Лабораторная работа №1

Программа управления движением манипулятора станка с числовым программным управлением по заданному закону в среде CoDeSys

Цель работы: изучение основ программирования в среде CoDeSys для практической реализации задачи программирования движения манипулятора станка с числовым программным управлением (ЧПУ) по заданному закону.

1. Теоретическая часть

CoDeSys – это современный инструмент для программирования контроллеров CoDeSys предоставляет программисту удобную среду для программирования контроллеров на языках стандарта МЭК 61131-3. Используемые редакторы и отладочные средства базируются на широко известных и хорошо себя зарекомендовавших принципах, знакомых по другим популярным средам профессионального программирования (такие, как Visual C++).

CODESYS (акроним англ. *Controller Development System*) — инструментальный программный комплекс промышленной автоматизации. Производится и распространяется компанией 3S-Smart Software Solutions GmbH (Кемптен, Германия). Версия 1.0 была выпущена в 1994 году. С ноября 2012 изменено написание на CODESYS.

Среда программирования

Основой комплекса CODESYS является среда разработки прикладных программ для программируемых логических контроллеров (ПЛК). Она распространяется бесплатно и может быть без ограничений установлена на нескольких рабочих местах.

В CODESYS для программирования доступны все пять определяемых стандартом IEC 61131-3 (МЭК 61131-3) языков:

- **IL** (Instruction List) — ассемблер-подобный язык
- **ST** (Structured Text) — Pascal-подобный язык
- **LD** (Ladder Diagram) — язык релейных схем
- **FBD** (Function Block Diagram) — язык функциональных блоков
- **SFC** (Sequential Function Chart) — язык диаграмм состояний

В дополнение к FBD поддержан язык программирования **CFC** (Continuous Function Chart) с произвольным размещением блоков и расстановкой порядка их выполнения.

В CODESYS реализован ряд других расширений спецификации стандарта IEC 61131-3. Самым существенным из них является поддержка объектно-ориентированного программирования (ООП).

Встроенные компиляторы CODESYS генерируют машинный код (двоичный код), который загружается в контроллер. Поддерживаются основные 16- и 32-разрядные процессоры: Infineon C166, TriCore, 80x86, ARM (архитектура), PowerPC, SH, MIPS (архитектура), Analog Devices Blackfin, TI C2000/28x и другие.

При подключении к контроллеру среда программирования переходит в режим отладки. В нем доступен мониторинг/изменение/фиксация значений переменных, точки останова, контроль потока выполнения, горячее обновление кода, графическая трассировка в реальном времени и другие отладочные инструменты.

CODESYS версии V3 построен на базе так называемой платформы автоматизации: **CODESYS Automation Platform**. Она позволяет изготовителям оборудования развивать комплекс путём подключения собственных плагинов.

Расширенная профессиональная версия среды разработки носит название **CODESYS Professional Developer Edition**. Она включает поддержу UML-диаграмм классов и состояний,

подключение системы контроля версий Subversion, статический анализатор и профилировщик кода. Распространяется по лицензии.

Инструмент **CODESYS Application Composer** позволяет перейти от программирования практических приложений к их быстрому составлению. Пользователь составляет собственную базу объектов, соответствующих определенным приборам, механическим узлам машины и т. п. Каждый объект включает программную реализацию и визуальное представление. Законченное приложение составляется из необходимых объектов, конфигурируется и автоматически генерируется программа на языках МЭК 61131-3.

Система исполнения

Для программирования контроллера в среде CODESYS в него должна быть встроена система исполнения (**Control Runtime System**). Она устанавливается в контроллер в процессе его изготовления. Существует специальный инструмент (**Software development kit**), позволяющий адаптировать её к различным аппаратным и программным платформам.

Программа CODESYS обладает следующими **преимуществами**:

- комплекс можно применять с относительно недорогими моделями контроллеров. Специализированные встроенные компиляторы машинного кода и гибко адаптируемая система исполнения позволяют "выжать" максимум из ограниченных аппаратных ресурсов;

- CoDeSys полноценно поддерживает все пять стандартных языков программирования. CoDeSys непосредственно способен генерировать машинный код для большинства широко распространенных процессоров. CoDeSys объединяет мощь высококлассных инструментов программирования для языков высокого уровня, таких как С или Паскаль с простотой работы и практической функциональностью ПЛК систем программирования;

- CoDeSys обладает рядом особенностей, выделяющих его среди конкурирующих систем: *быстрое внедрение; эффективные средства ввода; высокая производительность*.

Быстрое внедрение

CoDeSys имеет готовые решения для большинства широко распространенных платформ. Простота настройки не отражается на быстродействии прикладных проектов, компилятор и система исполнения тщательно отработаны. Все это позволяет подготовить контроллеры к выходу на рынок в минимальные сроки.

Эффективные средства ввода

Функции автоматического объявления и форматирования, адаптивный ассистент ввода максимально упрощают работу. Все команды имеют возможность управления мышью и быстрого ввода с клавиатуры. Это делает работу программиста комфортной и эффективной.

Высокая производительность

Встроенный компилятор непосредственно генерирует быстрый машинный код. Это обеспечивает максимально высокую производительность прикладных проектов. Современные интеллектуальные технологии, включая "инкрементальный компилятор", позволяют обрабатывать проекты, содержащие тысячи переменных и сотни программных компонентов очень быстро. CoDeSys обеспечивает разработчика набором высокоэффективных инструментальных средств, включая полноценную эмуляцию ПЛК, отладку по шагам, точки останова, визуализацию объекта управления, трассировку значений переменных, "горячую" корректировку кода.

Основным **назначением** программного комплекса CODESYS является программирование ПЛК и промышленных компьютеров в стандарте МЭК 61131-3. Ряд неординарных решений 3S привел к тому, что CoDeSys стал штатным инструментом программирования ПЛК более 100 ведущих европейских изготовителей: ABB, Beckhoff, Beck IPC, Berger Lahr, Bosch Rexroth, ifm, Keb, Kontron, Lenze, Moeller, WAGO, Fastwel и др. Некоторые из них используют CoDeSys как базовое ядро собственных систем программирования, известных под собственными торговыми марками.

2. Практическая часть

Задача:

Контроль оператором движения некоторого механизма. Оператор должен периодически подтверждать правильность функционирования механизма. В противном случае, необходимо выдать предупреждение, а затем остановить работу.

Рабочий орган нашей машины совершает циклическое движение по периметру прямоугольника.

Запуск CoDeSys

CoDeSys запускается точно также как большинство Windows приложений: **Пуск -> Программы -> 3S Software -> CoDeSys V2.3 -> CoDeSys V2.3**

Создание новой программы

Создайте новый проект командой **File -> New**.

Настройка целевой платформы (Target Settings)

Проект является машинно-независимым, его можно опробовать в режиме эмуляции. Однако рекомендуется для определенности конкретный контроллер. На страничке диалогового окна '**Configuration**' установите CoDeSys SP for Windows NT Realtime и подтвердите ввод - **Ok**.

Главная программа PLCPRG POU

Следующее диалоговое окно определяет тип первого программного компонента (**New POU**). Выберете язык реализации (**language of the POU**) **FBD** и сохраните предложенные по умолчанию тип компонента - программа (**Type of the POU Program**) и имя - **Name PLCPRG**. PLC_PRG это особый программный компонент (POU). В однозадачных проектах он циклически вызывается системой исполнения.

Объявление переключателя подтверждения

Переключатель подтверждения - это переменная, которая будет изменять значение при подтверждении корректности работы механизма оператором.

В первой цепи графического FBD редактора выделите строку вопросов **???** и введите наименование первой переменной. Пусть это будет **Observer** (наблюдатель). Теперь нажмите на клавиатуре стрелку вправо. В появившемся диалоге определения переменной сохраните наименование (**Name Observer**) и логический тип (**Type BOOL**). Измените класс переменной (**Class**) на глобальный (**VAR_GLOBAL**). Подтвердите определение - **OK**. Теперь определение переменной **Observer** должно появиться в окне глобальных переменных проекта (**Global_Variables**):

```
VARGLOBAL
Observer: BOOL;
ENDVAR
```

Детектор переднего фронта

Оператор должен подтверждать работу именно переключением клавиши, а не просто спать с постоянно нажатой клавишей подтверждения. Чтобы разделить эти ситуации необходимо определить моменты нажатия и отпускания, т.е. переходы значения логической переменной их нуля (FALSE) в единицу (TRUE) и наоборот.

Вернитесь в окно редактора PLC_PRG и выделите позицию справа от переменной **Observer**. Вы должны увидеть маленький пунктирный прямоугольник. Щелкните по нему правой клавишей мыши. В контекстном меню ввода задайте команду **Box**.

По умолчанию, вставляется элемент **AND**. Воспользуемся ассистентом ввода: нажмите клавишу **F2**. В диалоговом окне (слева) выберете категорию: стандартные функциональные блоки (**Standard Function Blocks**). Из триггеров (trigger) стандартной библиотеки (standard.lib) выберете **R_TRIG**. R_TRIG формирует логическую единицу по переднему фронту на входе.

Необходимо задать имя для нового экземпляра функционального блока R_TRIG.

Щелкните мышкой над изображением триггера и введите имя **Trigl**. В диалоге определения переменных должен быть указан класс **Class VAR** (локальные переменные), имя (**Name**) **Trigl** и тип (**Type R_TRIG**). Нажмите **OK**.

Детектор заднего фронта

Выделите вход функционального блока **Trig**l и вставьте (как было описано выше) элемент **AND** и переименуйте его в **OR** (логическое ИЛИ). Выделите свободный вход **OR** функционального и вставьте перед ним экземпляр функционального блока **F_TRIG** под именем **Trig2**. На вход **F_TRIG**, с помощью ассистента ввода (F2) подайте (категория **Global Variables**) переменную **Observer**.

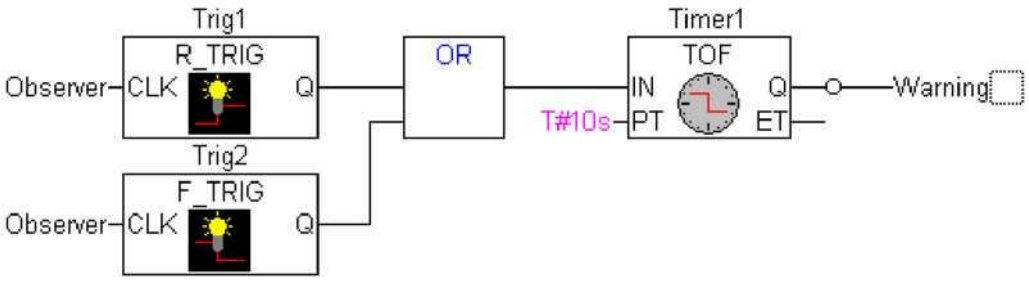
Контроль времени, первая часть

Вставьте после **OR** экземпляр функционального блока **TOF** (таймер с задержкой выключения) под именем **Timerl**. Замените три знака вопроса на входе **PT** константой **T#10s**. Она соответствует 10 секундам. Это время можно менять, в процессе отладки.

Выход «Предупреждение»

Выделите выход Q таймера Timerl и в контекстном меню (правая клавиша мыши) дайте команду Assign (присвоить). Замените знаки вопроса (???) на имя переменной **Warning**. В диалоге определения задайте класс **Class VARGLOBAL** и тип BOOL.

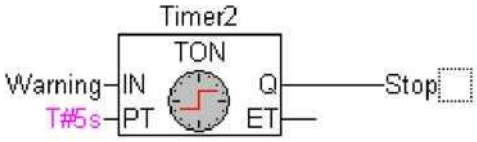
Теперь выделите позицию в середине линии соединяющей выход таймера и переменную **Warning**. Задайте команду **Negate** в контекстном меню. Маленький кружек означает инверсию значения логического сигнала.



Стоп-сигнал по второму интервалу времени

Создайте новую цепь командой меню **Insert->Network (after)**. Вставьте из стандартной библиотеки в новую цепь элемент (**Box**) типа **TON** (таймер с задержкой включения) под именем **Timer2**. Подайте переменную **Warning** на вход **IN** (используйте ассистент ввода <F2>) и константу **T#5s** на вход **PT**. Выход экземпляра функционального блока Timer2 присвойте (опять **Assign**) новой глобальной (Class VAR_GLOBAL) логической

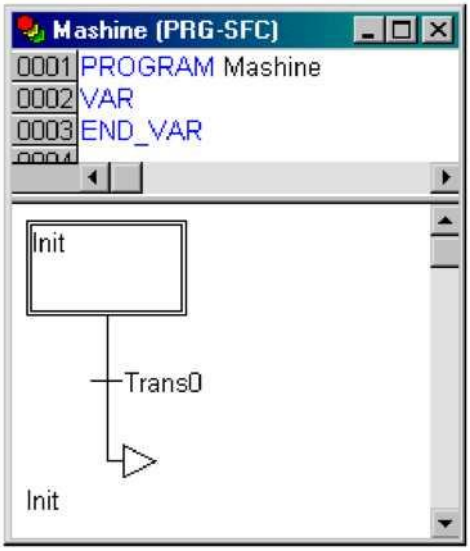
переменной **Stop**.



POU управления механизмом

В левой части окна CoDeSys расположен организатор объектов POUs (в нем присутствует PLC_PRG). Вставьте командой **Add object** в контекстном меню новый программный компонент с именем **Machine**, типом **Type program** и определите для него язык SFC (**Language SFC**).

По умолчанию, создается пустая диаграмма, содержащая начальный шаг "Init" и соответствующий переход "TransO" заканчивающийся возвратом к Init.



*(!) Далее используется упрощенный SFC, без МЭК действий. Если справа от Init виден прямоугольник с действием (Action), снимите в контекстном меню флаг **Use IEC-Steps** и переопределите заново POU Machine.*

Определяем последовательность работы механизма

Каждой фазе работы должен соответствовать определенный этап (шаг). Выделите переход (TransO) так, чтобы он оказался окружен пунктирной рамкой. В контекстном меню дайте команду вставки шага и перехода под выделенным: **Step-Transition (after)**. Аналогично повторите вставку еще 4 раза. Включая Init, должно получиться 6 шагов с переходами.

Щелкая мышью по именам переходов и шагов, можно заметить, что они выделяются цветом. Таким способом можно определить новые наименования.

Первый после **Init** шаг должен назваться **Go_Right**. Под ним **Go_Down**, **Go_Left**, **Go_Up** и **Count**.

Программирование первого шага

Щелкните дважды на шаге **Go_Right**. CoDeSys начнет определение действия шага и попросит выбрать язык его реализации (**Language**). Выберете **ST** (structured text) и перейдите в автоматически открытое окно текстового редактора. В этом шаге рабочий орган нашего механизма должен перемещаться по оси X вправо. Программа должна выглядеть так:

```
Xpos:=Xpos + 1;
```

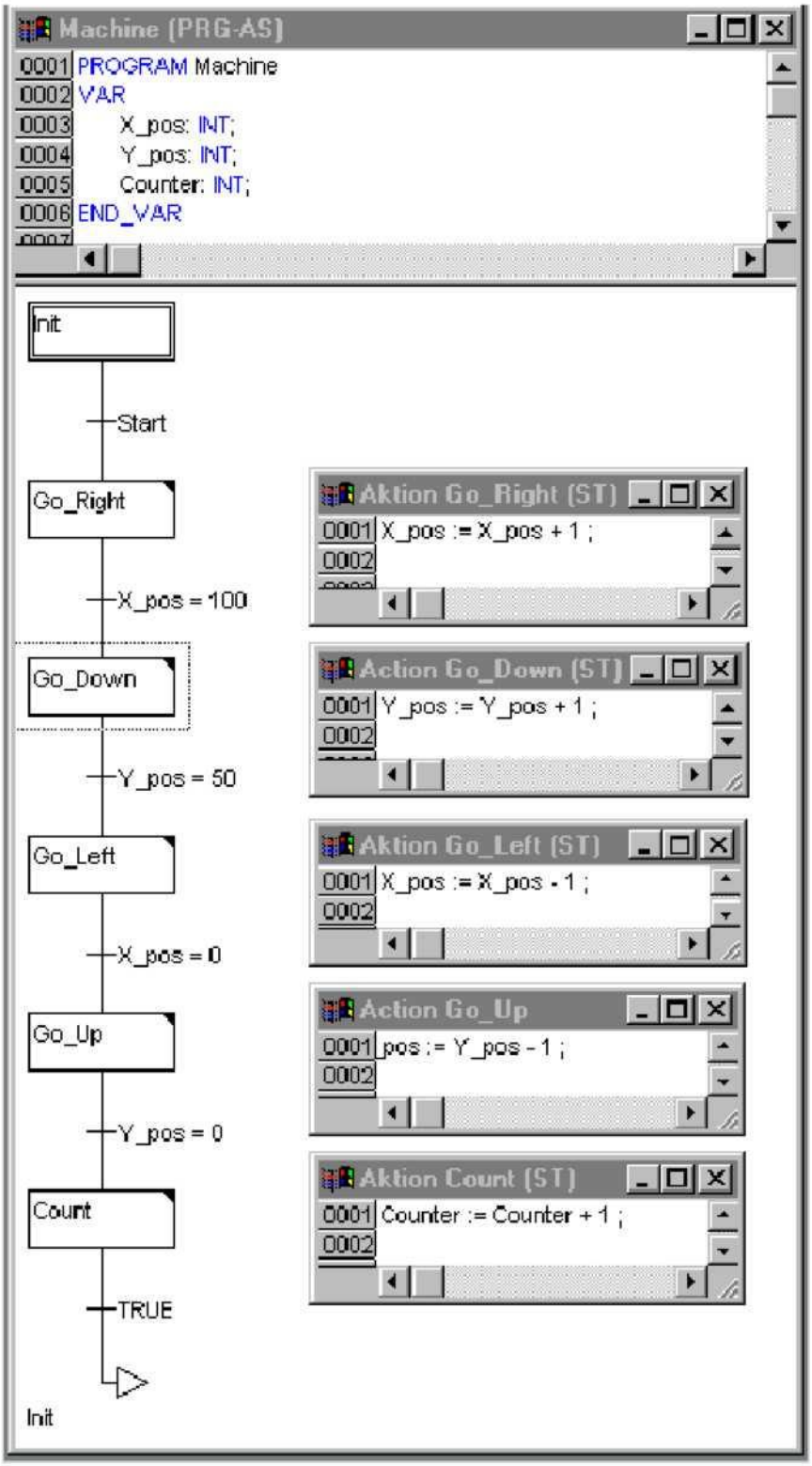
Завершите ввод клавишей Return, и определите переменную **X_pos** типа **INT** (целое).

Теперь верхний уголок шага должен быть закрашен. Это признак того что действие этого шага определено.

Программируем следующие шаги

Повторите описанную последовательность для всех оставшихся шагов. Переменные **Y_pos** и **Counter** должны быть типа **INT**.

Шаг **Go_Down** программа **Y_pos := Y_pos + 1 ;**
Шаг **Go_Left** программа **X_pos := X_pos - 1 ;**
Шаг **Go_Up** программа **Y_pos := Y_pos - 1 ;**
Шаг **Count** программа **Counter := Counter + 1 ;**



Определяем переходы

Переход должен содержат условие, разрешающее переключение на следующий шаг. Переход после шага Init назовите **Start** и определите новую логическую переменную (**Class VAR_GLOBAL** тип **Type BOOL**). При единичном значении этой переменной начинается цикл работы механизма.

Следующий переход должен содержать условие **X_Pos = 100**, так при значении позиции X включается следующая фаза движения.

Условие третьего шага **Y_pos = 50**, четвертого **X_pos = 0**, пятого **Y_pos = 0** и шестого **TRUE** (переход разрешен сразу же, после однократного выполнения)

Останов механизма

Вернитесь к **PLC_PRG** POU и добавьте третью цепь.

Вместо вопросов вставьте переменную **Stop**, и затем из контекстного меню вставьте оператор **Return**. Return прерывает работу программы PLC_PRG POU при единичном значении Stop.

Вызов POU управления механизмом

Добавьте еще одну цепь, выделите ее и вставьте элемент **Box** из контекстного меню. Как обычно это будет "AND". Нажмите <F2> и в ассистенте ввода задайте POU управления механизмом в категории пользовательских программ (**User defined Programs category**).

Компиляция проекта

Откомпилируйте проект целиком командой меню **Project->Rebuild all**, либо клавишей **<F11>**.

Если вы все сделали верно, то в нижней части окна должно появиться сообщение: **„0 errors”**. В противном случае необходимо исправить допущенные ошибки. В это помогут развернутые сообщения об ошибках.

Визуализация

Создание визуализации

Третья страничка организатора объектов CoDeSys называется визуализация (Visualization). Перейдите на страничку визуализации. В контекстном меню введите команду добавления объекта **Add object**. Присвойте новому объекту имя **Observation**.

В конце работы, окно визуализации будет выглядеть так:

На панели инструментов выберете прямоугольник (**Rectangle**). В окне редактора визуализации нажмите левую клавишу мыши и растяните прямоугольник до нужной высоты и ширины, отпустите клавишу.

