Министерство образования и науки Российской Федерации
ГОУ ВПО "Тамбовский государственный технический университет"

ТЕХНИЧЕСКИЕ СРЕДСТВА АВТОМАТИЗАЦИИ: ПРОГРАММИРОВАНИЕ КОНТРОЛЛЕРОВ В СРЕДЕ ISaGRAF

Лабораторные работы
для студентов 4 курса специальности 220301
"Автоматизация технологических процессов и производств"
по дисциплине "Технологические средства автоматизации"

Тамбов
Издательство ТГТУ
2008

УДК 681.5.08.(075)
ББК ℊ973.26-04я73
Е511

Рекомендовано Редакционно-издательским советом университета

Р е ц е н з е н т
Доктор технических наук, профессор,
заведующий кафедрой "ИС и ЗИ" ТГТУ
Ю.Ю. Громов

С о с т а в и т е л и :
И.А. Елизаров, *А.А. Третьяков*,
В.Н. Назаров, *М.Н. Солуданов*

Е511 Технические средства автоматизации: программирование контроллеров в среде ISaGRAF : лабораторные работы / сост. : И.А. Елизаров, А.А. Третьяков, В.Н. Назаров, М.Н. Солуданов. – Тамбов : Изд-во Тамб. гос. техн. ун-та, 2008. – 24 с. – 75 экз.

Представлены лабораторные работы, в которых даны последовательность выполнения, несколько практических примеров, а также краткие сведения о языках FBD, ST, LD, SFC стандарта IEC 61131-3.

Предназначены для студентов 4 курса специальности 220301 "Автоматизация технологических процессов и производств".

УДК 681.5.08.(075)
ББК ℘973.26-04я73

ТЕХНИЧЕСКИЕ СРЕДСТВА АВТОМАТИЗАЦИИ: ПРОГРАММИРОВАНИЕ КОНТРОЛЛЕРОВ В СРЕДЕ ISaGRAF

♦ИЗДАТЕЛЬСТВО ТГТУ♦

Учебное издание

ТЕХНИЧЕСКИЕ СРЕДСТВА АВТОМАТИЗАЦИИ: ПРОГРАММИРОВАНИЕ КОНТРОЛЛЕРОВ В СРЕДЕ ISaGRAF

Лабораторные работы

Составители:
ЕЛИЗАРОВ Игорь Александрович,
ТРЕТЬЯКОВ Александр Александрович,
НАЗАРОВ Виктор Николаевич,
СОЛУДАНОВ Михаил Николаевич

Редактор Ю.В. Шиманова
Инженеры по компьютерному макетированию:
Н.И. Колмакова, М.А. Филатова

Подписано в печать 10.12.2008.
Формат 60 × 84 / 16. 1,39 усл. печ. л. Тираж 75. Заказ № 557

Издательско-полиграфический центр
Тамбовского государственного технического университета
392000, Тамбов, Советская, 106, к. 14

ОБЩИЕ СВЕДЕНИЯ

Задачи программирования современных промышленных контроллеров достаточно специфичны и сложны и зачастую требуют для их эффективного решения соответствующих инструментальных средств автоматизации программирования.

В настоящее время большинство фирм-производителей программируемых логических контроллеров (ПЛК) снабжают свою продукцию специализированным программным обеспечением для их программирования. Эти инструментальные средства программирования, как правило, поддерживают международный стандарт на языки программирования ПЛК – IEC 61131-3.

Стандарт IEC 61131-3 описывает два компонента: так называемые общие элементы (тип данных, переменные, стандартные функции и блоки и др.) и собственно языки программирования. Языки программирования для контроллеров определяются таким образом, что части прикладной программы могут быть запрограммированы на любом языке и скомпонованы в единую исполняемую программу.

Стандарт IEC 61131-3 описывает синтаксис и семантику пяти языков программирования ПЛК:

1. Язык *последовательных функциональных схем SFC* (Sequential Function Chart) – графический язык, позволяет описать логику программы на основе чередующихся процедурных шагов и условных переходов, а также представить последовательно-параллельные задачи в понятной и наглядной форме. Основными элементами языка являются шаг и переход.

Шаг представляет собой набор операций над переменными. Переход – набор логических условий, определяющий передачу управления к другому шагу.

По внешнему виду описание на языке SFC напоминает хорошо известные логические блок-схемы алгоритмов, при этом SFC предоставляет возможности построения сложных распараллеленных алгоритмов. Язык SFC является наиболее важным из семейства языков стандарта IEC 61131-3. Однако SFC не имеет средств для описания шагов и переходов, и их содержание должно быть выражено средствами других языков стандарта.

2. Язык *функциональных блоковых диаграмм FBD* (Function Block Diagrams) – графический язык, позволяет создать программу практически любой сложности с использованием библиотечных функций (арифметических, тригонометрических, строковых) и функциональных блоков (логических, ПИД-регулирования, мультиплексоров и др.).

