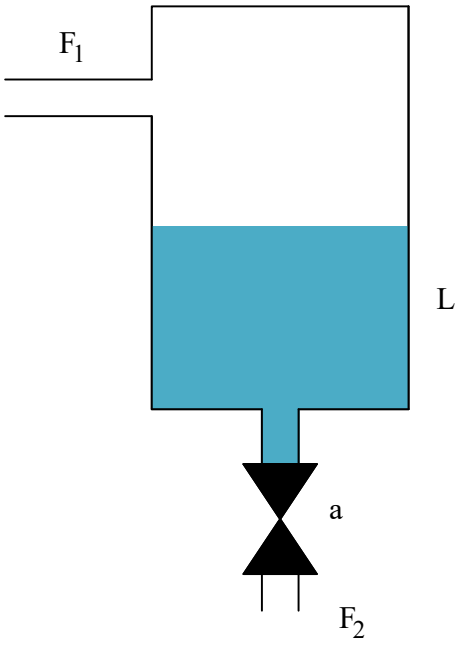


Рис. 69. Объект управления

Для выработки управляющих воздействий на исполнительный механизм задвижки необходимо использовать ПИД-регулятор. В свою очередь, исполнительный механизм задвижки не имеет аналогового интерфейса управления. Управление осуществляется через блок ШИМ.

2. *Разработка блока ШИМ.* Используя аналоговый выходной сигнал ПИД-регулятора получить дискретные управляющие сигналы для управления исполнительным механизмом задвижки через ШИМ (рис. 70).

* В качестве программного обеспечения контроллера и SCADA-системы могут использоваться Codesys и Masterscada, а для передачи данных – Codesys OPC.

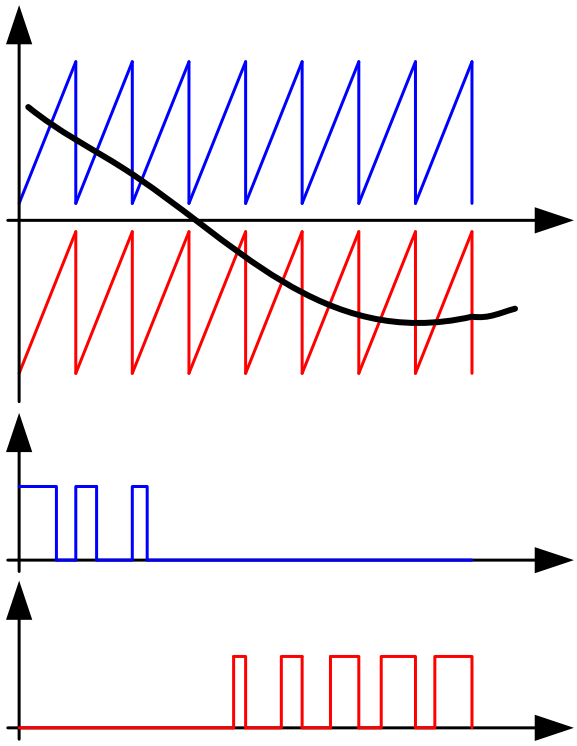


Рис. 70. Широтно-импульсная модуляция

3. *Настройка параметров регулятора.* Обеспечить настройку параметров регулятора, гарантирующих перерегулирование переходного процесса уровня в резервуаре жидкости не более 10 %. Для настройки параметров регулятора может быть использован любой известный алгоритм, а также предложен собственный.

4. *Разработка мнемосхемы.* Задание требуемого уровня осуществляется с мнемосхемы и передается в пакет программирования контроллера. Значения текущего уровня жидкости в резервуаре и положение задвижки должны считываться в SCADA-системе и динамически изменять цвета: заливка резервуара и цвет фона задвижки (зеленый – полностью открыта, красный – полностью закрыта, серый – промежуточное положение, более 0 % и менее 100 %).

Настроечные параметры ПИД-регулятора, доступные для редактирования, размещаются на мнемосхеме и передаются в пакет программирования контроллера.

На мнемосхеме исходя из значений текущего уровня жидкости в резервуаре представить мнемонические индикаторы состояния уровня жидкости в резервуаре – НИЖЕ НОРМЫ (менее 10 %), ВЫШЕ НОРМЫ (более 85 %). В случае состояния ВЫШЕ НОРМЫ задвижка на сливе должна быть закрыта, а задание уровня сброшено на значение 0.

Также на мнемосхеме требуется представить график изменения текущего уровня в резервуаре и заданное значение уровня.

Стандартными средствами разработать тренд для параметров – уровень жидкости в резервуаре, состояние задвижке на сливе резервуара.

Разработать элементы управления SCADA-системой: печать мнемосхемы, останов системы, смена оператора, вызов тренда, вызов журнала событий, переход в навигатор проекта.