Настройка первого элемента визуализации

Диалоговое окно настройки элемента вызывается двойным щелчком мыши на его изображении. Задайте в окошке содержимое (**Contents**) категории текст (**Text Category**) слово **ОК**.

Теперь перейдите в категорию переменных (**Variables Category**), щелкните мышью в поле изменение цвета (**Change Color**) и воспользуйтесь ассистентом ввода **<F2>**. Вставьте переменную **.Observer** из списка глобальных переменных. Далее перейдите в категорию цвета (**Colors**). Задайте цвет закраски элемента (**Inside**), например, светло-голубой. Для «возбужденного» состояния необходимо определить другой цвет (**Alarm color**), например, голубой. В категории ввода (**Input Category**) необходимо еще раз ввести переменную Observer и поставить флажок **Toggle variable**. Закройте диалог настройки.

В итоге, прямоугольник будет отображаться светло-голубым при значении переменной Observer равном FALSE и голубым, при значении TRUE. Ее значение будет изменяться при каждом «нажатии» нашей клавиши.

Развитие визуализации

Нарисуйте окружность **«Внимание»**. В настройках, **Text Category**, **Contents** задайте текст *Внимание.*

Colors Category, **Color** закраска **Inside** серым цветом, **Alarm color** красным цветом.

Скопируйте созданную окружность командой **Edit -> Copy** и вставьте ее один раз командой **Edit -> Paste**.

Поправьте настройки новой окружности **Стоп**:

Text Category, Contents текст *Стоп.*

Variable Category, Color change переменная **.Stop**

Нарисуйте прямоугольник для клавиши **Пуск**, имеющей следующие настройки:

Text Category, Contents текст **Пуск**

Variable Category, Color change переменная **.Start**

Input Category, флажок **Toggle variable** включен, переменная **.Start**

Colors Category, Color закраска **Inside** красным, и **Alarm color** зеленым.

Нарисуйте прямоугольник для счетчика со следующими настройками:

Text Category, Contents текст **Счетчик: %s** (%s заместитель для отображения значения переменной)

Variable Category, Textdisplay переменная **Machine.Counter**

Нарисуйте небольшой прямоугольник, обозначающий рабочий инструмент механизма, со следующими настройками:

Absolute movement Category, X-Offset переменная **Machine.Xpos**

Absolute movement Category, Y-Offset переменная **Machine.Ypos**

Colors Category, Color закраска **Inside** голубым цветом.

Если хотите, нарисуйте две декоративных рамки для разделения областей контроля и механизма. Задайте в них соответствующие надписи с выравниванием по низу (**Vertical** alignment **bottom**). Используя контекстное меню, поместите декоративные прямоугольники на задний план (**Send to back**).

(!) Последующие два пункта есть смысл разбирать, только если у Вас имеется контроллер с CoDeSys. Рассмотрим подключение на примере CoDeSys SP RTE. В ином случае можете проверить работу примера в режиме эмуляции. Для этого переходите к пункту «Запуск проекта».

Запуск целевой системы

Запустите систему исполнения (обратите внимание, что **CoDeSys SP RTE** работает

только в Windows NT 4.0, Windows 2000 или Windows XP). Теперь в панели задач вы увидите иконку CoDeSys SP RTE. Щелкните по ней правой клавишей и дайте команду на старт системы (**Start System**).

Настройка канала и соединение

Если вы в первый раз подключаете контроллер к CoDeSys необходимо выполнить определенные настройки.

В меню **Online** откройте диалог **Communication parameters**. Нажмите клавишу **New** для настройки нового соединения. Желательно присвоить ему некоторое осмысленное имя.

В простейшем случае CoDeSys SP RTE работает на той же машине (компьютере) что и среда программирования CoDeSys. Это означает, что можно применить способ соединения посредством разделяемой памяти (**Shared memory (Kernel)**). Если контроллер расположен на другой машине сети, необходимо изменить параметр 'localhost' на имя машины или задать соответствующий IP адрес.

Настройка подтверждается клавишей **OK**.

Запуск проекта

Соединение с контроллером устанавливается командой **Online -> Login** из среды программирования CoDeSys. Если используется удаленное соединение, CoDeSys попросит подтвердить загрузку (download) кода проекта.

Команда запускает **Online -> Run** проект. Перейдите в окно визуализации и проверьте работу механизма.

Для запуска проекта в режиме эмуляции установите флажок в меню **Online -> Simulation**. Далее переходите в режим *online* и запускайте проект, как описано выше.

3. Требования к отчету

Отчет по лабораторной работе предоставляется в электронном виде в формате документа Microsoft Word (.doc или .docx), и должен содержать следующие основные пункты:

3.1 Скриншоты программного кода
3.2 Скриншоты визуализации
3.3 Краткое описание алгоритмов построения программы
3.4 Скриншоты полученных результатов работы программы
3.5 Выводы по лабораторной работе

4. Контрольные вопросы:

4.1 Основное назначение CODESYS
4.2 Преимущества CODESYS
4.3 Основные языки CODESYS
4.4 Назначение и принцип работы элемента **trigger** (R_TRIG) из библиотеки элементов
4.5 Назначение и принцип работы элемента **timer** (TON, TOF) из библиотеки элементов
4.6 Для чего нужна визуализация программы?
4.7 Какой заместитель для отображения переменной следует использовать в элементе **счетчик**?
4.8 Какой синтаксис заместителя следует использовать для задания требуемого времени работы таймера?
4.9 Чем характеризуется переменная типа **BOOL**?
4.10 Как работает элемент **AND**?
4.11 Как работает элемент **OR**?

5. Список используемых источников

5.1. http://www.kipshop.ru/CoDeSys/steps/codesys_v23_ru.pdf

5.2. https://ru.wikipedia.org/wiki/CoDeSys

5.3. https://studfiles.net/preview/1979051/page:2/

5.4. http://dspace.kgsu.ru/xmlui/bitstream/handle/123456789/3587/22_%D0%A1%D0%B1%D1%80%D0%BE%D0%B4%D0%BE%D0%B2-%D0%9D%D0%91_2014_%D0%9C%D0%A3_3.pdf?sequence=1&isAllowed=y

5.5. http://www.codesys.ru/docs/3S_brochure_ru.pdf

Лабораторная работа №2

Программа управления блоком светофора в среде CoDeSys на базе программируемого логического контроллера ОВЕН ПЛК-100

Цель работы: изучение основ программирования в среде CoDeSys для практической реализации задачи программирования движения манипулятора станка с числовым программным управлением (ЧПУ) по заданному закону.

1. Теоретическая часть

Программи́руемый логи́ческий контро́ллер (сокр. *ПЛК*; англ. *programmable logic controller*, сокр. *PLC*; более точный перевод на русский — контроллер с программируемой логикой), **программируемый контроллер** — специальная разновидность электронной вычислительной машины. Чаще всего ПЛК используют для автоматизации технологических процессов. В качестве основного режима работы ПЛК выступает его длительное автономное использование, зачастую в неблагоприятных условиях окружающей среды, без серьёзного обслуживания и практически без вмешательства человека.

Иногда на ПЛК строятся системы числового программного управления станков.

ПЛК — устройства, предназначенные для работы в системах реального времени.

ПЛК имеют ряд **особенностей**, отличающих их от прочих электронных приборов, применяемых в промышленности:

- в отличие от микроконтроллера (однокристального компьютера) — микросхемы, предназначенной для управления электронными устройствами — ПЛК являются самостоятельным устройством, а не отдельной микросхемой.
- в отличие от компьютеров, ориентированных на принятие решений и управление оператором, ПЛК ориентированы на работу с машинами через развитый ввод сигналов датчиков и вывод сигналов на исполнительные механизмы;
- в отличие от встраиваемых систем ПЛК изготавливаются как самостоятельные изделия, отдельные от управляемого при его помощи оборудования.

В системах управления технологическими объектами логические команды, как правило, преобладают над арифметическими операциями над числами с плавающей точкой, что позволяет при сравнительной простоте микроконтроллера (шины шириной 8 или 16 разрядов), получить мощные системы, действующие в режиме реального времени. В современных ПЛК числовые операции в языках их программирования реализуются наравне с логическими. Все языки программирования ПЛК имеют лёгкий доступ к манипулированию битами в машинных словах, в отличие от большинства высокоуровневых языков программирования современных компьютеров.

Первые логические контроллеры появились в виде шкафов с набором соединённых между собой реле и контактов. Эта схема не могла быть изменена после этапа проектирования и поэтому получила название — жёсткая логика. Первым в мире, программируемым логическим контроллером, в 1968 году стал Modicon 084 (1968) (от англ. *modular digital controller*), имевший 4 кБ памяти.

Термин PLC ввел Одо Жозеф Стругер (англ.)русск. (Allen-Bradley) в 1971 году. Он также сыграл ключевую роль в унификации языков программирования ПЛК и принятии стандарта IEC61131-3. Вместе с Ричардом Морли (англ.)русск. (Modicon) их называют 'отцами ПЛК'. Параллельно с термином ПЛК в 1970-е годы широко использовался термин *микропроцессорный командоаппарат*.

В первых ПЛК, пришедших на замену релейным логическим контроллерам, логика работы программировалась схемой соединений LD. Устройство имело тот же принцип работы, но реле и контакты (кроме входных и выходных) были виртуальными, то есть существовали в виде программы, выполняемой микроконтроллером ПЛК. Современные ПЛК являются свободно программируемыми.

Виды ПЛК

- Основные ПЛК,
- Программируемое (интеллектуальные) реле,
- Программные ПЛК на базе IBM PC-совместимых компьютеров (англ. *SoftPLC*),
- ПЛК на базе простейших микропроцессоров (i8088/8086/8051 и т. п.),
- Контроллер ЭСУД (Электронная система управления двигателем).

Устройство ПЛК

Часто ПЛК состоит из следующих частей:

- центральная микросхема (микроконтроллер, или микросхема FPGA), с необходимой обвязкой;
- подсистема часов реального времени;
- энергонезависимая память;
- интерфейсы последовательного ввода-вывода (RS-485, RS-232, Ethernet)
- схемы защиты и преобразования напряжений на входах и выходах ПЛК.

Обычно вход или выход ПЛК нельзя сразу же подключить к соответствующему выходу центральной микросхемы. Эти выходы характеризуются низкими уровнями напряжений, обычно от 3,3 до 5 вольт. Входы и выходы ПЛК обычно должны работать с напряжениями 24 В постоянного либо 220 В переменного тока. Поэтому между выходом ПЛК и выходом микросхемы необходимо предусматривать усилительные и защитные элементы.

Структуры систем управления

- ***Централизованная***: в корзину ПЛК устанавливаются модули ввода-вывода. Датчики и исполнительные устройства подключаются отдельными проводами непосредственно, либо при помощи модулей согласования к входам/выходам сигнальных модулей;
- ***Распределенная***: удалённые от ПЛК датчики и исполнительные устройства связаны с ПЛК посредством каналов связи и, возможно, корзин-расширителей с использованием связей типа «ведущий-ведомый» (англ. Master-Slave).

Интерфейсы ПЛК

- RS-232
- RS-485
- Modbus
- CC-Link
- Profibus
- DeviceNet
- ControlNet
- CAN
- AS-Interface
- Industrial Ethernet

Удаленное управление и мониторинг

- SCADA
- операторские панели
- Веб-интерфейс

Языки программирования ПЛК

Для программирования ПЛК используются стандартизированные языки МЭК (IEC) стандарта IEC61131-3

Языки программирования (графические)

- LD (Ladder Diagram) — Язык релейных схем — самый распространённый язык для PLC
- FBD (Function Block Diagram) — Язык функциональных блоков — 2-й по распространённости язык для PLC
- SFC (Sequential Function Chart) — Язык диаграмм состояний — используется для программирования автоматов

- CFC (Continuous Function Chart) — Не сертифицирован IEC61131-3, дальнейшее развитие FBD

Языки программирования (текстовые)

- IL (Instruction List) — Ассемблеро-подобный язык
- ST (Structured Text) — Паскале-подобный язык
- C-YART — Си-подобный язык (YART Studio)

Структурно в IEC61131-3 среда исполнения представляет собой набор ресурсов (в большинстве случаев это и есть ПЛК, хотя некоторые мощные компьютеры под управлением многозадачных ОС представляют возможность запустить несколько программ типа softPLC и имитировать на одном ЦП несколько ресурсов). Ресурс предоставляет возможность исполнять задачи. Задачи представляют собой набор программ. Задачи могут вызываться циклически, по событию, с максимальной частотой.

Программа — это один из типов программных модулей POU. Модули (POU) могут быть типа программа, функциональный блок и функция. В некоторых случаях для программирования ПЛК используются нестандартные языки, например: Блок-схемы алгоритмов С-ориентированная среда разработки программ для ПЛК. HiGraph 7 — язык управления на основе графа состояний системы.

Инструменты программирования ПЛК на языках МЭК 61131-3 могут быть специализированными для отдельного семейства ПЛК или универсальными, работающими с несколькими (но далеко не всеми) типами контроллеров:

- CoDeSys
- ISaGRAF
- ИСР "КРУГОЛ"
- Beremiz
- KLogic

Программирование ПЛК

- Конфигурируемые: В ПЛК хранится несколько программ, а через клавиатуру ПЛК выбирается нужная версия программы;
- Свободно программируемые: программа загружается в ПЛК через его специальный интерфейс с Персонального компьютера используя специальное ПО производителя, иногда с помощью программатора.

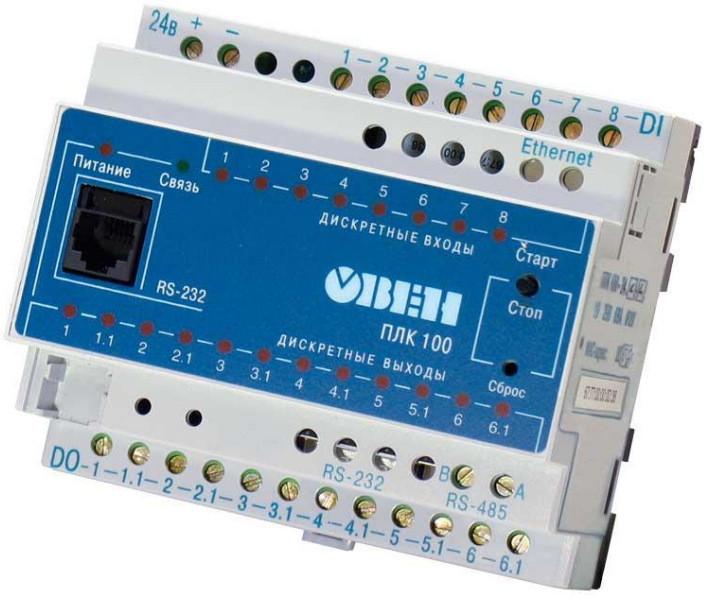
Программирование ПЛК имеет отличие от традиционного программирования. Это связано с тем, что ПЛК исполняют бесконечную последовательность программных циклов, в каждом из которых:

- считывание входных сигналов, в том числе манипуляций, например, на клавиатуре оператором;
- вычисления выходных сигналов и проверка логических условий;
- выдача управляющих сигналов и при необходимости управление индикаторами интерфейса оператора.

Поэтому при программировании ПЛК используются флаги - булевые переменные признаков прохождения алгоритмом программы тех или иных ветвей условных переходов. Отсюда, при программировании ПЛК от программиста требуется определённый навык.

Например, процедуры начальной инициализации системы после сброса или включения питания. Эти процедуры нужно исполнять только однократно. Поэтому вводят булевую переменную (флаг) завершения инициализации, устанавливаемую при завершении инициализации. Программа анализирует этот флаг, и если он установлен, то обходит исполнение кода процедур инициализации.

Программируемый логический контроллер ОВЕН ПЛК 100



Программируемый логический контроллер ОВЕН ПЛК 100 предназначен для:

- создания систем управления малыми и средними объектами
- построения систем диспетчеризации
- построения системы управления и диспетчеризации на базе **ОВЕН ПЛК** возможно как с помощью проводных средств – используя встроенные интерфейсы Ethernet, RS-232, RS-485, так и с помощью беспроводных средств – использую радио, GSM, ADSL модемы

Конструктивные особенности ОВЕН ПЛК 100:

- Контроллер выполнен в компактном DIN-реечном корпусе
- Расширение количества точек ввода\вывода осуществляется путем подключения внешних модулей ввода\вывода по любому из встроенных интерфейсов
- Два варианта питания 220В и 24В постоянного

Вычислительные ресурсы ОВЕН ПЛК 100:

В контроллере изначально заложены мощные вычислительные ресурсы при отсутствии операционной системы:

- высокопроизводительный процессор RISC архитектуры ARM9, с частотой 180МГц компании Atmel;
- большой объем оперативной памяти – 8МБ;
- большой объем постоянной памяти – Flash память, 4МБ;
- объем энергонезависимой памяти, для хранения значений переменных – до 16КБ.

Конкурентные преимущества ОВЕН ПЛК 100:

1. Отсутствие ОС, что повышает надежность работы контроллеров
2. Скорость работы дискретных входов – до 10КГц при использовании подмодулей счетчика

3. Большое количество интерфейсов на борту: Ethernet, 3 последовательных порта, USB Device для программирования контроллера, работающих независимо друг от друга

4. Расширенный температурный диапазон работы: от минус 20 до плюс 70 градусов Цельсия

5. Широкие возможности самодиагностики контроллера

6. Встроенный аккумулятор, позволяющий «пережидать» пропадани питания – выполнять программу при пропадании питания, и переводить выходные элементы в «безопасное состояние»

7. Встроенные часы реального времени

8. Возможность создавать и сохранять архивы на Flash контроллера

9. Возможность работы по любому нестандартному протоколу по любому из портов, что позволяет подключать устройства с нестандартным протоколом (электро-, газо-, водосчетчики, считыватели штрих - кодов и т.д.)

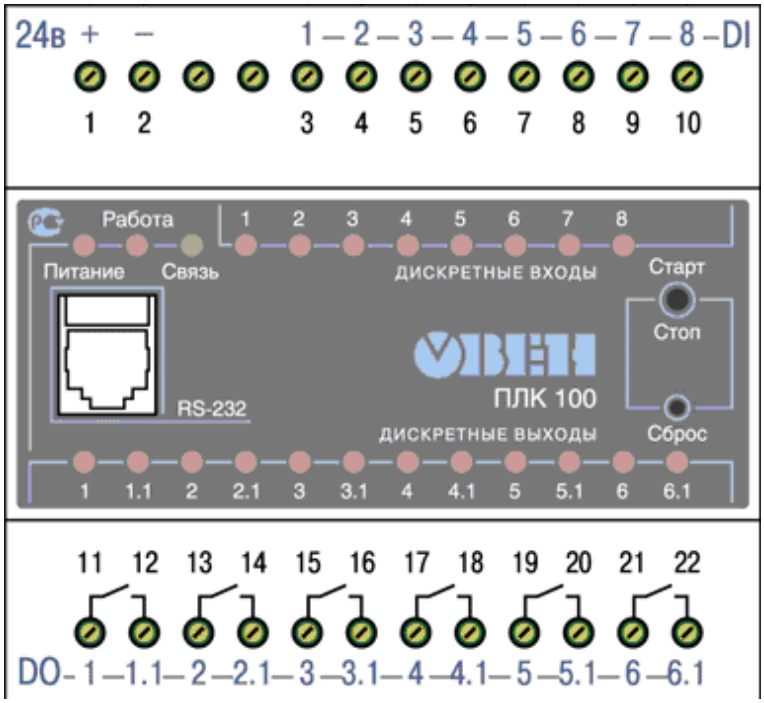
10. Набор готовых программных модулей, предоставляемых бесплатно

Модификация контроллера ОВЕН ПЛК-100

** по заказу*

Внимание!
С выходными элементами типа К выпускаются контроллеры только на 24 В.

Элементы индикации и управления



Светодиодная индикация:
– наличия питания;
– работы программы;
– наличия связи со средой программирования CoDeSys;
– состояния дискретных входов;
– состояния дискретных выходов.

Кнопка «Старт/Стоп» предназначена для запуска и остановки программы в контроллере.

Скрытая кнопка «Сброс» предназначена для перезагрузки контроллера (нажимается тонким заостренным предметом).

Схемы подключения

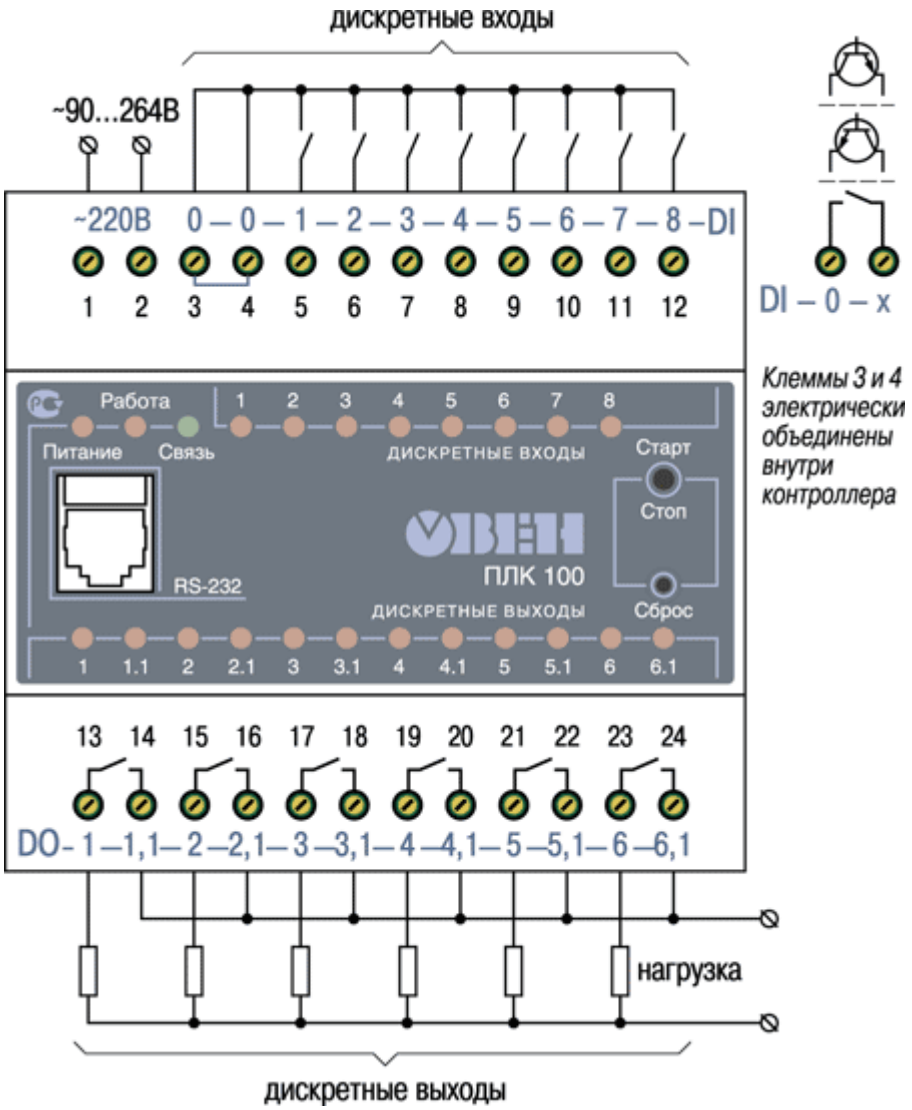
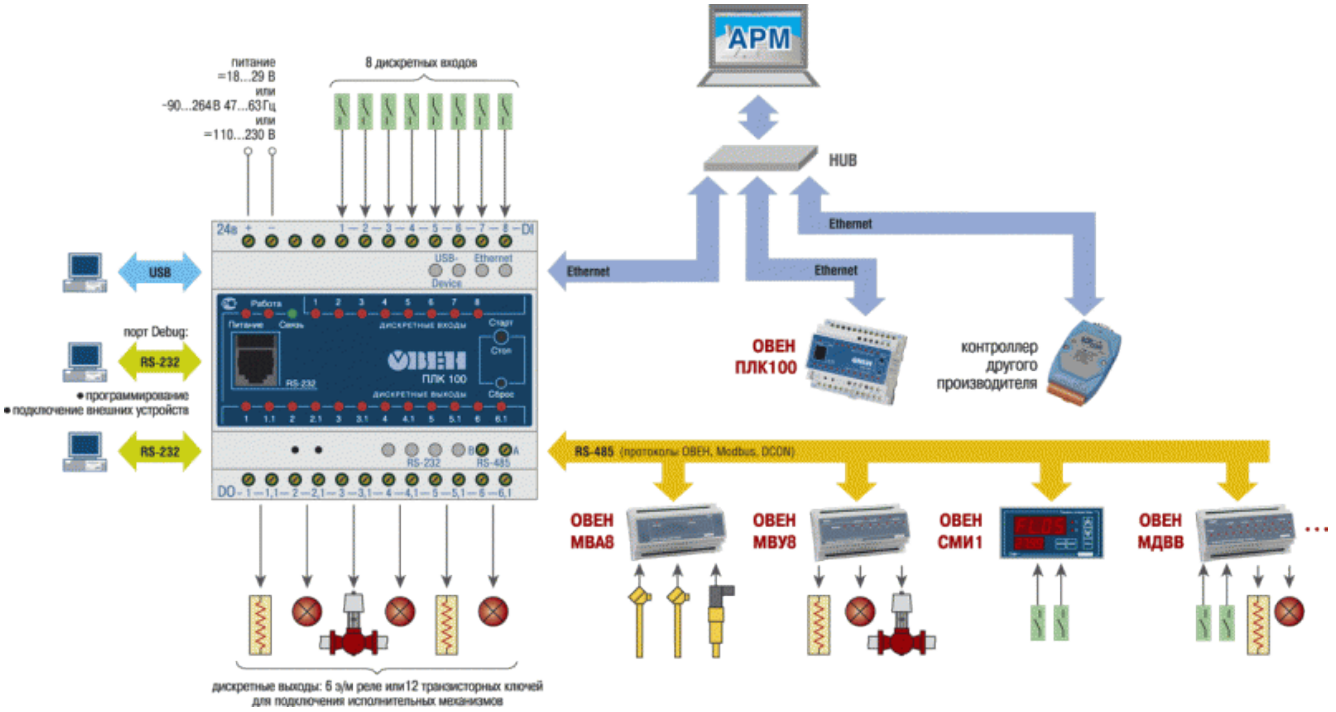


Схема подключения питания, дискретных входов и выходов ПЛК100-220.Р



Возможная схема работы контроллера ОВЕН ПЛК-150 в промышленной сети

Более миллиона контроллеров работают по всему миру, доказывая день за днем свою прочность, надежность и оптимальную приспособляемость к соответствующему заданию. Контроллеры автоматизируют все, что только поддается автоматизации: горные железные дороги, автоматические линии, очистные сооружения, электростанции и производственные установки любой величины и для любой отрасли. Промышленное программное обеспечение PLC разрабатывается с учетом требований международных стандартов IEC 61131-3, требованиям которого отвечают все языки программирования контроллеров. Это облегчает изучение программного обеспечения и позволяет снизить затраты на подготовку персонала.

2. Практическая часть

Блок управления светофором

Рассматриваемая программа – простой блок управления движением на перекрестке, имеющем светофоры для двух пересекающихся направлений движения. Понятно, что светофоры должны иметь два противоположных состояния - красный и зеленый. Чтобы избежать несчастных случаев, добавляются общепринятые переходные стадии: желтый и желто-красный. Последняя стадия должна быть длиннее предыдущей.

На примере данной программы показывается, как управляемые по времени процессы можно представить средствами языков стандарта МЭК 61131-3, как легко можно комбинировать различные языки в CoDeSys и познакомимся со средствами моделирования в CoDeSys.

Создание POU

Запустите CoDeSys и выберите **"File" → "New"**.

В окне диалога определяется первый POU. По умолчанию он получает наименование PLC_PRG. Изменять его не рекомендуется. Тип этого POU – программа. Каждый проект должен иметь программу с таким именем. В качестве языка программирования данного POU выбирается язык Continuous Function Chart (**CFC**).

Создайте еще три объекта. Воспользуйтесь командой **"Project" → "Object Add"** в системном или в контекстном (нажмите правую кнопку мыши в окне Object Organizer) меню. Создайте:

- программу на языке Sequential Function Chart (**SFC**) с именем SEQUENCE,

- функциональный блок на языке Function Block Diagram (**FBD**) с именем TRAFFICSIGNAL

- и еще один аналогичный блок - WAIT, который будет описан на языке Список Инструкции (**IL**).

Назначение блока TRAFFICSIGNAL

В POU TRAFFICSIGNAL сопоставляются определенные стадии процесса соответствующим цветам. То есть программа удостоверяется, что красный свет зажжен в красной стадии и в желто-красной стадии, желтый свет в желтой и желто-красной стадии и т.д.

Назначение блока WAIT

В WAIT создается простой таймер, который на вход получает длину стадии в миллисекундах и на выходе выдает состояние ИСТИНА по истечении заданного периода времени.

Назначение блока SEQUENCE

В SEQUENCE все будет объединено так, чтобы нужные огни зажигались в правильное время и на нужный период времени.

Назначение блока PLC_PRG

В PLC_PRG вводится входной сигнал включения, разрешающий начало работы светофора и 'цветовые команды' каждой лампы связаны с соответствующими выходами аппаратуры.

Блок "TRAFFICSIGNAL"

Вернемся теперь к POU TRAFFICSIGNAL. В редакторе объявлений определите входную переменную (между ключевыми словами VAR_INPUT и END_VAR) по имени STATUS типа INT. STATUS будет иметь четыре возможных состояния, определяющие соответствующие стадии - зеленая, желтая, желто-красная и красная.

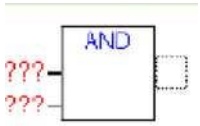
Поскольку наш блок TRAFFICSIGNAL имеет три выхода, нужно определить еще три переменных RED, YELLOW и GREEN. Теперь раздел объявлений блока TRAFFICSIGNAL должен выглядеть так:

Функциональный блок TRAFFICSIGNAL, раздел объявлений:

Программирование блока "TRAFFICSIGNAL"

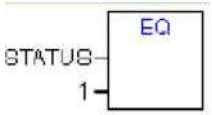
Далее задаются связи входа STATUS с выходными переменными. Для этого перейдите в раздел кода POU (body). Щелкните на поле слева от первой цепи (серая область с номером 1). Вы теперь выбрали первую цепь. Теперь дайте команду меню **"Insert" → "Operator"**.

В первой цепи будет вставлен прямоугольник с оператором AND и двумя входами:



Щелкните мышкой на тексте AND и замените его на EQ.

Три знака вопроса около верхнего из двух входов замените на имя переменной STATUS. Для нижнего входа вместо трех знаков вопроса нужно поставить 1. В результате Вы должны получить следующий элемент:



Щелкните теперь на месте позади прямоугольника EQ. Теперь выбран выход EQ. Выполните команду меню **"Insert" → "Assignment"**.

Измените три вопроса ??? на GREEN. Вы теперь создали цепь следующего вида:

STATUS сравнивается с 1, результат присваивается GREEN. Таким образом, GREEN будет включен, когда STATUS равен 1.

Для других цветов TRAFFICSIGNAL нам понадобятся еще две цепи. Создаете их командой **"Insert" → "Network (after)"**. Законченный POU должен выглядеть следующим образом:

Чтобы вставить оператор перед входом другого оператора, Вы должны выделить сам вход, а не текст (выделяется прямоугольником). Далее используйте команду **"Insert" → "Operator"**.

Первый POU закончен. Как и планировалось, TRAFFICSIGNAL будет управлять включением выходов, руководствуясь значением переменной STATUS.

Подключение библиотеки standard.lib

Для создания таймера в POU WAIT понадобится POU из стандартной библиотеки. Откройте менеджер библиотек командами **"Window" → "Library Manager"**. Выберете **"Insert" → "Additional library"**. Должно открыться диалоговое окно выбора файлов. Выберете **standard.lib** из списка библиотек.

Переходим обратно к POU WAIT. Этот POU будет работать таймером, задающим длительность стадий TRAFFICSIGNAL. POU должен иметь входную переменную TIME типа TIME и генерировать на выходе двоичную (Boolean) переменную OK. Данная переменная должна принимать значение TRUE, когда желательный период времени закончен.

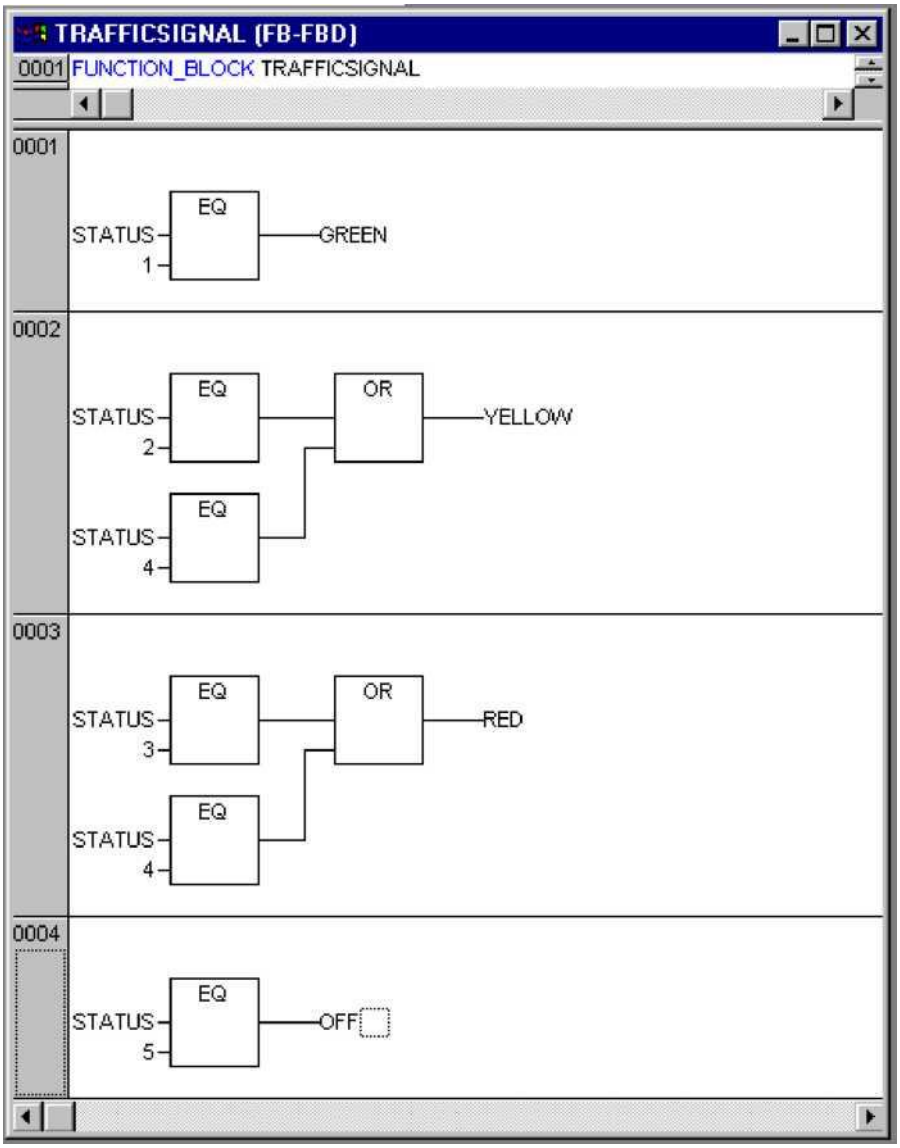
Предварительно устанавливаем эту переменную в FALSE в конце строки объявления (но до точки с запятой) " := FALSE ".

Далее необходимо использовать генератор времени POU TP. Он имеет два входа (IN, PT) и два выхода (Q, ET). TP делает следующее:

- пока IN установлен в FALSE, ET будет 0 и Q будет FALSE. Как только IN переключится в TRUE, выход ET начнет отсчитывать время в миллисекундах. Когда ET достигнет значения заданного PT, счет будет остановлен. Тем временем выход Q равен TRUE, пока ET меньше PT. Как только ET достигнет значения PT, выход Q снова переключится в FALSE.

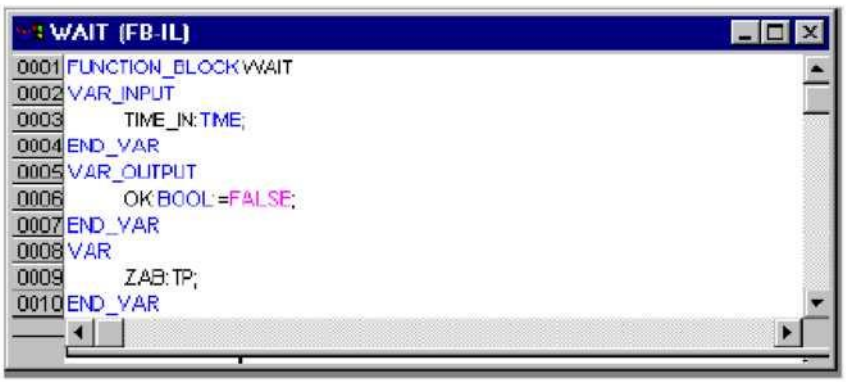
Описание всех POU из стандартной библиотеки приведено в справке программы CODESYS.

Функциональный блок TRAFFICSIGNAL:

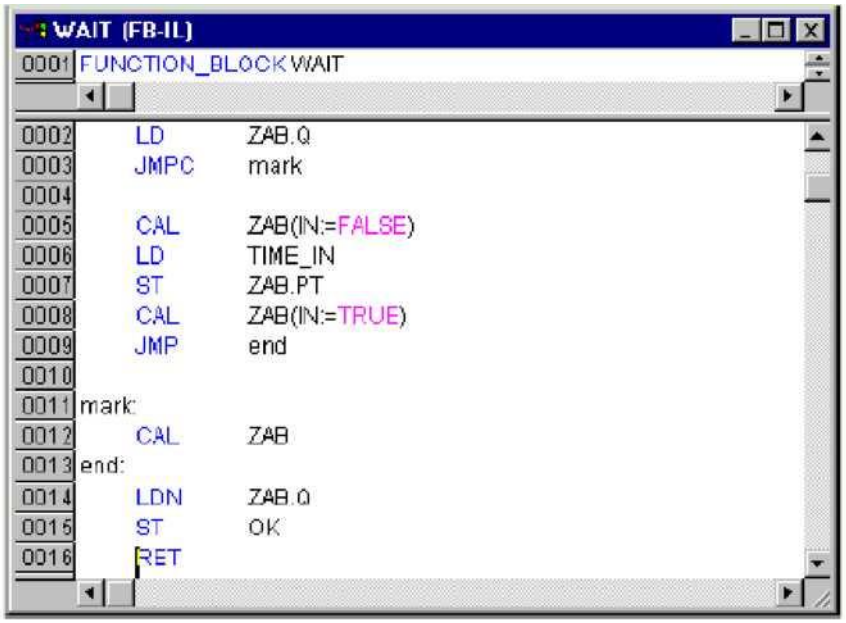


Чтобы использовать TP в POU WAIT, необходимо создать его локальный экземпляр. Для этого объявляется локальная переменную ZAB (отсчитанное время) типа TP (между ключевыми словами VAR, END_VAR).

Раздел объявлений WAIT теперь должен выглядеть так:



Для создания желаемого таймера текст программы должен быть следующим:



Сначала проверяется, установлен ли Q в TRUE (возможно, отсчет уже запущен), в этом случае не трогаем установки ZAB, а вызываем функциональный блок ZAB без входных переменных - чтобы проверить, закончен ли период времени.

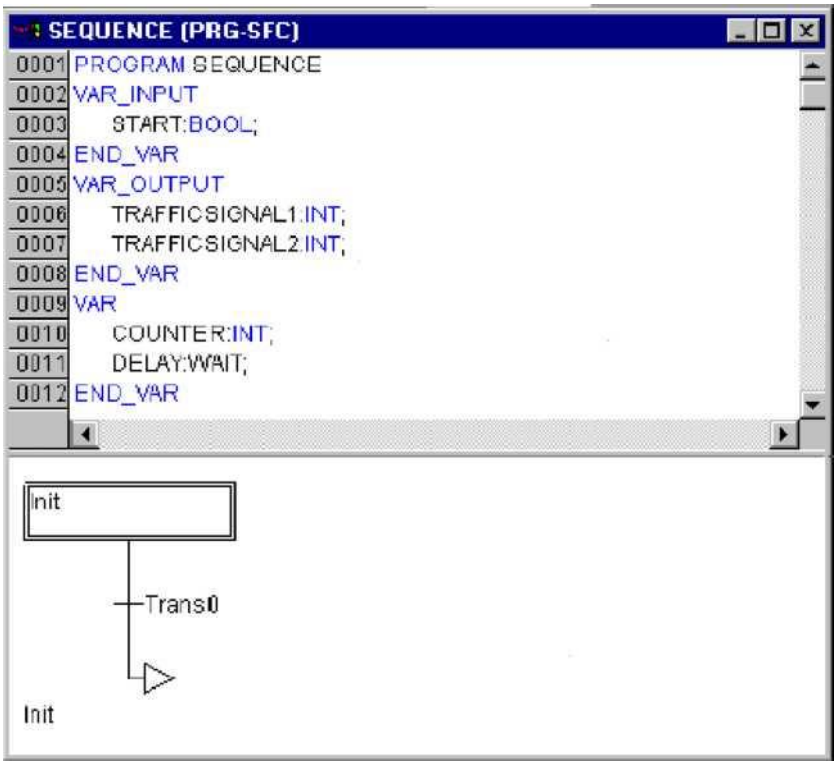
В противном случае устанавливаем переменную IN ZAB в FALSE и одновременно ET в 0 и Q в FALSE. Таким образом, все переменные установлены в начальное состояние. Теперь устанавливаем необходимое время TIME переменной PT и вызываем ZAB с IN:=TRUE. Функциональный блок ZAB теперь будет работать, пока не достигает значения TIME и не установит Q в FALSE.

Инвертированное значение Q будет сохраняться в переменной OK после каждого выполнения WAIT. Как только Q станет FALSE, OK примет значение TRUE.