В разрабатываемой SCADA-системе создать не менее двух учетных записей с разными правами доступа к системе.

На текущей мнемосхеме отразить рабочую информацию: текущую дату и время, учетную запись активного оператора.

ЗАКЛЮЧЕНИЕ

Таким образом, представленный в пособии материал формирует знания читателя в области разработки АСУ ТП.

В пособии рассмотрены такие вопросы в области построения АСУ ТП, как классификация и их виды, предъявляемые к ним требования согласно стандартам, разновидности их возможной архитектуры построения, ПЛК, предназначенные для автоматизации технологических процессов, ПО для ПЛК и рабочих мест операторов, управляющих различными процессами.

Пособие сопровождено контрольными вопросами и практическими заданиями.

Ознакомление с пособием и выполнение практических заданий формируют базовые компетенции по разработке алгоритмического и программного обеспечения в разрезе решения задач автоматизации технологических процессов и производств, а также навыки владения средствами разработки ПО, использования различных способов построения ПО АСУ ТП, импортирования и экспортирования данных между ПО ПЛК и SCADA различных производителей.

СПИСОК ПРИНЯТЫХ СОКРАЩЕНИЙ

АСУ – автоматизированная система управления;
ПЛК – программируемый логический контроллер;
ПО – программное обеспечение;
ОС – операторская станция;
АСУ ТП – автоматизированная система управления технологическим процессом;
АСУ П – автоматизированная система управления производством;
ИАСУ – интегрированная автоматизированная система управления;
УВК – управляющий вычислительный комплекс;
АЦП – аналогово-цифровой преобразователь;
ЦАП – цифро-аналоговый преобразователь;
МЭК – международная электротехническая комиссия;
ООП – объектно ориентированное программирование;
ЧМИ – человеко-машинный интерфейс;
IL – instruction list;
LD – ladder diagram;
SFC – sequential function chart;
ST – structured text;
FBD – functional block diagram;
HMI – human-machinery interface;
MMI – man-machinery interface;
SCADA – supervisory control and data acquisition;
OPC – OLE (Objects Linked and Embedded) for Process Control;
OPC DA – OLE for Process Control Data Access;
OPC HDA – OLE for Process Control Historical Data Access.

СПИСОК ЛИТЕРАТУРЫ

1. Максимычев О.И. Программирование логических контроллеров (PLC) : учебное пособие / О.И. Максимычев, А.В. Либенко, В.А. Виноградов. – Москва : МАДИ, 2016. – 188 с.
2. Сергеев А.И. Программирование контроллеров систем автоматизации : учебное пособие / А.И. Сергеев, А.М. Черноусова, А.С. Русяев. – Оренбург : ОГУ, 2016. – 125 с.
3. Русакова Е.А. Системы сбора информации : учебное пособие / Е.А. Русакова. – Екатеринбург : УрГУПС, 2016. – 259 с.
4. Efimov S.V. Computer-aided system software : study aid / S.V. Efimov. – Tomsk : TPU Publishing House, 2014. – 92 p.
5. Efimov S.V. Computer-aided system software. Laboratory operation manual : study aid / S.V. Efimov. – Tomsk : TPU Publishing House, 2014. – 76 p.
6. Зюбин В.Е. Программирование ПЛК : языки МЭК 61131-3 и возможные альтернативы / В.Е. Зюбин // Промышленные АСУ и контроллеры. – № 11. – 2005. – С. 31–35.
7. Денисенко В.В. Компьютерное управление технологическим процессом, экспериментом, оборудованием / В.В. Денисенко. – Москва : Горячая линия – Телеком, 2009. – 608 с.
8. Петров И.В. Программируемые контроллеры. Стандартные языки и инструменты / И.В. Петров ; под ред. В.П. Дьяконова. – Москва : СОЛОН-Пресс, 2003. – 256 с.
9. Руководство пользователя по программированию ПЛК в CoDeSys 2.3. – Режим доступа: https://ftp.owen.ru/CoDeSys23/06_Documentation/Cds23_Manual_v2.8.pdf.
10. Стандартные языки программирования контроллеров // Системы управления. – Режим доступа: http ://texproc.ru/index.php/ispiu/95-sjpispip?start=2.
11. Елисеева А.А. Анализ методов настройки ПИД-регулятора / А.А. Елисеева, А.М. Малышенко // Молодежь и современные информационные технологии : сборник трудов VII Всероссийской научно-практической конференции студентов, аспирантов и молодых ученых. – Томск, 2009. – С. 30–31.
12. Ефимов С.В. Анализ и синтез стационарных и интервальных систем управления на основе зависимости расположения их полюсов и нулей от прямых показателей качества : автореф. дис. … канд. техн. наук / С.В. Ефимов. – Томск, 2011. – 18 с. Режим доступа: http://www.lib.tpu.ru/fulltext/a/2011/16.pdf.
13. Минаев И.Г. Программируемые логические контроллеры. Практическое руководство для начинающего инженера / И.Г. Минаев, В.В. Самойленко. – Ставрополь : АГРУС, 2009. – 100 с.