Работа с таймером закончена. Далее необходимо объединять два блока WAIT и TRAFFICSIGNAL в главной программе PLC_PRG.

БЛОК "SEQUENCE" (ПЕРВАЯ ВЕРСИЯ)

Сначала объявляются все необходимые переменные. Это входная переменная START типа BOOL, две выходных переменные TRAFFICSIGNAL1 и TRAFFICSIGNAL2 типа INT и одна DELAY типа WAIT. Программа SEQUENCE теперь выглядит так:



Создание SFC диаграммы

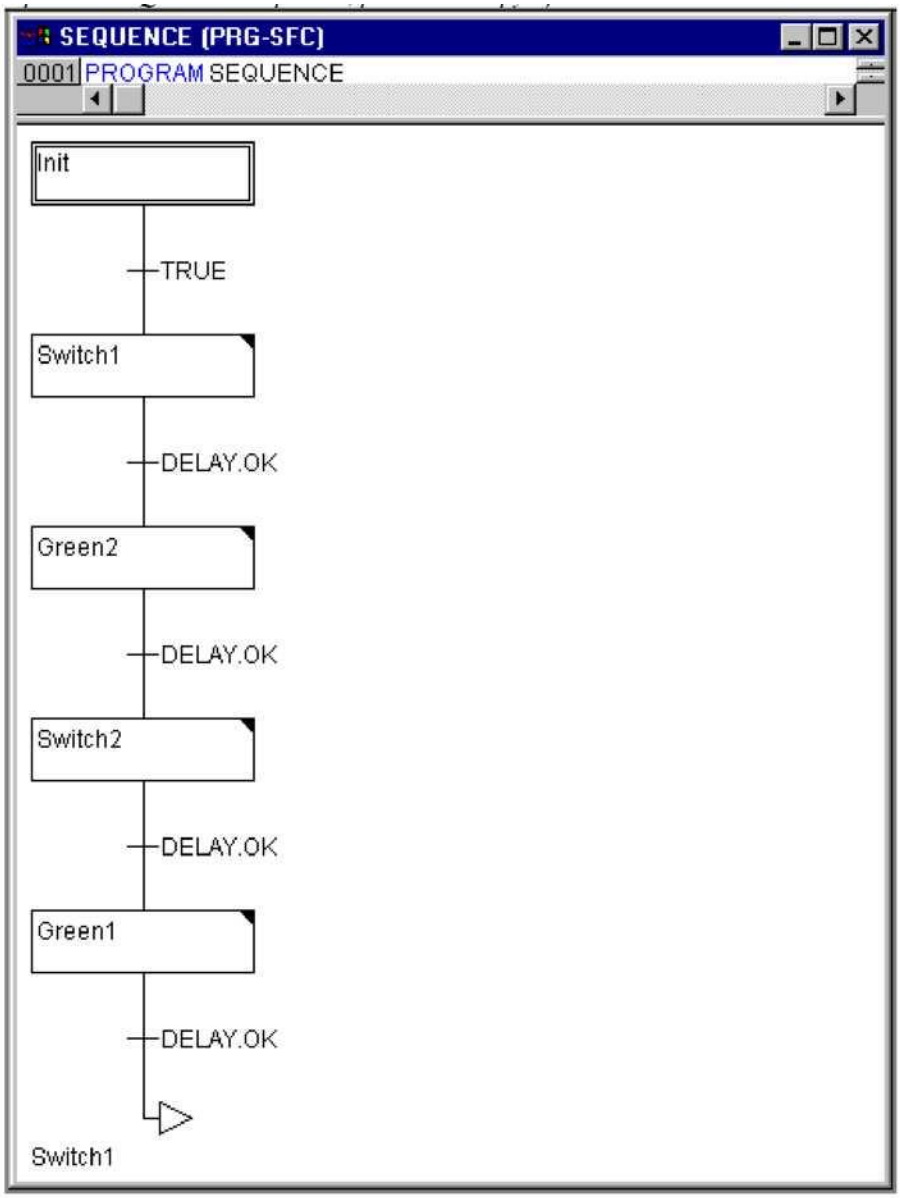
Первоначально SFC граф всегда состоит из этапа **"Init"**, перехода **"TransO"** и возврата назад к **Init**.

Прежде чем программировать конкретные этапы и переходы, выстроим структуру графа. Сначала понадобятся этапы для каждой стадии TRAFFICSIGNAL. Вставьте их, отмечая TransO и выбирая команды **"Insert" → "Step transition (after)"**. Повторите эту процедуру еще три раза.

Для редактирования названия перехода или этапа нужно просто щелкнуть мышкой на нужном тексте. Назовите первый переход после Init **"START"**, а все прочие переходы **"DELAY.OK"**.

Первый переход разрешается, когда START устанавливается в TRUE, все же прочие - когда DELAY в OK станет TRUE, т.е. когда заданный период закончится.

Этапы (сверху вниз) получают имена **Switch1, Green2, Switch2, Green1**. "Switch" должен включать жёлтую фазу, в Green1 TRAFFICSIGNAL1 будет зеленым, в Green2 TRAFFICSIGNAL2 будет зеленым. Наконец, измените адрес возврата Init на Switch1. Если всё сделано верно, диаграмма должна выглядеть так:

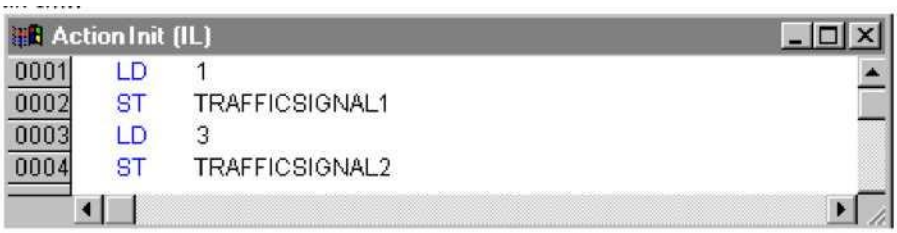


Далее программируются этапы. Если сделать двойной щелчок мышью на изображении этапа, то должен открыться диалог определения нового действия. В данном случае будем использовать язык IL (Список Инструкций).

Программирование этапов и переходов

Во время действия этапа Init проверяем, активен сигнал включения START или нет. Если сигнал не активен, то светофор выключается. Этого можно достичь, если записать в переменные TRAFFICSIGNAL1 и TRAFFICSIGNAL2 число 5.

Этап Init:



В Green1 TRAFFICSIGNAL1 будет зеленым (STATUS:=1), TRAFFICSIGNAL2 будет красным (STATUS:=3), задержка в 5000 миллисекунд.

Этап Green1:

Switch1 изменяет состояние TRAFFICSIGNAL1 на 2 (жёлтое) и, соответственно, TRAFFICSIGNAL2 на 4 (жёлто-красное). Кроме того, теперь устанавливается задержка в 2000 миллисекунд. Это выглядит так:

Этап Switch1:

Green2 включает красный в TRAFFICSIGNAL1 (STATUS:=3) и зеленый в TRAFFICSIGNAL2 (STATUS:=1). Задержка устанавливается в 5000 миллисекунд.

Этап Green2:

В Switch2 STATUS в TRAFFICSIGNAL1 изменяется на 4 (жёлто-красный), соответственно, TRAFFICSIGNAL2 будет 2 (жёлтый). Задержка теперь должна быть в 2000 миллисекунд.

Этап Switch2:

Первая версия программы закончена.

Если хотите проверить ее работу в режиме эмуляции, сделайте следующее:

Откройте POU PLC_PRG. Каждый проект начинает работу с PLC_PRG. Вставьте в него компонент и замените “AND” на “SEQUENCE”. Входы и выходы оставьте пока свободными.

Теперь можете откомпилировать программу (**'Project'** → **'Build'**) и проверить отсутствие ошибок. В окне сообщений вы должны увидеть текст: "0 Errors, 0 Warnings".

Теперь включите флажок **'Online'** → **'Simulation'** и дайте команду **'Online'** → **'Login'**. Запустите программу **'Online'** → **'Run'**.

Откройте программу SEQUENCE. Программа запущена, но не работает, поскольку переменная START должна иметь значение TRUE. Далее это будет делать PLC_PRG, но сейчас можно изменить ее вручную. Для этого щелкните дважды мышью по объявлению этой переменной. Ее значение теперь выделено цветом и равно TRUE. Дайте команду записи значений переменных (**'Online' → 'Write values'**). Теперь можно понаблюдать за работой программы. Активные шаги диаграммы выделяются голубым цветом.

Для продолжения редактирования программы закройте режим онлайн командой **'Online' → 'Logout'**.

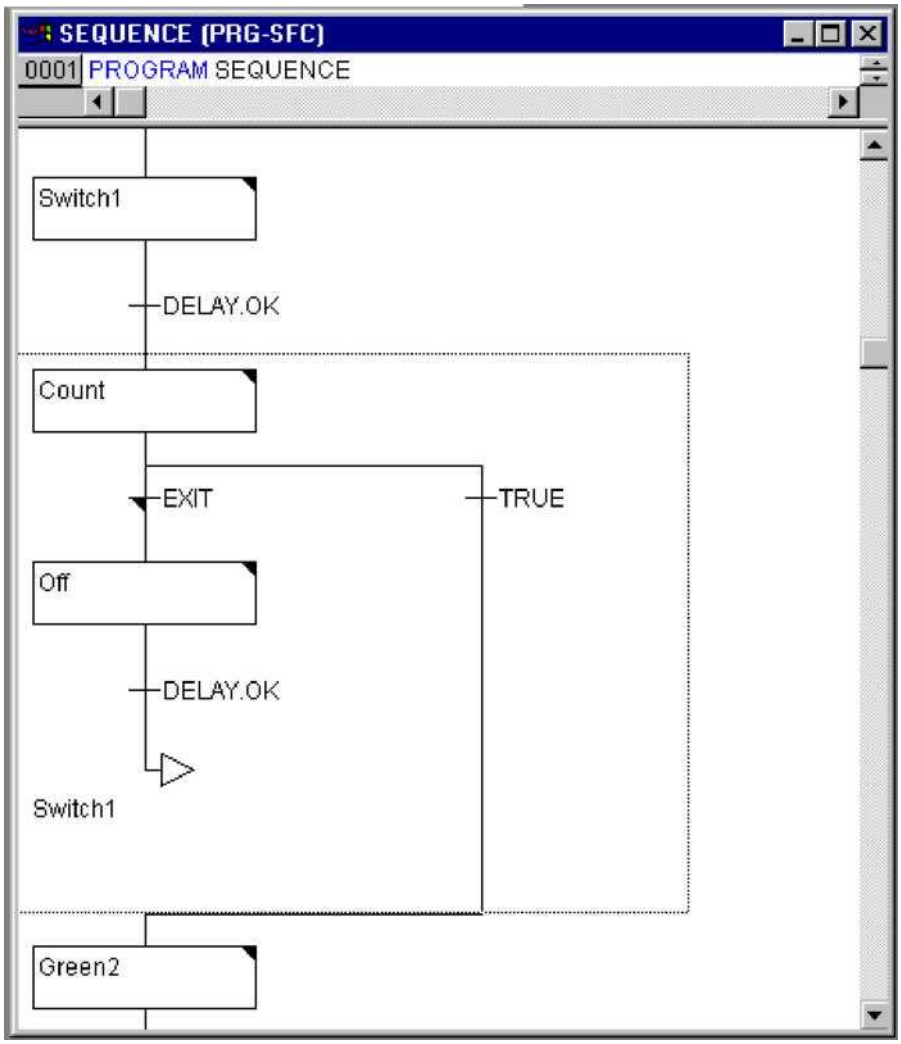
Блок «SEQUENCE» (вторая версия)

Немного усложним программу. Разумно будет выключать наши светофоры на ночь. Для этого создаем в программе счетчик, который после некоторого числа циклов TRAFFICSIGNAL произведет отключение устройства.

Для начала нужна новая переменная COUNTER типа INT. Объявите её в разделе объявлений SEQUENCE.

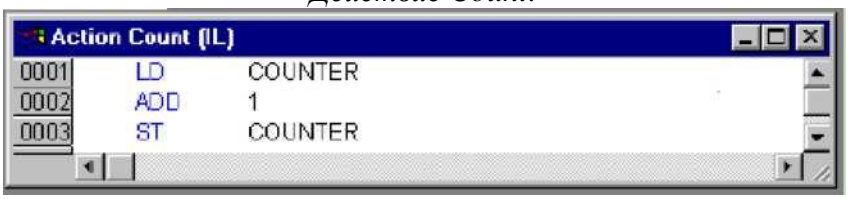
Теперь выберете переход после Switch1 и вставьте ещё один этап и переход. Выберете результирующий переход и вставьте альтернативную ветвь вправо. После левого перехода вставьте дополнительный этап и переход. После нового результирующего перехода вставьте удаленный переход (**jump**) на Init.

Назовите новые части так: верхний из двух новых этапов нужно назвать "Count" и нижний "Off". Переходы будут называться (сверху вниз слева на право) EXIT, TRUE и DELAY.OK. Теперь новые части должны выглядеть как фрагмент, выделенный рамкой.



Теперь два новых этапа и перехода необходимо наполнить содержанием.
На этапе Count выполняется только одно действие - COUNTER увеличивается на 1:

Действие Count:

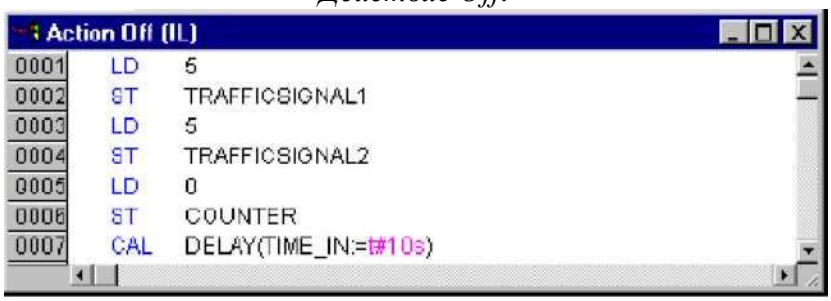


На переходе EXIT1 проверяется достижение счетчиком заданного значения, например 7:

Переход EXIT:

На этапе Off состояние обоих светофоров устанавливается в 5 (светофор выключен), COUNTER сбрасывается в 0 и устанавливается задержка времени в 10 секунд.

Действие Off:



Результат

В рассматриваемой ситуации ночь наступает после семи циклов TRAFFICSIGNAL. Светофоры полностью выключаются до рассвета, и процесс повторяется снова. При желании можно еще раз проверить работу программы в эмуляторе, прежде чем продолжить ее усовершенствование.

Блок программы PLC_PRG

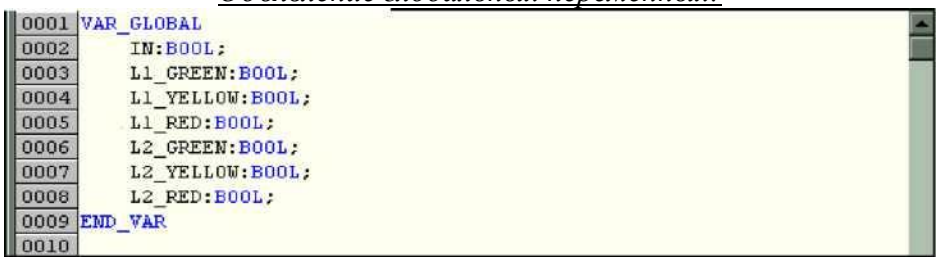
В блоке SEQUENCE определено два строго коррелированных во времени светофора. Теперь необходимо распределить входные и выходные переменные в блоке PLC_PRG. Следующие действия обеспечат возможность запустить систему выключателем IN и обеспечить переключение всех шести ламп (2 светофора) путем передачи “команд переключения” на каждом шаге SEQUENCE. Объявим теперь соответствующие булевы переменные (Boolean) для всех шести выходов и одного входа, затем создадим программу и сопоставим переменные соответствующим IEC адресам.

Следующий шаг - это объявление переменных LIGHT1 и LIGHT2 типа TRAFFICSIGNAL в редакторе объявлений.

Объявления LIGHT1 и LIGHT2

Для представления шести ламп светофоров нужно 6 переменных типа Boolean, которые не будут объявлены в разделе объявлений блока PLC_PRG, вместо этого используем глобальные переменные (Global Variables) из ресурсов (Resources). Двоичная входная переменная IN, необходимая для установки переменной START блока SEQUENCE в TRUE, будет определена таким же образом. Выберете вкладку **Resources** и откройте список **Global Variables**.

Объявление глобальных переменных:



В окне редактора выбираем Continuous Function Chart, после чего доступна соответствующая панель инструментов.

Щелкните правой клавишей мыши в окне редактора и выберите элемент **Box**. Щелкните на тексте AND и напишите ’’SEQUENCE”. Элемент автоматически преобразуется в SEQUENCE с уже определенными входными и выходными переменными.

Вставьте далее два элемента и назовите их TRAFFICSIGNAL. TRAFFICSIGNAL – это функциональный блок, в котором будет три красных знака вопроса, которые нужно заменить уже объявленными локальными переменными LIGHT1 и LIGHT2.

Теперь создайте элемент типа Input, который получит название IN и шесть элементов типа Output, которым нужно дать следующие имена: L1_green, L1_yellow, L1_red, L2_green, L2_yellow, L2_red.

Все элементы программы теперь на месте, необходимо теперь соединить входы и выходы. Для этого щелкните мышью на короткой линии входа/выхода и тяните ее (не отпуская клавишу мыши) к входу/выходу нужного элемента.

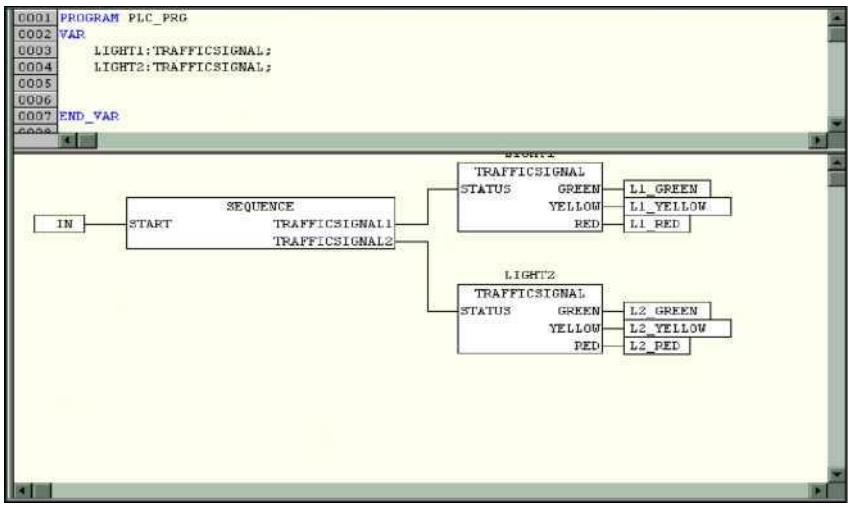
Программа должна принять вид, показанный ниже.

Теперь программа полностью готова.

Блок «TRAFFICSIGNAL» (эмуляция)

Теперь проверьте окончательно программу в режиме эмуляции. Убедитесь в правильности ее работы, контролируя последовательность выполнения и значения переменных в окнах редакторов CoDeSys.

PLC_PRG:



Визуализация примера

С помощью визуализации можно быстро и легко оживить переменные проекта. Нарисуем два светофора и их выключатель, который позволит нам включать и выключать блок управления светофором.

Создание новой визуализации

Для того чтобы создать визуализацию, выберите вкладку **Visualizations** в организаторе объектов. Теперь выполните команду **'Project' → 'Object Add'**.

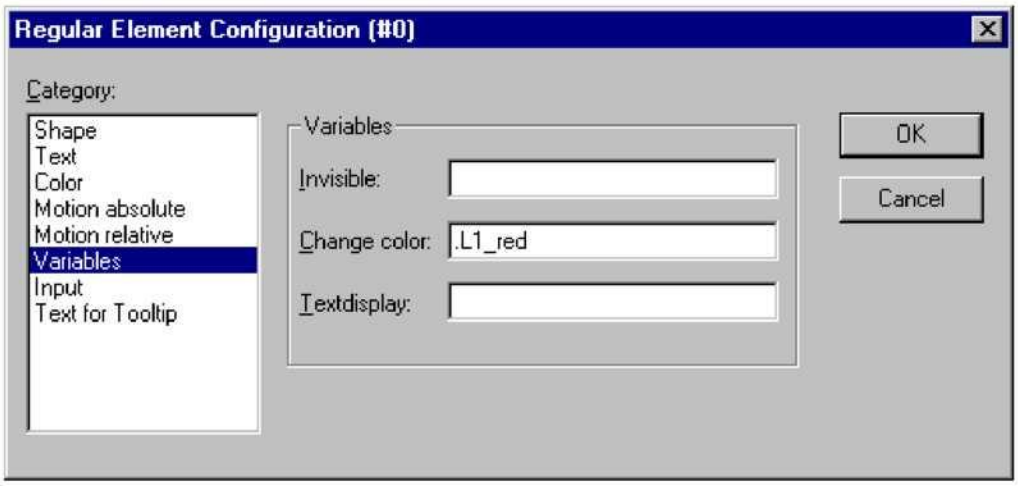
Диалог для создания новой визуализации:

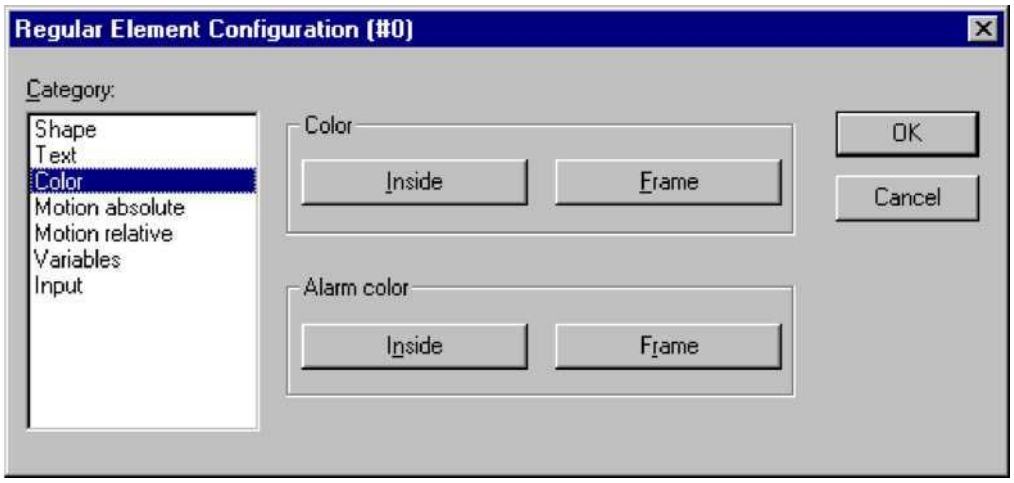
Введите любое имя для визуализации, например Lights. Когда нажмете кнопку Ok, откроется окно, в котором будет создаваться визуализация.

Вставка элемента в визуализацию

Для создания визуализации светофора выполните следующие действия:

- Выберите команду **'Insert' → 'Ellipse'** и нарисуйте окружность с диаметром около 2 сантиметров. Для этого щелкните мышью на рабочем поле и, удерживая левую кнопку мыши, растяните появившуюся окружность до требуемого размера.
- Дважды щелкните мышью на окружности. Появится диалоговое окно для настройки элемента визуализации.
- Выберите категорию **Variables** и в поле **Change color** введите имя переменной **.L1_red**. Вводить имя переменной удобно с помощью Input Assistant (клавиша <F2>). Глобальная переменная **L1_red** будет управлять цветом нарисованной окружности.
- Выберите категорию **Color**. В области **Color** нажмите кнопку **Inside** и в появившемся окне выберите любой нейтральный цвет, например, черный.
- Нажмите кнопку **Inside** в области **Alarm Color** и выберите красный цвет.





Полученная окружность будет черной, когда значение переменной ложно, и красной, когда переменная истинна.

Таким образом, создан первый фонарь первого светофора.

Остальные цвета светофора

Теперь вызовите команду копирования **'Edit' → 'Copy'** (<Ctrl>+<C>) и дважды выполните команду вставки **'Edit' → 'Paste'** (<Ctrl>+<V>). Получится две новых окружности. Перемещать эти окружности можно с помощью мышки. Расположите их так, чтобы они представляли собой вертикальный ряд в левой части окна редактора. Двойной щелчок по окружности приводит к открытию окна для настройки свойств элемента визуализации. В поле **Change Color** окон настройки свойств соответствующих окружностей введите следующие переменные:

для средней окружности: **.L1_yellow**

для нижней окружности: **.L1_green**

В категории **Color** в области **Alarm color** установите цвета окружностей (желтый и зеленый).

Корпус светофора

Теперь вызовите команду **“Insert” → “Rectangle”** и вставьте прямоугольник так, чтобы введенные ранее окружности находились внутри него. Выберите цвет прямоугольника и затем выполните команду **“Extras” → “ Send to back”**, которая переместит его на задний план. После этого окружности снова будут видны.

Активизируйте режим эмуляции, выполнив команду **“Online” → “Simulation”** (режим эмуляции активен, если перед пунктом меню **“Online” → “Simulation”** стоит галочка).

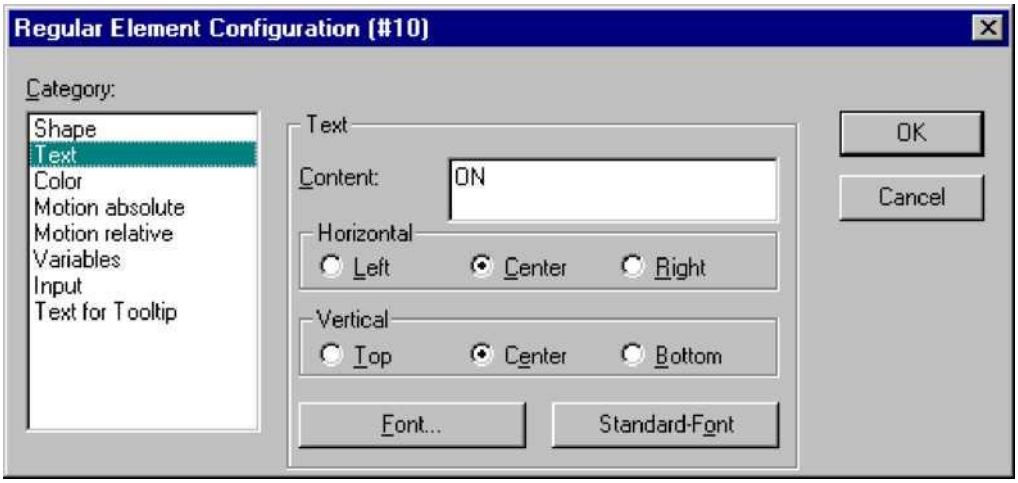
Запустите программу путем выполнения команд **“Online” → “Login”** и **“Online” → “Run”** и увидите, как будут меняться цвета светофора.

Второй светофор

Самый простой способ создать второй светофор - скопировать все элементы первого. Выделите элементы первого светофора и скопируйте их, выполнив команды **“Edit” → “Copy”** и **“Edit” → “Paste”**. Замените имена переменных, управляющих цветами (например, **.L1_red** на **.L2_red**), и второй светофор будет готов.

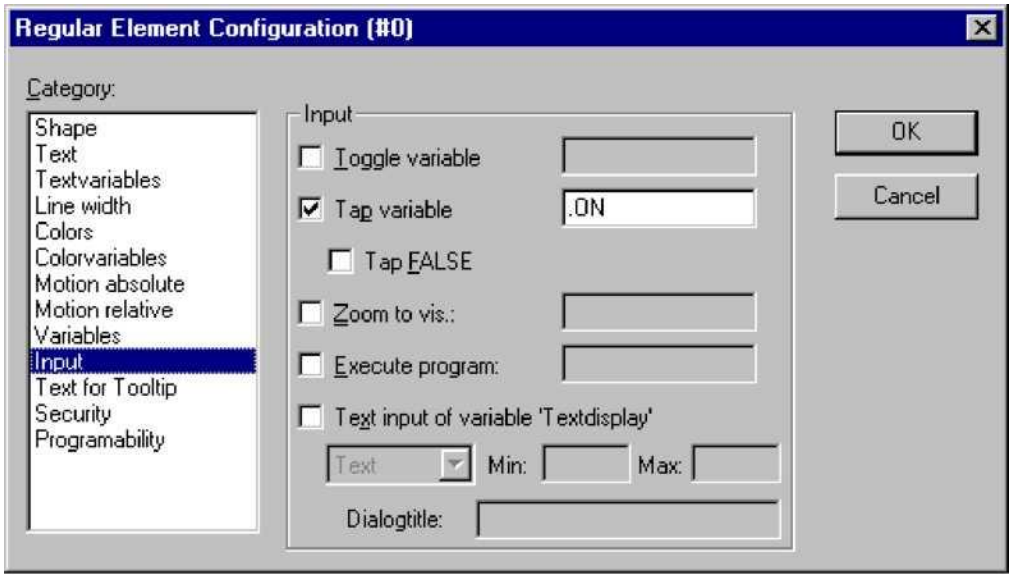
Переключатель ON

Как описано выше, вставьте прямоугольник, установите его цвет и введите переменную **.ON** в поле **Change Color** категории **Variables**. В поле **Content** категории **Text** введите имя **“ON”**.



Для того чтобы переменная ON переключалась при щелчке мышкой на этом элементе, в поле **Toggle variable** категории **Input** введите переменную **.ON**. Созданный нами переключатель будет включать/выключать светофоры.

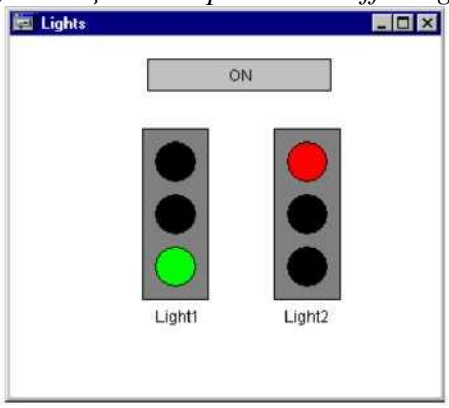
Отобразить включенное состояние можно цветом, как и для светофора. Впишите переменную в поле **Change Color**.



Надписи в визуализации

Под светофорами вставим два прямоугольника. В свойствах элемента в категории **Color** цвет границы (**frame**) прямоугольника задайте белым. В поле **Contents** (категория **Text**) введите названия светофоров **“Light1”** и **“Light2”**.

Визуализация для проекта Traffic Signal:



На этом написание программы и реализация ее визуализации окончены.

3. Требования к отчету

Отчет по лабораторной работе предоставляется в электронном виде в формате документа Microsoft Word (.doc или .docx), и должен содержать следующие основные пункты:

3.1 Скриншоты программного кода
3.2 Скриншоты визуализации
3.3 Краткое описание алгоритмов построения программы
3.4 Скриншоты полученных результатов работы программы
3.5 Выводы по лабораторной работе

4. Контрольные вопросы:

4.1 Что такое ПЛК?
4.2 Особенности ПЛК по сравнению с другими промышленными электронными приборами.
4.3 Виды ПЛК.
4.4 Устройство ПЛК.
4.5 Структура систем управления.
4.6 Интерфейсы ПЛК.
4.7 Основные характеристики ОВЕН ПЛК -100.
4.8 Схема работы ОВЕН ПЛК -100 в промышленной сети.
4.9 Как работает элемент **JUMP**?
4.10 Как работает элемент **EQ**?
4.11 Чем характеризуются переменные типа **INT**?

5. Список используемых источников

5.1. http://www.kipshop.ru/CoDeSys/steps/codesys_v23_ru.pdf
5.2. https://ru.wikipedia.org/wiki/CoDeSys
5.3. https://studfiles.net/preview/1979051/page:2/
5.4. http://dspace.kgsu.ru/xmlui/bitstream/handle/123456789/3587/22_%D0%A1%D0%B1%D1%80%D0%BE%D0%B4%D0%BE%D0%B2-%D0%9D%D0%91_2014_%D0%9C%D0%A3_3.pdf?sequence=1&isAllowed=y
5.5. http://www.codesys.ru/docs/3S_brochure_ru.pdf
5.6. https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D1%83%D0%B5%D0%BC%D1%8B%D0%B9_%D0%BB%D0%BE%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BA%D0%BE%D0%B

D%D1%82%D1%80%D0%BE%D0%BB%D0%BB%D0%B5%D1%80

5.7. http://manometr-com.ru/index/catalog/avtomatika/sistemyi-avtomatizaczii/programmiruemyie-logicheskie-kontrolleryi/programmiruemyij-logicheskij-kontroller-oven-plk-100.html

5.8. https://www.e-ope.ee/_download/euni_repository/file/1296/loengud_PDF.zip/lek_PDF/lek_1.pdf

Лабораторная работа №3

Автоматическое регулирование термоциклических воздействий на испытуемые образцы по заданному закону в среде CoDeSys на базе программируемого логического контроллера ОВЕН ПЛК-100

Цель работы: изучение продвинутого программирования в среде CoDeSys для практической реализации задачи автоматического регулирования температурного воздействия на испытуемые образцы по заданному циклу с помощью программируемого логического контроллера ОВЕН ПЛК-100.

1. Теоретическая часть

Автоматическое регулирование

Автоматическим регулированием называют поддержание на заданном уровне или изменение по определенному принципу какого-либо параметра технологического процесса, выполняемое без непосредственного участия человека с помощью специальных технических средств.

Машины, аппараты или агрегаты, в которых выполняют регулирование, называют *объектами регулирования,* а технологические параметры, подлежащие регулированию – *регулируемыми параметрами.*

Технические средства (прибор или совокупность приборов), при помощи которых осуществляют автоматическое регулирование, объединяют общим названием «*регулятор*». Объект регулирования и регулятор образуют *автоматическую систему регулирования* ***(АСР).*** Состав и устройство регулятора могут быть разной степени сложности, но, тем не менее, можно выделить ряд функциональных элементов автоматики, характерных для любой системы. Структурная схема АСР, изображенная на рис. 1, представляет регулятор расчлененным на функциональные элементы.

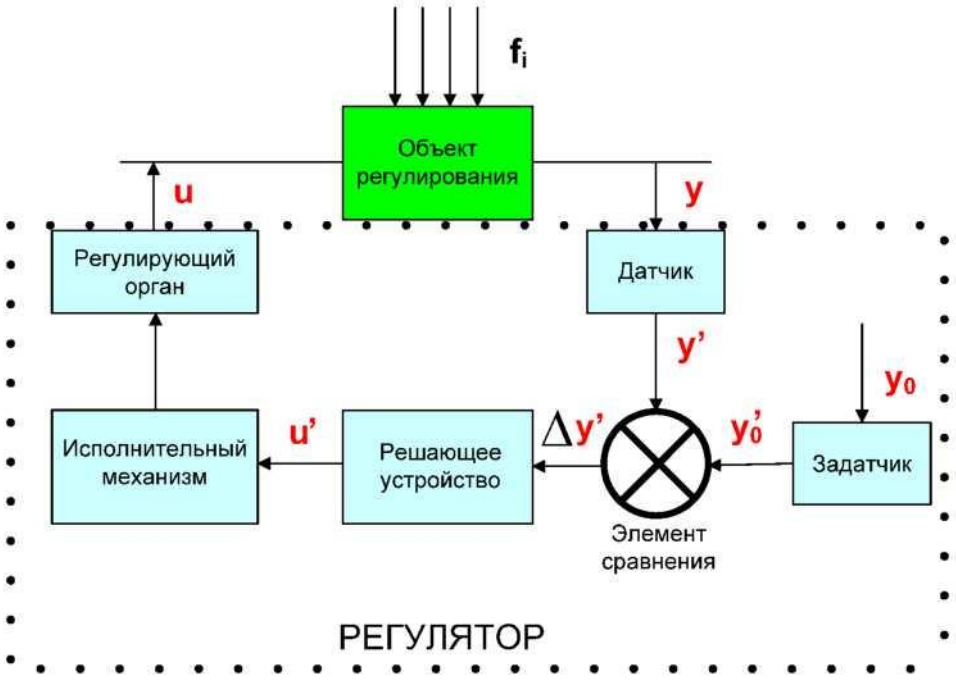


Рис. 1. Структурная схема автоматической системы регулирования

Датчик воспринимает текущее значение регулируемого параметра ***y*** и преобразует его в сигнал измерительной информации ***y′,*** поступающий на элемент сравнения. Здесь производится сравнение ***y′*** с сигналом ***y′0,*** вырабатываемым задатчиком и пропорциональным заданному значению регулируемого параметра ***y0.*** Разностный сигнал ***Δy′*** = ***y′ - y′,*** пропорциональный отклонению регулируемого параметра от заданного значения (*рассогласованию*), поступает на вход решающего устройства, которое формирует в определенной зависимости от ***y′*** сигнал управляющего воздействия ***u'.***

Под действием этого сигнала исполнительный механизм перемещает регулирующий орган. Регулирующий орган непосредственно воздействует на объект регулирования, изменяя подводимый к нему поток энергии или вещества. Это воздействие ***u*** направлено на приведение регулируемого параметра к заданному значению и называется *регулирующим.*

Кроме регулирующего воздействия, на объект регулирования влияют также и другие факторы, называемые *возмущающими воздействиями* f_i, из-за которых регулируемый параметр отклоняется от заданного значения.

Позиционное регулирование

Законом регулирования называют функциональную связь между регулирующим воздействием и отклонением регулируемого параметра от заданного значения:

$$u(\tau) \quad f(y(\tau) \quad y_0). \qquad (1)$$

Простейшим законом регулирования является ***позиционный,*** при котором регулятор в зависимости от текущего значения регулируемого параметра переключает регулирующее воздействие с одного фиксированного уровня на другой. В практике используют обычно двух- и трехпозиционное регулирование, при которых таких уровней, соответственно, два или три. Математическая формулировка *идеального* (без зоны нечувствительности) двухпозиционного регулирования имеет вид:

$$u(\tau) = U_1 \text{ при } \Delta y \leq 0 \text{ или } y(\tau) \leq y_0 \qquad (2)$$

$$u(\tau) = U_2 \text{ при } \Delta y > 0 \text{ или } y(\tau) > y_0 \qquad (3)$$

Для работы *реального двухпозиционного регулятора* характерно наличие *зоны нечувствительности.* Зона нечувствительности *(зона неоднозначности, гистерезис* или *дифференциал)* – это разность значений регулируемого параметра, при которых происходят прямое и обратное переключения регулирующего воздействия.

Статическая характеристика реального двухпозиционного регулятора изображена на рис. 2.

При повышении регулируемого параметра до значения $y_{макс}$ регулирующее воздействие переключается с уровня U_1 на уровень U_2, обратное переключение происходит при уменьшении регулируемого параметра до значения $y_{мин}$. Таким образом, дифференциал регулятора составляет

$$\delta = y_{макс} - y_{мин}. \qquad (4)$$

Задание двухпозиционному регулятору устанавливают либо указанием предельных значений регулируемого параметра $y_{макс}$ и $y_{мин}$, либо с помощью уставки $T_{уст}$ и дифференциала **δ,** как это показано на рис. 3 для примера регулирования температуры.

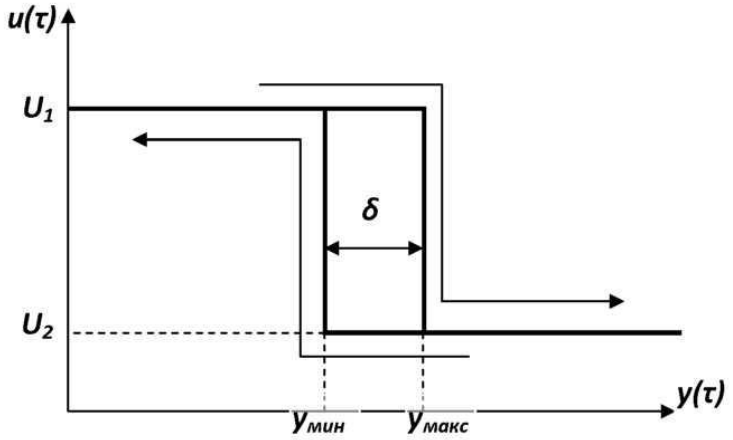


Рис. 2. Статическая характеристика двухпозиционного регулятора

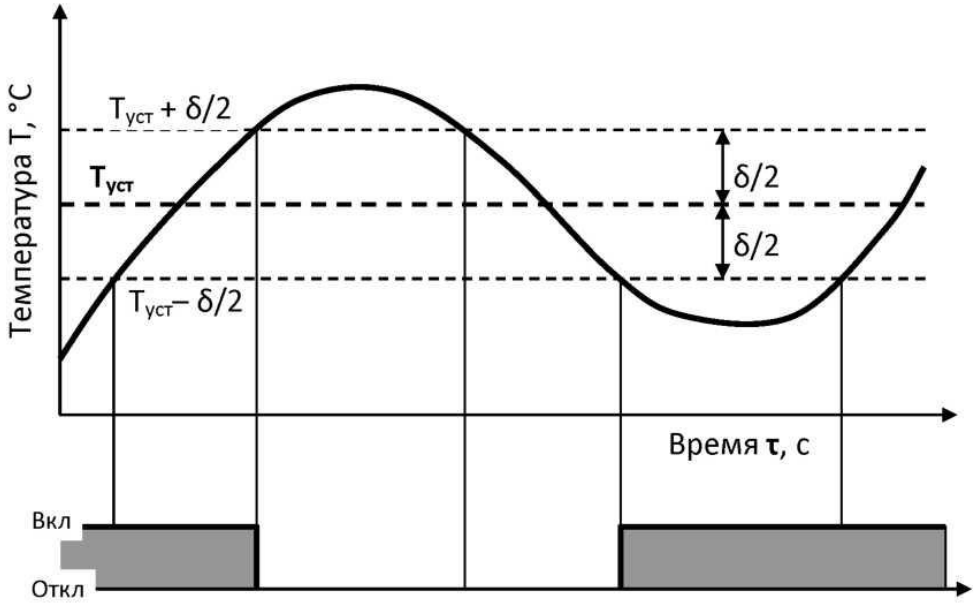


Рис. 3. График процесса двухпозиционного регулирования

Тип логики двухпозиционного регулятора может быть различным. На рис. 3 изображена логика «прямой гистерезис», которую применяют для управления работой нагревателя. Включение выходного реле регулятора в этом случае происходит при условии $\mathbf{T < T_{уст} - \delta/2}$, а выключение при $\mathbf{T > T_{уст} + \delta/2}$.