14. Парр Э. Программируемые контроллеры: руководство для инженера : пер. с англ. / Э. Парр. – Москва : БИНОМ : Лаборатория знаний, 2007. – 516 с.

15. Стандартные языки программирования контроллеров // Системы управления. – Режим доступа: https://studme.org/176968/tehnika/standartnye_yazykiprogrammirovaniya_logicheskih_kontrollerov.

16. Белов А.В. Конструирование устройств на микроконтроллерах / А.В. Белов. – Санкт-Петербург : Наука и техника, 2005. – 256 с.

17. Болл С. Аналоговые интерфейсы микроконтроллеров / С. Болл. – Москва : Додэка-XXI, 2007. – 360 с.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 3
1. КЛАССИФИКАЦИЯ АСУ 4
1.1. Информационные системы 4
1.2. Управляющие системы 4
Контрольные вопросы и задания 5
2. ВИДЫ АСУ 6
2.1. АСУ ТП 6
2.2. АСУ П 7
2.3. ИАСУ 7
Контрольные вопросы и задания 8
3. АРХИТЕКТУРА АСУ 9
3.1. Требования к архитектуре 9
3.2. Разновидности архитектур 10
3.3. Информационные потоки 11
Контрольные вопросы и задания 13
4. ФУНКЦИИ АСУ 14
Контрольные вопросы и задания 15
5. ОСНОВНЫЕ ФУНКЦИИ ПЛК И ОС 16
Контрольные вопросы и задания 16
6. СОСТАВ АСУ 17
Контрольные вопросы и задания 17
7. СТАНДАРТ МЭК 61131 18
7.1. История развития стандарта 18
7.2. Характеристики стандарта МЭК 61131-3 19
7.3. Языки стандарта 19
7.4. Требования к языкам стандарта 20
7.5. Значимость стандарта 20
7.6. Тенденции развития стандарта 21
Контрольные вопросы и задания 22
8. СТРУКТУРНЫЕ МОДЕЛИ 23
8.1. Модель программного обеспечения 23
8.2. Модель взаимодействия 24
8.3. Модель программирования 25
Контрольные вопросы и задания 27
9. ИНСТРУМЕНТЫ ПРОЕКТИРОВАНИЯ ПО ПЛК 28
9.1. Инструменты комплексов программирования ПЛК 29
9.1.1. Встроенный редактор 29
9.1.2. Текстовый редактор 30
9.1.3. Графический редактор 31
9.2. Средства отладки проекта 32

9.3. Средства управления проектом 34
9.4. Средства восстановления проекта 34
9.5. Средства обеспечения безопасности 34
Контрольные вопросы и задания 35
10. СТРУКТУРА ПО ПЛК 36
10.1. Задачи 36
10.2. Ресурсы 38
10.3. Конфигурация 39
Контрольные вопросы и задания 39
11. КОМПОНЕНТЫ ОРГАНИЗАЦИИ ПРОГРАММ 40
11.1. Определение компонента 40
11.2. Формальные и актуальные параметры 41
11.3. Параметры и переменные компонента 41
11.4. Функции 42
11.5. Функциональные блоки 43
11.6. Программы 44
Контрольные вопросы и задания 44
12. ЯЗЫКИ СТАНДАРТА МЭК 61131-3 45
12.1. Язык линейных инструкций (Instruction list, IL) 45
12.1.1. Аккумулятор 45
12.1.2. Переход на метку 46
12.1.3. Скобки 46
12.1.4. Модификаторы 47
12.1.5. Операторы 47
12.1.6. Вызов функциональных блоков и программ 48
12.1.7. Вызов функции 49
12.1.8. Комментирование 49
12.2. Структурированный текст (Structured Text, ST) 49
12.2.1. Выражения 49
12.2.2. Порядок вычислений выражений 50
12.2.3. Пустое выражение 50
12.2.4. Оператор IF 50
12.2.5. Оператор множественного выбора CASE 51
12.2.6. Циклы While и Repeat 52
12.2.7. Цикл FOR 53
12.2.8. Прерывание итераций операторами EXIT и RETURN 53
12.3. Релейные диаграммы LD 54
12.3.1. Реле с фиксацией 55
12.3.2. Порядок выполнения и обратные связи 55
12.3.3. Расширение возможностей LD 56
12.4. Функциональные диаграммы блоков (FBD) 57
12.4.1. Соединительные линии 57
12.4.2. Порядок выполнения FBD 57