Качество позиционного регулирования характеризуется периодом колебаний $\boldsymbol{\tau_к}$, амплитудой $\boldsymbol{A}$, и условной статической ошибкой регулирования $\mathbf{\Delta T_{ст}}$ (рис. 4).

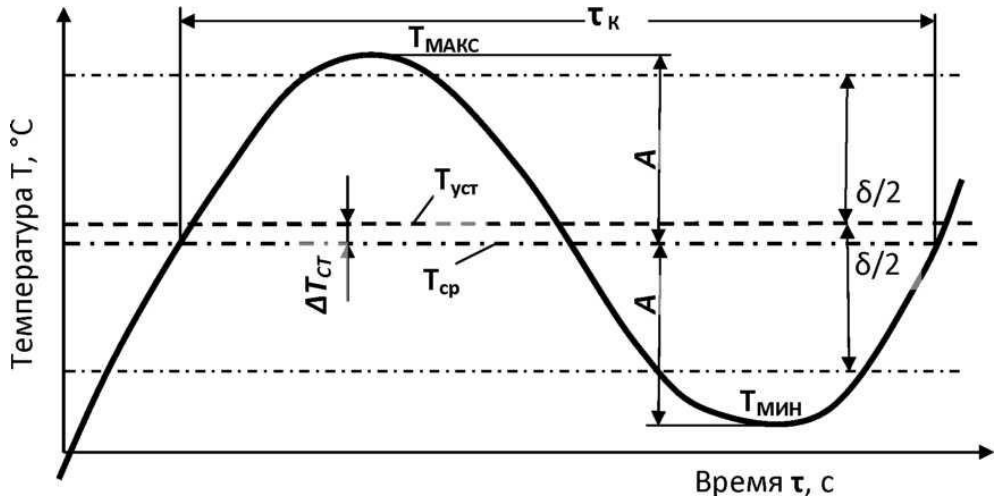


Рис. 4. Показатели качества при двухпозиционном регулировании

Амплитуду A можно определить как половину диапазона колебаний; для примера на рис. 3:

$$A = (T_{МАКС} - T_{МИН})/2. \qquad (5)$$

Условная статическая ошибка $\Delta T_{ст}$ определяется как разность между фактическим средним значением регулируемого параметра и уставкой регулирования:

$$\Delta T_{СТ} = T_{СР} - T_{УСТ}. \qquad (6)$$

Фактическое среднее значение регулируемого параметра $T_{ср}$ определяется как среднее арифметическое

$$T_{СР} = (T_{МАКС} + T_{МИН})/2. \qquad (7)$$

Параметрами настройки двухпозиционного регулятора являются заданное значение или уставка (y_0, $T_{уст}$), уровни регулирующего воздействия (U_1 и U_2), зона неоднозначности (δ).

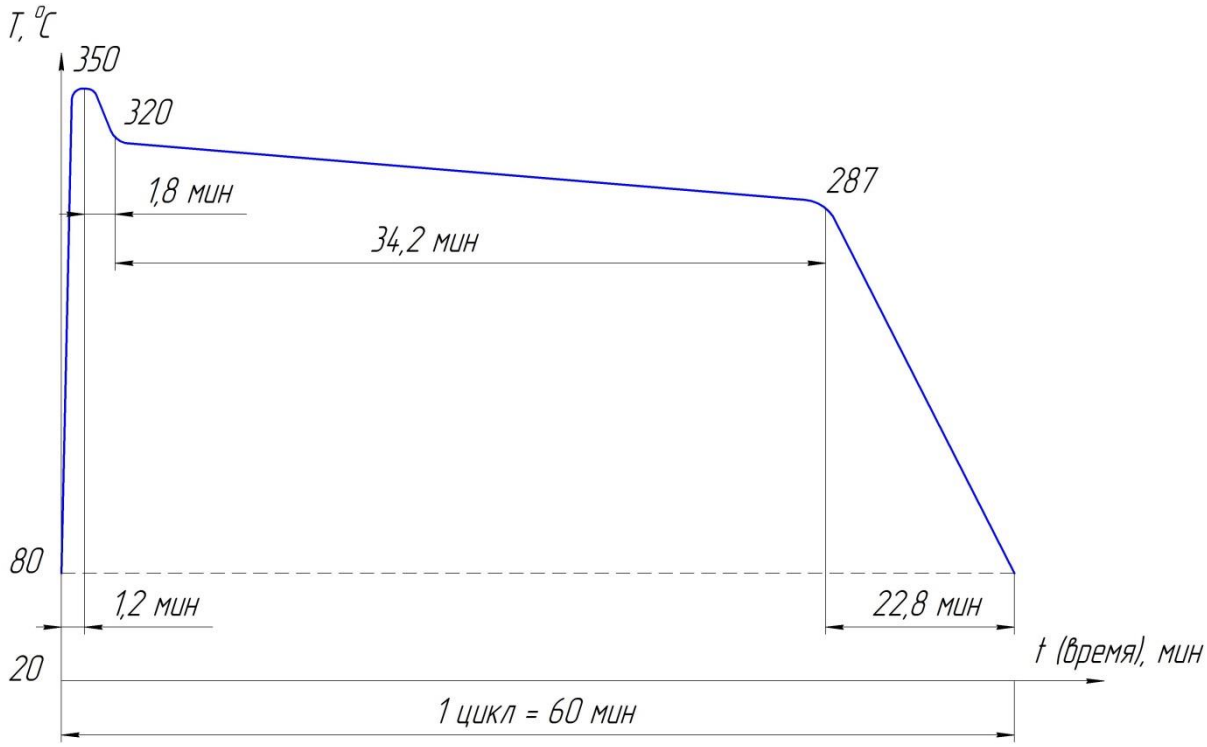
2. Практическая часть

2.1. Программа регулирования термоциклических воздействий по заданному закону в среде CoDeSys на базе программируемого логического контроллера ОВЕН ПЛК-100

Задание: написать программу регулирования, обеспечивающую термоциклирование композитных образцов по заданному закону из температуры от времени цикла.

Исходные данные:
начальная температура по термопаре – 80°С;
количество циклов для реализации – 3;

Требуемый график термоцикла приведен ниже:



Примечание: при учебной реализации процесса необходимо изменить время цикла, качественно сохранив график термоциклирования.

Этап 1 – нагрев с 80 до 350°С – может осуществляться с максимальной скоростью 10°С/сек;

Этап 2 – быстрое охлаждение с 350 до 320°С в течение 20 секунд – максимальная скорость охлаждения в этом случае может составлять 5°С/сек;

Этап 3 – плавное охлаждение с 320 до 287°С в течение 120 секунд – максимальная скорость охлаждения в этом случае может составлять 5°С/сек;

Этап 4 – плавное охлаждение с 287 до 80°С в течение 60 секунд – максимальная скорость охлаждения в этом случае может составлять 5°С/сек;

Далее приведены фотографии реальной установки для термоциклирования и график реализованного с ее помощью термоцикла.

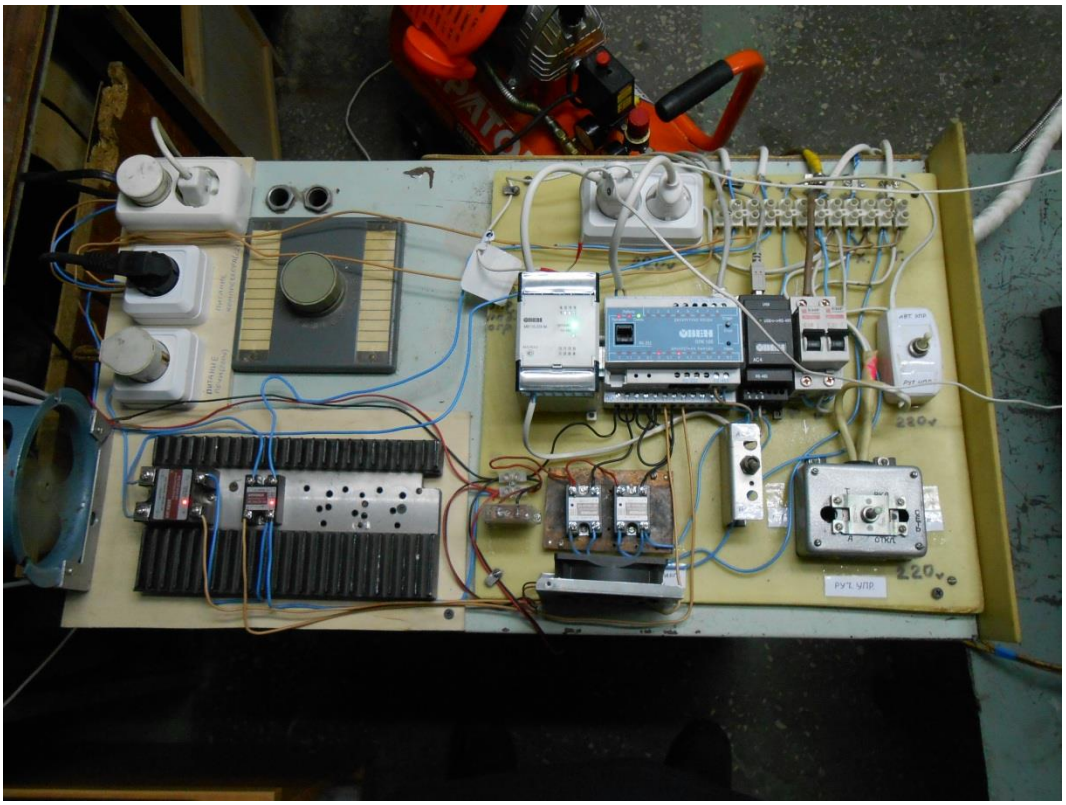


Рис. 2.1. Фотография установки для испытаний КМ при термоциклических воздействиях

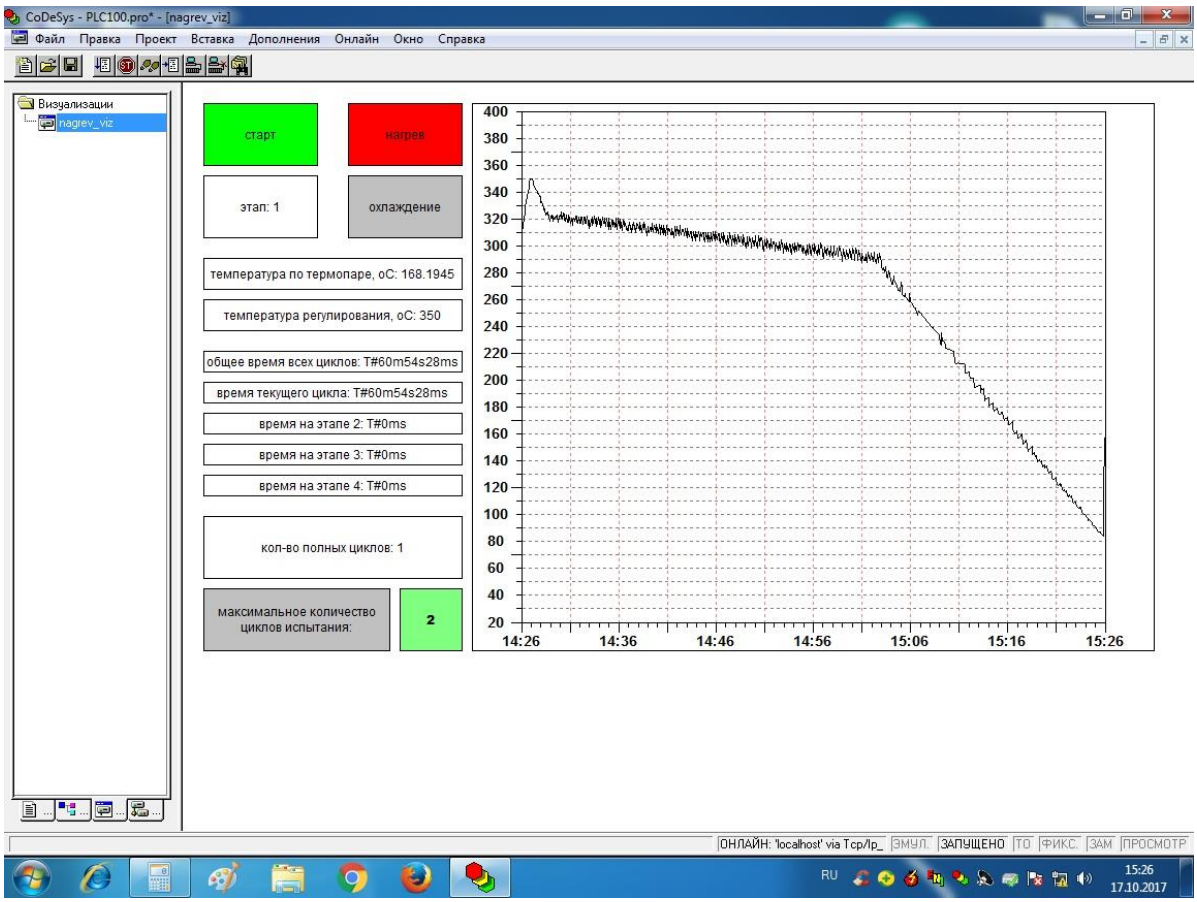
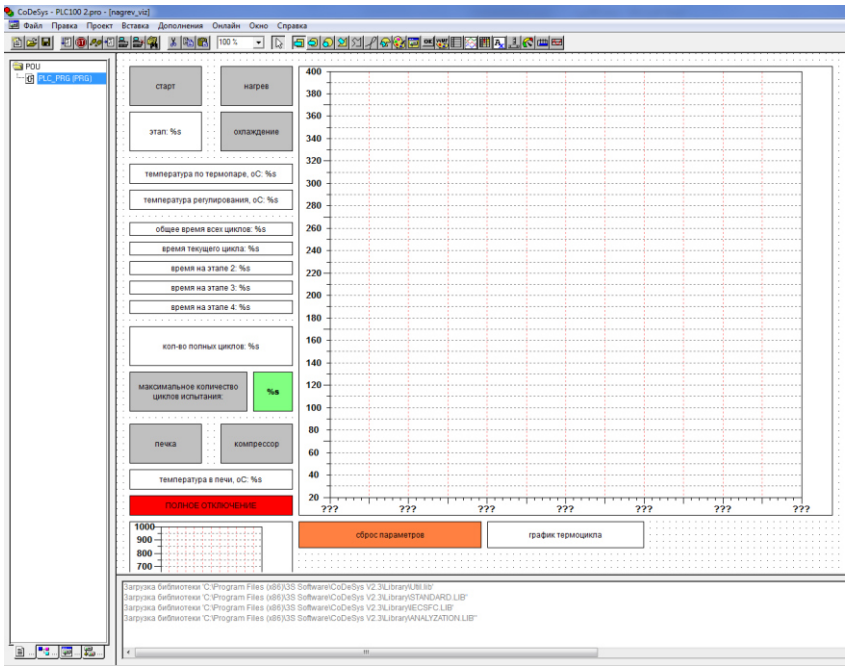
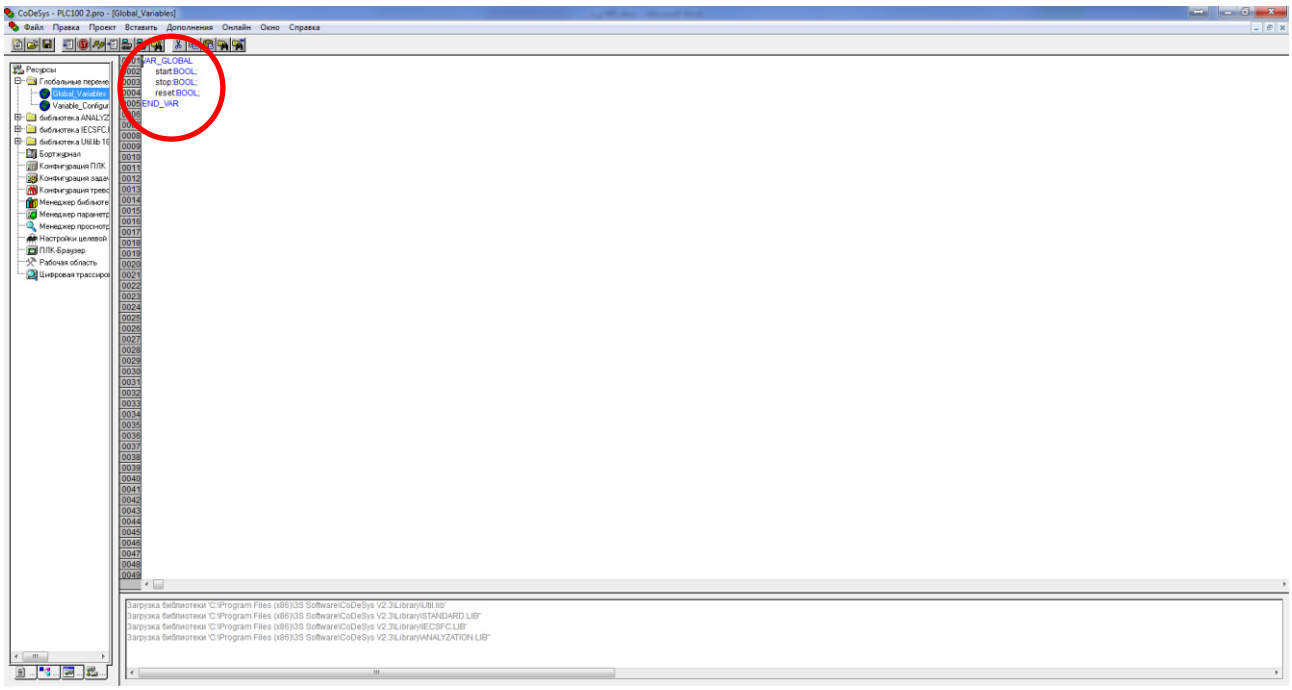


Рис. 2.2. График реализованного на испытательной установке термоцикла

Окончательный вид визуализации программы регулирования представлен на рис. далее.



В качестве глобальных переменных объявлены следующие переменные

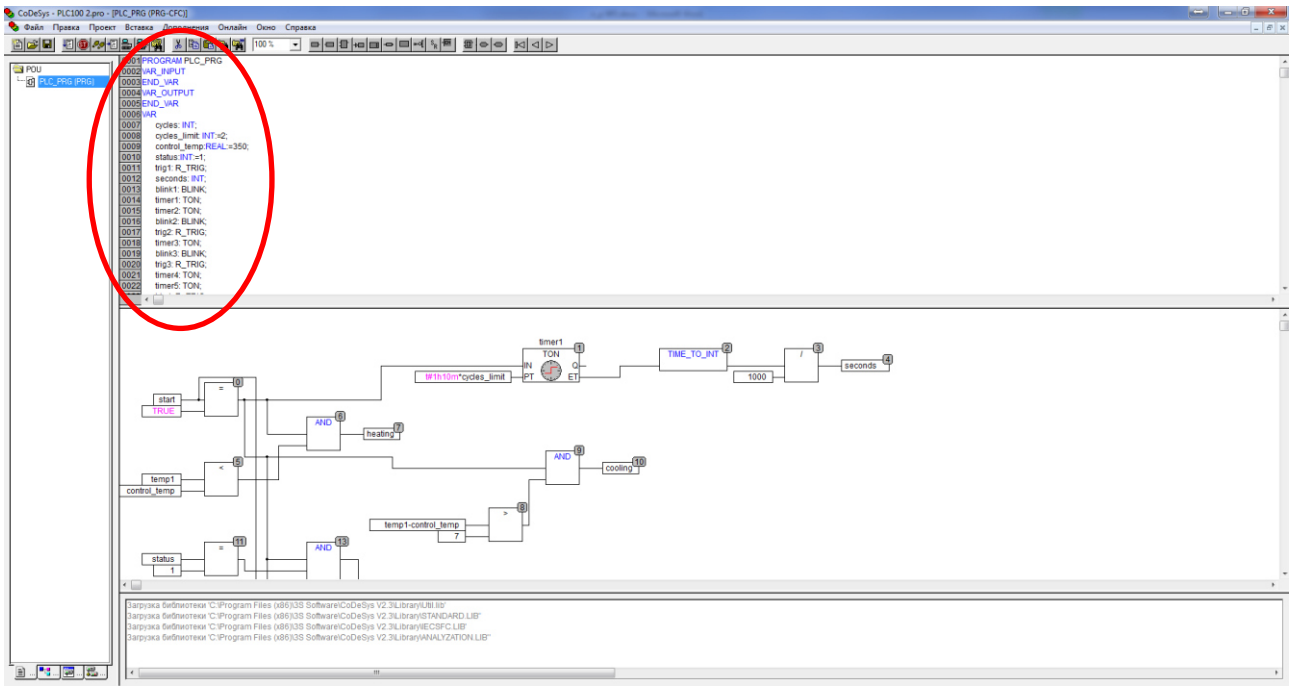


К данным переменным необходимо добавить также булевые переменные **heating** и **cooling**.

BOOL константы:

BOOL константы могут иметь значение TRUE или FALSE.

В качестве локальных переменных используем переменные, представленные на рисунке далее.



REAL/LREAL константы:

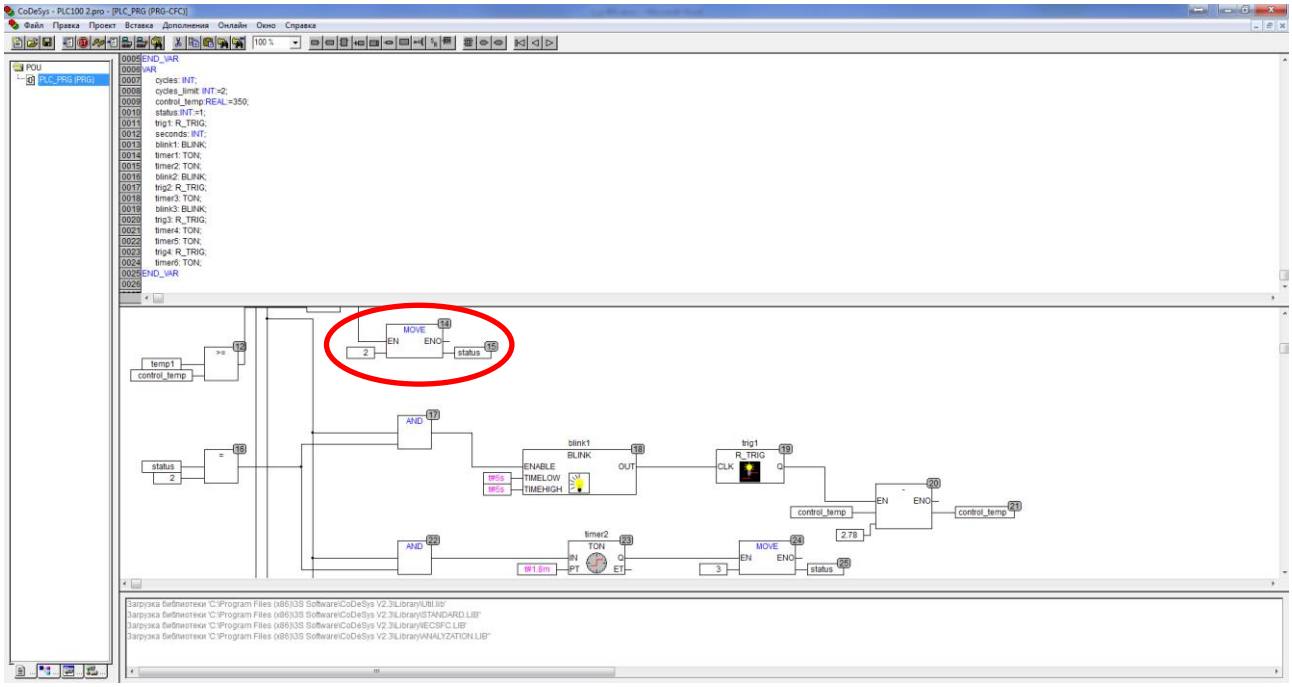
REAL и LREAL константы представляются в формате с десятичной точкой либо в экспоненциальном формате. Запятая вместо точки не допускается.

Примеры:

7.4 но не 7,4

1.64e+009 но не 1,64e+009

Для перехода от этапа к этапу используется переменная **status**.



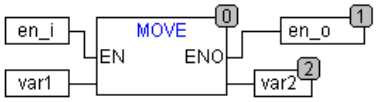
Команда **MOVE**

MOVE

Присвоение значения одной переменной другой соответствующего типа. В графических редакторах CFC и LD существует возможность управлять разрешением работы блока (разрешать или запрещать операцию) с помощью входов EN/EN0. В FBD этого делать нельзя.

Пример применения EN/EN0 в CFC:

Только если значение *en_i* равно TRUE, значение переменной *var1* будет присвоено *var2*.



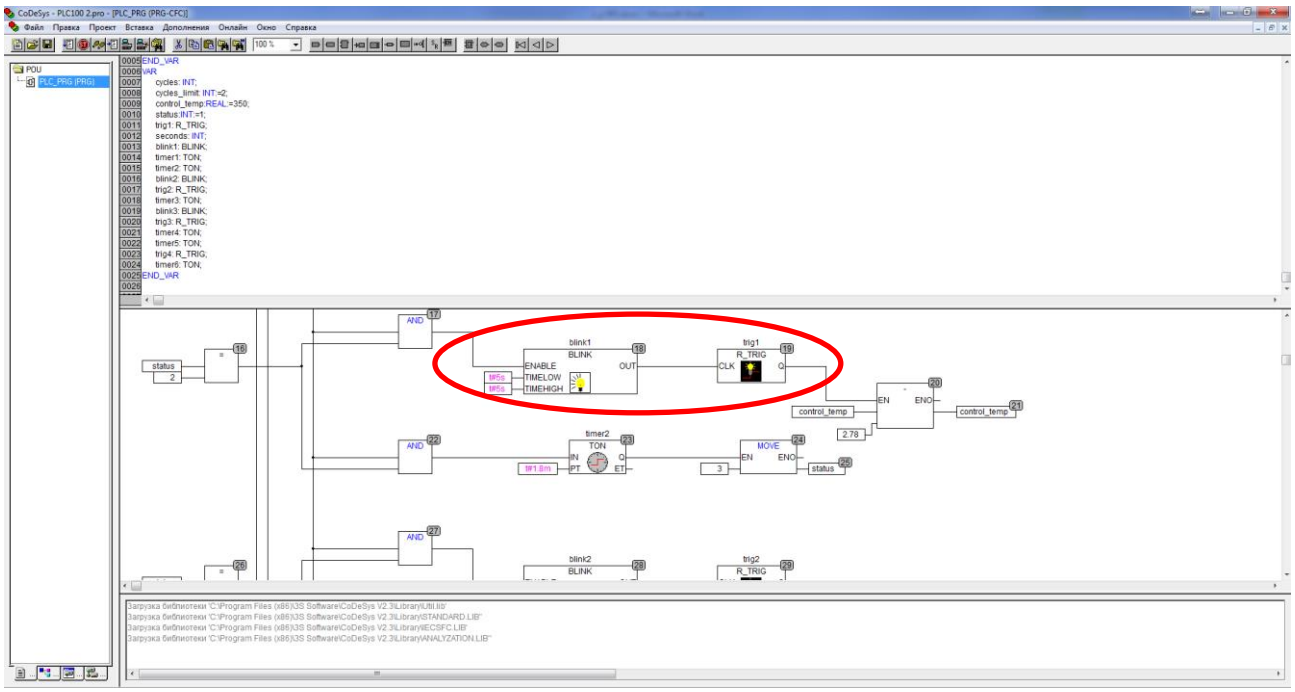
Пример IL:

```
LD ivar1
MOVE ivar2
ST ivar2 (* Результат: ivar2 принимает значение ivar1 *)
    (! Аналогичный результат дает:
          LD ivar1
          ST ivar2 )
```

Пример ST:

```
ivar2 := MOVE(ivar1);
         (! Аналогичный результат дает: ivar2 := ivar1; )
```

Для того, чтобы организовать нагрев и охлаждение с требуемой скоростью, а также изменять контрольную температуру в данный момент времени, используется связка команд **BLINK** и **R_TRIG**.



BLINK

Входит в util.lib.

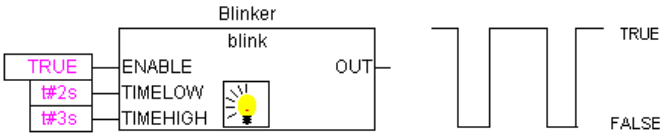
Функциональный блок 'генератор прямоугольных импульсов'

Входы: ENABLE типа BOOL, TIMELOW и TIMEHIGH типа TIME. Выход OUT типа BOOL.

Генератор запускается по входу ENABLE = TRUE. Длительность импульса задается TIMEHIGH, длительность паузы TIMELOW.

Примечание: При переходе ENABLE в FALSE, выход OUT остается в том состоянии, в котором он был в этот момент. Если вам необходимо чтобы выходная переменная сбрасывалась в FALSE при ENABLE равном FALSE, то используйте выражение "OUT AND ENABLE" на выходе (т.е. добавьте блок AND на выход и на второй вход подайте ENABLE) .

Пример CFC:



R_TRIG

Входит в *standard.lib*.

Детекторы импульсов

```
FUNCTION_BLOCK R_TRIG
VAR_INPUT
 CLK : BOOL;
END_VAR
VAR_OUTPUT
 Q : BOOL;
END_VAR
VAR
 M : BOOL := FALSE;
END_VAR
 Q := CLK AND NOT M;
 M := CLK;
```

Функциональный блок R_TRIG генерирует импульс по переднему фронту входного сигнала.

Выход Q равен FALSE до тех пор, пока вход CLK равен FALSE. Как только CLK получает значение TRUE, Q устанавливается в TRUE. При следующем вызове функционального блока выход сбрасыва-ется в FALSE. Таким образом, блок выдает единичный импульс при каждом переходе CLK из FALSE в TRUE.

Пример объявления:

```
RTRIGInst : R_TRIG ;
```

Пример IL:

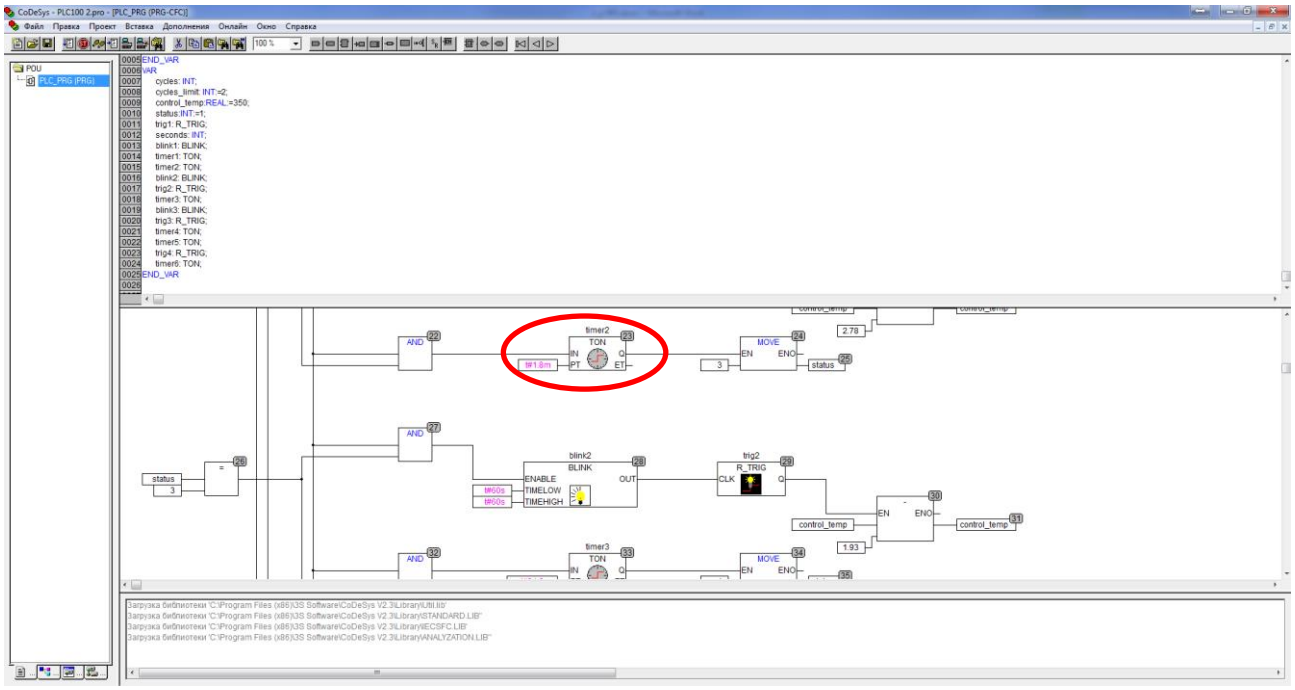
```
CAL RTRIGInst(CLK := VarBOOL1)
LD  RTRIGInst.Q
ST  VarBOOL2
```

Пример FBD:

Пример ST:

```
RTRIGInst(CLK:= VarBOOL1);
VarBOOL2 := RTRIGInst.Q;
```

Для перехода с этапа, на котором требуется обеспечить определенное время выдержки, на следующий этап, используется команда **TON**.



TON

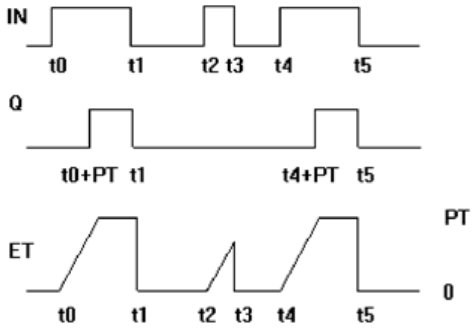
Входит в *standard.lib*.

Функциональный блок 'таймер с задержкой включения'.

TON(IN, PT, Q, ET) Входы IN и PT типов BOOL и TIME соответственно. Выходы Q и ET аналогично типов BOOL и TIME.

Пока IN равен FALSE, выход Q = FALSE, выход ET = 0. Как только IN становится TRUE, начинается отсчет времени (в миллисекундах) на выходе ET до значения, равного PT. Далее счетчик не увеличивается. Q равен TRUE, когда IN равен TRUE и ET равен PT, иначе FALSE. Таким образом, выход Q устанавливается с задержкой PT от фронта входа IN.

Временная диаграмма работы TON:



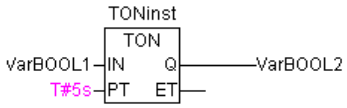
Пример объявления:

```
TONInst : TON ;
```

Пример IL:

```
CAL TONInst(IN := VarBOOL1, PT := T#5s)
LD  TONInst.Q
ST  VarBOOL2
```

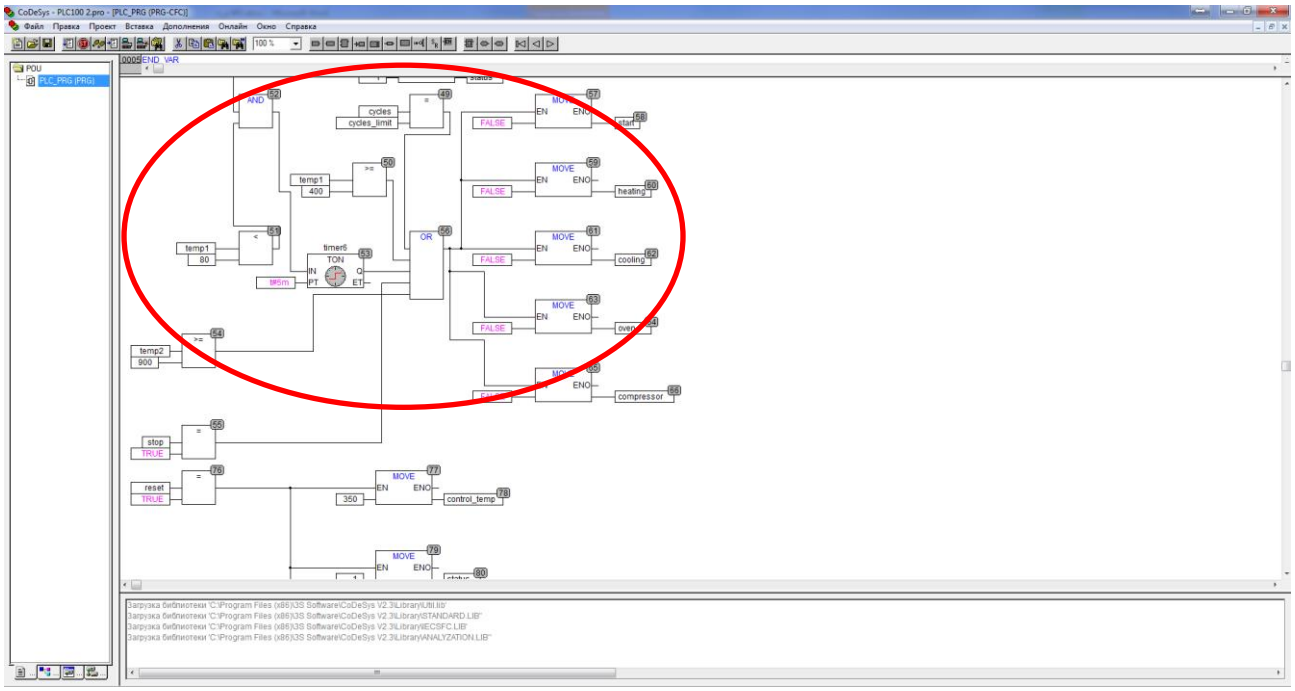
Пример FBD:



Пример ST:

```
TONInst(IN := VarBOOL1, PT:= T#5s);
```

Автоматическое отключение после достижения необходимого числа циклов реализовано комплексом условий с операторами **EQ** (=), **AND** и **OR**.



EQ

Равно

Двоичный оператор возвращает TRUE, если значение первого параметра равно второму.

Операнды могут быть типов BOOL, BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL, TIME, DATE, TIME_OF_DAY, DATE_AND_TIME и STRING.

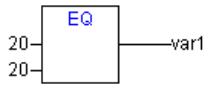
Пример IL:

```
LD 40
EQ 40
ST Var1 (*Результат - ИСТИНА*)
```

Пример ST:

```
VAR1 := 40 = 40;
```

Пример FBD:



AND

Побитное И. Операция применима к типам BOOL, BYTE, WORD или DWORD.

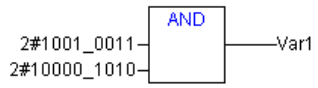
Пример IL:

```
Var1 BYTE
LD 2#1001_0011
AND 2#1000_1010
ST Var1 (* Результат 2#1000_0010 *)
```

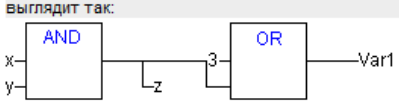
Пример ST:

```
var1 := 2#1001_0011 AND 2#1000_1010
```

Пример FBD:



Внимание: В логических выражениях нельзя гарантировать присваивание промежуточных результатов. Например, если условие SFS перехода выглядит так:



то, значение переменной z может быть не присвоено. Это происходит по причине оптимизации вычислений компилятором. Если значения x и y FALSE, то конечный результат очевиден и остаток выражения вычислять не нужно.

OR

Побитное ИЛИ. Операция применима к типам BOOL, BYTE, WORD или DWORD.

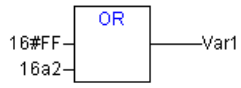
Пример IL:

```
var1 :BYTE;
LD 2#1001_0011
OR 2#1000_1010
ST  var1  (*  Результат 2#1001_1011 *)
```

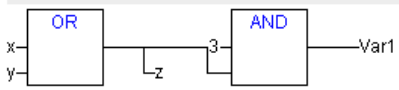
Пример ST:

```
Var1 := 2#1001_0011 OR 2#1000_1010
```

Пример FBD:



Внимание: См. примечание к AND.



Дополнительные команды, используемые в проекте:

Команда «**+**» эквивалентна команде **ADD**

ADD

Сложение переменных типов: BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL и LREAL.

Две переменных типа TIME можно складывать (напр. t#45s + t#50s = t#1m35s). Результат имеет тип TIME.

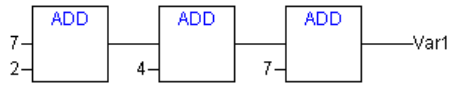
Пример IL:

```
LD  7
ADD 2,4,7
ST  Var1
```

Пример ST:

```
var1 := 7+2+4+7;
```

Пример FBD:



Команда «**-**» эквивалентна команде **SUB**

SUB

Вычитание значений переменных типов: BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL и LREAL.

Переменной TIME можно присвоить результат вычитания двух других переменных типа TIME. Отрицательное время не определено.

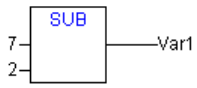
Пример IL:

```
LD  7
SUB 2
ST  Var1
```

Пример ST:

```
var1 := 7-2;
```

Пример FBD:



Команда «*» эквивалентна команде **MUL**

MUL

Перемножение значений переменных типов: BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL и LREAL.

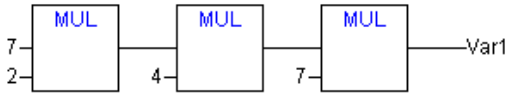
Пример IL:

```
LD  7
MUL 2,4,7
ST  Var1
```

Пример ST:

```
var1 := 7*2*4*7;
```

Пример FBD:



Команда «≥» эквивалентна команде **GE**

GE

Больше или равно

Двоичный оператор возвращает TRUE, если значение первого параметра больше или равно второму.