12.4.3. Инверсия логических сигналов 57
12.4.4. Соединители и обратные связи 58
12.4.5. Метки, переходы и возврат 58
12.5. Последовательные функциональные схемы (SFC) 59
12.5.1. Шаги 59
12.5.2. Переходы 59
12.5.3. Начальный шаг 60
12.5.4. Параллельные ветви 60
12.5.6. Альтернативные ветви 61
Контрольные вопросы и задания 62
13. СТАНДАРТНЫЕ КОМПОНЕНТЫ 63
13.1. Арифметические операторы 63
13.2. Операторы битового сдвига 64
13.3. Логические битовые операторы 65
13.4. Операторы выбора и ограничения 65
13.5. Операторы сравнения 66
13.6. Математические функции 67
13.7. Строковые функции 69
13.8. Таймеры 70
13.8.1. Таймер-пульс TP 70
13.8.2. Таймер с задержкой выключения TOF 71
13.8.3. Таймер с задержкой включения TON 72
13.8.4. Часы реального времени RTC 73
13.9. Триггеры 73
13.9.1. Триггер с доминантой включения SR 74
13.9.2. Триггер с доминантой выключения RS 75
13.10. Детектор импульсов 76
13.10.1. Детектор переднего фронта R_TRIG 76
13.10.2. Детектор заднего фронта F_TRIG 77
13.11. Счетчики 77
13.11.1. Инкрементный счетчик CTU 78
13.11.2. Декрементный счетчик CTD 79
13.11.3. Инкрементно-декрементный счетчик CTUD 79
13.12. Побитовый доступ к целым 80
13.12.1. Чтение бита 80
13.12.2. Запись бита 81
13.12.3. Упаковка в байт 81
13.12.4. Распаковка байта 82
13.13. HYSTERESIS 84
13.14. Пороговый сигнализатор 85
13.15. Ограничение скорости изменения сигнала 87
13.16. Интерполяция зависимостей 88
13.17. Дифференцирование 90

13.18. Интегрирование 91
13.19. ПИД-регулятор 92
13.19.1. Описание алгоритма регулирования 92
13.19.2. Настройка ПИД-регулятора 95
Контрольные вопросы и задания 96
14. ИНТЕРФЕЙС СИСТЕМЫ УПРАВЛЕНИЯ 97
14.1. Функции SCADA 97
14.2. Человеко-машинный интерфейс 99
14.3. Диспетчерское управление 100
14.4. Функции системы автоматического управления 100
14.5. Хранение истории процесса 101
14.6. Безопасность SCADA 101
14.7. Свойства SCADA 101
14.7.1. Инструментальные свойства 102
14.7.2. Эксплуатационные свойства 102
14.7.3. Открытость SCADA-систем 102
14.7.4. Рентабельность SCADA-систем 103
Контрольные вопросы и задания 103
15. OPC-СЕРВЕР 104
15.1. Обзор стандарта OPC 104
15.2. OPC DA-сервер 105
15.3. OPC HDA-сервер 106
15.4. OPC UA-сервер 107
Контрольные вопросы и задания 108
ПРАКТИЧЕСКИЕ ЗАДАНИЯ 109
Задание 1. Первичная обработка сигналов 109
Задание 2. Дифференцирование сигналов 110
Задание 3. Интегрирование сигналов 112
Задание 4. Разработка функционального блока ПИД-регулятора 113
Задание 5. Регулирование уровня жидкости в емкости 114
Задание 6. Разработка интерфейса системы управления в MasterSCADA 115
Задание 7. Регулирование уровня жидкости в емкости 117
ЗАКЛЮЧЕНИЕ 120
СПИСОК ПРИНЯТЫХ СОКРАЩЕНИЙ 121
СПИСОК ЛИТЕРАТУРЫ 122

Учебное издание

ЕФИМОВ Семён Викторович
ПУШКАРЕВ Максим Иванович
ФАДЕЕВ Александр Сергеевич

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ АВТОМАТИЗИРОВАННЫХ СИСТЕМ УПРАВЛЕНИЯ ТЕХНОЛОГИЧЕСКИМИ ПРОЦЕССАМИ

Учебное пособие

Корректура *Е.Л. Тен*
Компьютерная верстка *К.С. Чечельницкая*
Дизайн обложки *А.И. Сидоренко*

Подписано к печати 14.01.2020. Формат 60×84/16. Бумага «Снегурочка».
Печать CANON. Усл. печ. л. 7,44. Уч.-изд. л. 6,73.
Заказ 02-20. Тираж 100 экз.

Издательство
ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