Операнды могут быть типов BOOL, BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL, TIME, DATE, TIME_OF_DAY, DATE_AND_TIME и STRING.

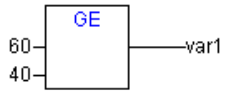
Пример IL:

```
LD 60
GE 40
ST Var1  (*Результат - ИСТИНА*)
```

Пример ST:

```
VAR1 := 60 >= 40;
```

Пример FBD:



Команда «<» эквивалентна команде **LT**

LT

Меньше

Двоичный оператор возвращает TRUE, если значение первого параметра меньше второго.

Операнды могут быть типов BOOL, BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL, TIME, DATE, TIME_OF_DAY, DATE_AND_TIME и STRING.

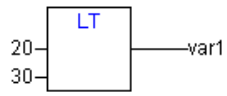
Пример IL:

```
LD 20
LT 30
ST Var1 (* Результат ИСТИНА* )
```

Пример ST:

```
VAR1 := 20 < 30;
```

Пример FBD:



3. Требования к отчету

Отчет по лабораторной работе предоставляется в электронном виде в формате документа Microsoft Word (.doc или .docx), и должен содержать следующие основные пункты:

3.1 Скриншоты программного кода
3.2 Скриншоты визуализации
3.3 Краткое описание алгоритмов построения программы
3.4 Скриншоты полученных результатов работы программы
3.5 Выводы по лабораторной работе

4. Контрольные вопросы:

4.1 Что такое автоматическое регулирование?
4.2 Что такое регулятор?
4.3 Что такое АСР? Принципы работы АСР?
4.4 Что такое величина рассогласования?
4.5 Особенности позиционного регулирования.
4.6 Понятие зоны нечувствительности.
4.7 Понятие уставки.
4.8 Как работает элемент **BLINK**?
4.9 Как работает элемент **GE**?

4.10 Как работает элемент **LT**?

4.11 Чем характеризуются переменные типа **LREAL**?

5. Список используемых источников

5.1. http://www.kipshop.ru/CoDeSys/steps/codesys_v23_ru.pdf

5.2. https://ru.wikipedia.org/wiki/CoDeSys

5.3. https://studfiles.net/preview/1979051/page:2/

5.4. http://dspace.kgsu.ru/xmlui/bitstream/handle/123456789/3587/22_%D0%A1%D0%B1%D1%80%D0%BE%D0%B4%D0%BE%D0%B2-%D0%9D%D0%91_2014_%D0%9C%D0%A3_3.pdf?sequence=1&isAllowed=y

5.5. http://www.codesys.ru/docs/3S_brochure_ru.pdf

5.6. https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D1%83%D0%B5%D0%BC%D1%8B%D0%B9_%D0%BB%D0%BE%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D0%BB%D0%B5%D1%80

5.7. http://manometr-com.ru/index/catalog/avtomatika/sistemyi-avtomatizaczii/programmiruemyie-logicheskie-kontrolleryi/programmiruemyij-logicheskij-kontroller-oven-plk-100.html

5.8. https://www.e-ope.ee/_download/euni_repository/file/1296/loengud_PDF.zip/lek_PDF/lek_1.pdf

5.9. https://media.ls.urfu.ru/Projects/518/uploaded/files/91263_%D0%9C%D0%B5%D1%82%D0%BE%D0%B4%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B5%20%D1%83%D0%BA%D0%B0%D0%B7%D0%B0%D0%BD%D0%B8%D1%8F%20%D0%BA%20%D0%9B%D0%A0%2022%D0%92(%D1%80%D0%B5%D0%B4).pdf

САМОСТОЯТЕЛЬНАЯ КОНТРОЛЬНАЯ РАБОТА

по реализации автоматического регулирования термического воздействия по заданному закону при формовании композита на базе программного обеспечения CODESYS и программируемого логического контроллера ОВЕН ПЛК -100

Задание: написать программу для автоматического регулирования, обеспечивающую заданный режим термического воздействия при формовании композитных образцов, выдержку в течение установленного времени; программа должна предусматривать построение графика и визуализацию основных параметров процесса: температуры, времени от начала нагрева.

Исходные данные:

начальная температура – 20°С;

Принципиальная схема термообработки приведена на рис. далее

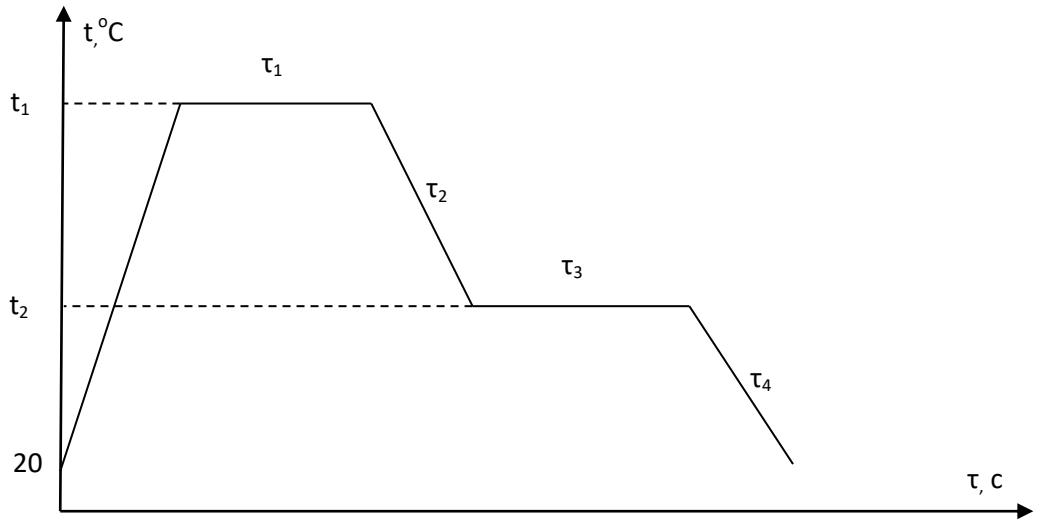
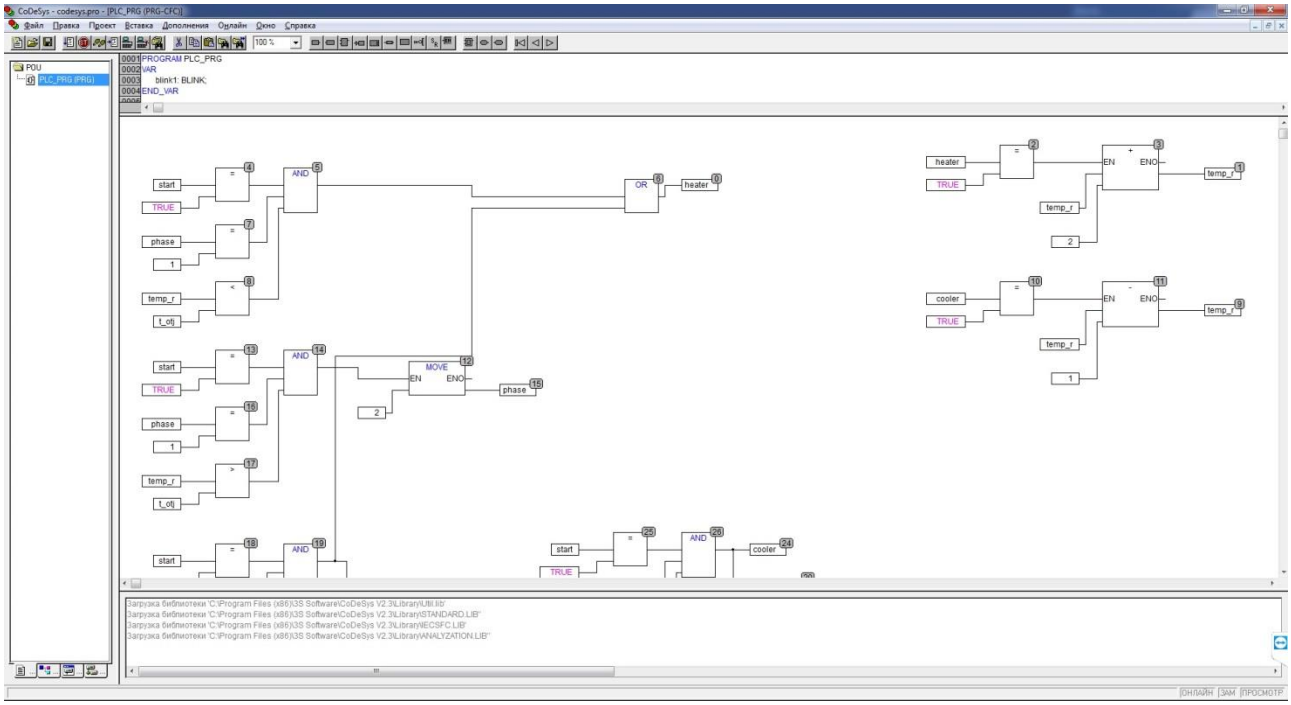
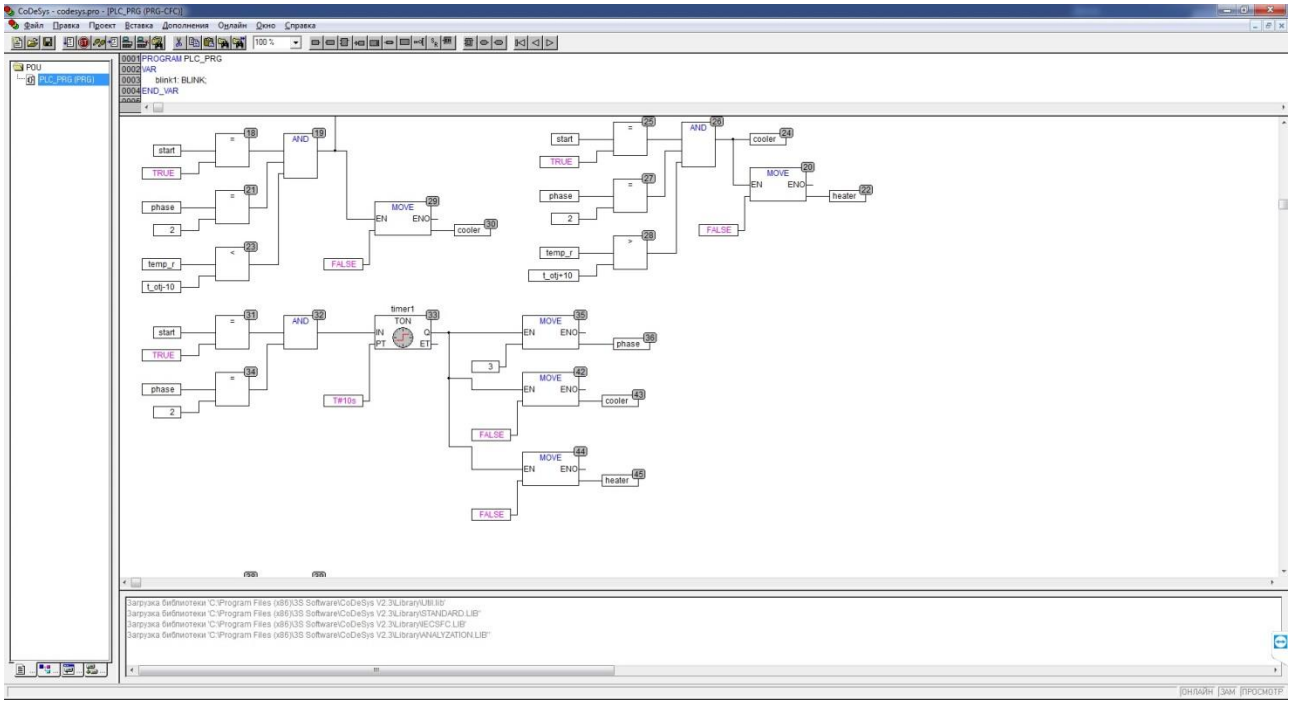


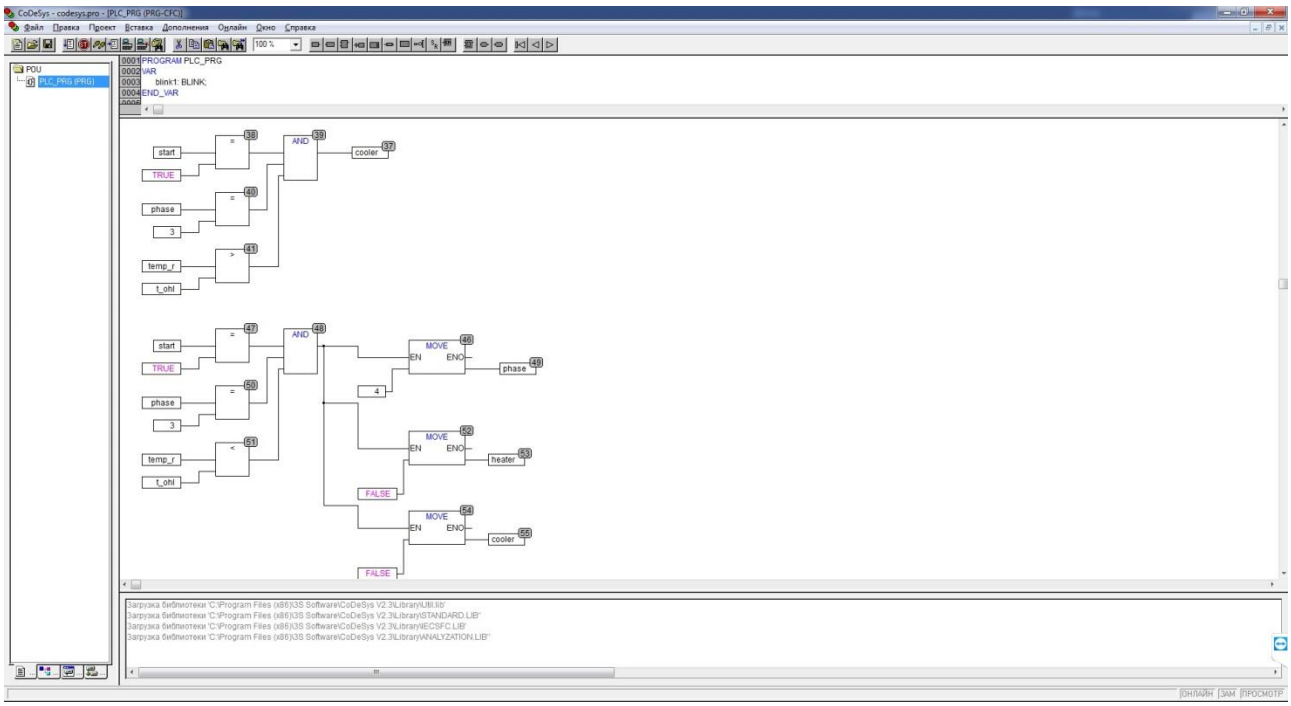
Таблица вариантов исходных данных

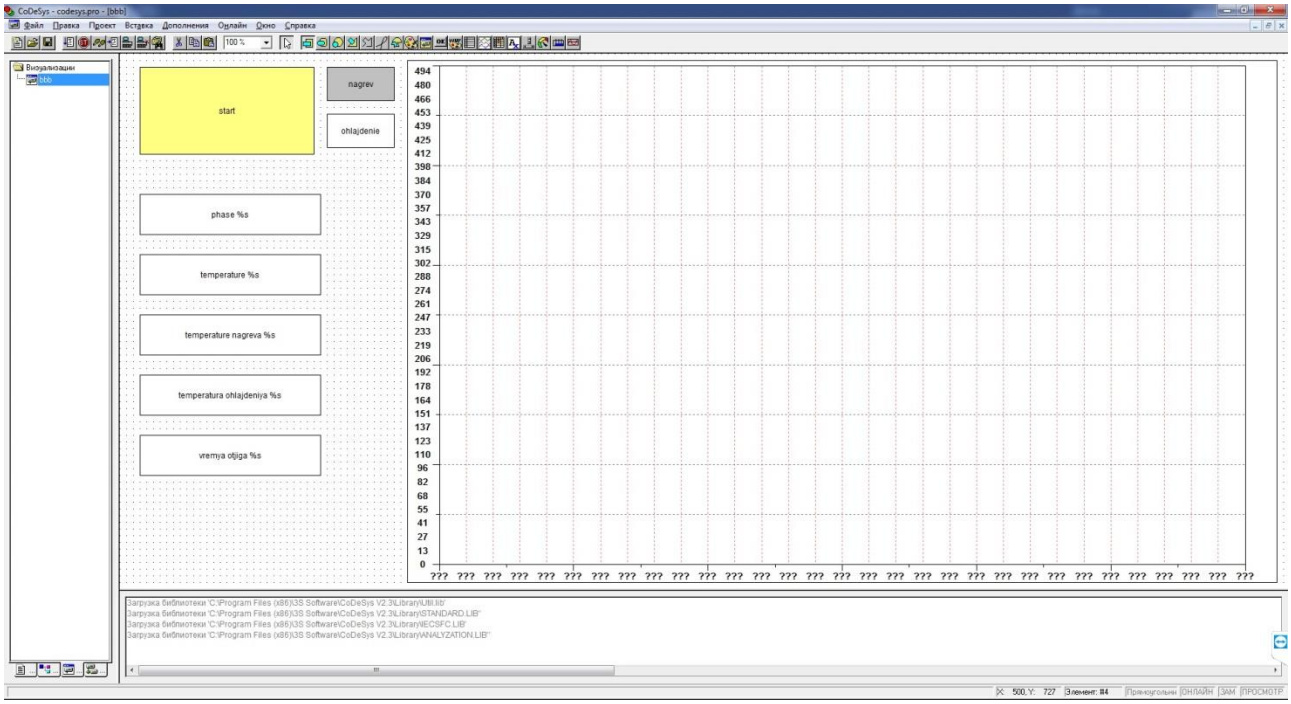
№ варианта	скорость нагрева, °С/сек	скорость охлаждения °С/сек	t_1, °С	t_2, °С	Длительность выдержки τ_1	Длительность выдержки τ_3
1	10	5	250	150	20	30
2	11	6	260	160	30	35
3	9	7	270	170	25	40
4	12	5	240	180	15	45
5	8	6	230	190	20	25
6	10	7	250	200	30	30
7	11	5	260	140	25	35
8	9	6	270	130	15	40
9	12	7	240	120	20	45
10	8	5	230	110	30	25
11	10	6	250	100	25	30
12	11	7	260	150	15	35
13	9	5	270	160	20	40
14	12	6	240	170	30	45
15	8	7	230	180	25	25
16	10	5	250	190	15	30
17	11	6	260	200	20	35
18	9	7	270	140	30	40
19	12	5	240	130	25	45
20	8	6	230	120	15	25
21	10	7	250	110	20	30
22	11	5	260	100	30	35
23	9	6	270	150	25	40
24	12	7	240	160	15	45
25	8	4	230	170	25	25

Пример типовой программы представлен на рис. далее






CoDeSys - codesys.pro - [PLC_PRG (PRG-CFC)]
Файл Правка Проект Вставка Дополнения Онлайн Окно Справка
POU
PLC_PRG (PRG)
0001 PROGRAM PLC_PRG
0002 VAR
0003 blink1: BLINK;
0004 END_VAR
start
TRUE
phase
3
temp_r
t_ohl
AND
cooler
MOVE
EN
ENO
4
heater
FALSE
Загрузка библиотеки "C:\Program Files (x86)\3S Software\CoDeSys V2.3\Library\Util.lib"
Загрузка библиотеки "C:\Program Files (x86)\3S Software\CoDeSys V2.3\Library\STANDARD.LIB"
Загрузка библиотеки "C:\Program Files (x86)\3S Software\CoDeSys V2.3\Library\IECSFC.LIB"
Загрузка библиотеки "C:\Program Files (x86)\3S Software\CoDeSys V2.3\Library\ANALYZATION.LIB"
ОНЛАЙН ЗАМ ПРОСМОТР


CoDeSys - codesys.pro - [bbb]
Файл Правка Проект Вставка Дополнения Онлайн Окно Справка
Визуализации
bbb
start
nagrev
ohlajdenie
phase %s
temperature %s
temperature nagreva %s
temperatura ohlajdeniya %s
vremya otjiga %s
494
480
466
453
439
425
412
398
384
370
357
343
329
315
302
288
274
261
247
233
219
206
192
178
164
151
137
123
110
96
82
68
55
41
27
13
0
Загрузка библиотеки "C:\Program Files (x86)\3S Software\CoDeSys V2.3\Library\Util.lib"
Загрузка библиотеки "C:\Program Files (x86)\3S Software\CoDeSys V2.3\Library\STANDARD.LIB"
Загрузка библиотеки "C:\Program Files (x86)\3S Software\CoDeSys V2.3\Library\IECSFC.LIB"
Загрузка библиотеки "C:\Program Files (x86)\3S Software\CoDeSys V2.3\Library\ANALYZATION.LIB"
ОНЛАЙН ЗАМ ПРОСМОТР