Программа на языке FBD выглядит как набор блоков, между входами/ выходами которых графически установлены связи. Программирование сводится к выбору необходимых библиотечных функций и блоков и соединению их соответствующих входов/выходов. В результате получается максимально наглядная и хорошо контролируемая программа.

3. Графический язык *релейных диаграмм* или *релейной логики LD* (Ladder Diagrams) является стандартизованным вариантом класса языков релейно-контактных схем и применяется для описания логических выражений различного уровня сложности. Логические выражения на этом языке описываются в виде контактов и катушек реле, которые широко применялись в области автоматизации в 60 – 70-х гг. XX в. Ввиду своих ограниченных возможностей язык дополнен такими средствами, как таймеры, счётчики и др.

4. Язык *структурированного текста ST* (Structured Text) относится к классу текстовых языков высокого уровня и предоставляет булевые и арифметические операторы, оператор ветвления if-then-else, операторы цикла. По мнемонике язык ST похож на Паскаль. На его основе можно создавать гибкие процедуры обработки данных. Язык структурированного текста является основным для программирования последовательных шагов и переходов языка SFC.

5. *Язык инструкций IL* (Instruction List) – текстовый язык низкого уровня. Выглядит как типичный язык Ассемблера. Язык IL позволяет создавать эффективные, оптимальные по быстродействию программы.

- Одной из самых распространённых инструментальных систем программирования контроллеров, реализующих стандарт IEC 61131-3, является система ISaGRAF. Инструментальная система ISaGRAF относится к классу систем CASE-типа (Computer Aided Software Engineering).

Система ISaGRAF включает в себя систему разработки (ISaGRAF Workbench) и систему исполнения (ISaGRAF Target).

Общая структура системы ISaGRAF представлена на рис. 1.

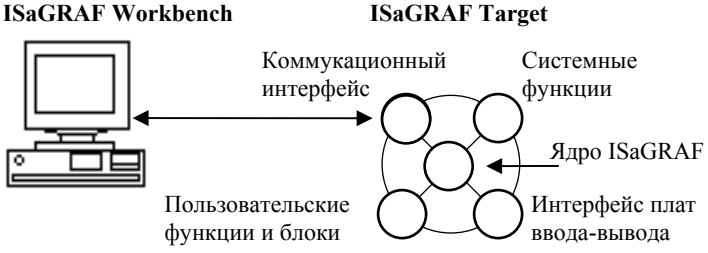


Рис. 1

Система разработки предназначена для создания прикладных задач, исполняемых под управлением ядра ISaGRAF на системах исполнения, и устанавливается на IBM PC – совместимом компьютере под управлением операционной системы семейства MS Windows. Система разработки компилирует проект в системно-независимый код – Target Independent Code (TIC). TIC-код загружается через каналы коммуникаций (RS-232, RS-485, Ethernet TCP/IP) в целевую машину (контроллер) для исполнения.

Система исполнения загружается или прожигается в ПЗУ целевой машины. Она включает в себя ядро ISaGRAF и набор дополнительных модулей.

Ядро ISaGRAF реализует поддержку стандартных языков программирования ПЛК, набора стандартных функций и функциональных блоков.

Коммуникационный интерфейс обеспечивает поддержку со стороны ПЛК процедуры загрузки ISaGRAF-приложения, доступ к переменным этого приложения во время отладки, а также взаимодействие с системами SCADA (Supervisory Control And Data Acquisition – диспетчерское управление и сбор данных) посредством протокола Modbus.

Системные функции предназначены для описания специфики аппаратной части и операционной системы, установленной в конкретном контроллере. ISaGRAF-приложение может функционировать под различными операционными системами (Windows NT, Windows CE, Embedded NT, OS 9/9000, MS DOS, Lynx OS, QNX, VxWorks и др.).

Пользовательские функции и функциональные блоки описывают алгоритмы и процедуры обработки информации и управления, которые не реализованы в стандартном варианте системы ISaGRAF.

Интерфейс плат ввода/вывода предназначен для организации доступа к аппаратуре ввода/вывода (локальным платам ввода/вывода, контроллерам промышленных сетей CAN, Profibus, Ethernet и др.).

Лабораторная работа 1

СОЗДАНИЕ ПРОСТЕЙШЕГО ПРОЕКТА В СРЕДЕ ISaGRAF НА ЯЗЫКЕ FBD

Цель работы: знакомство со средой программирования промышленных контроллеров ISaGRAF версии 3.32 и выше (3.4x, 3.5x) и языком программирования FBD.

Задание на лабораторную работу: в ходе работы разработать приложение на языке FBD для виртуального контроллера, реализующее вычисление следующих арифметических и логических выражений:

$$y_1 = (3x_1 + 2x_2)x_3;$$

$$y_2 = \overline{z_1 \wedge z_2} \vee z_3;$$

$$y_3 = \begin{cases} false, & \text{если } x_4 < 0; \\ true, & \text{если } x_4 > 0; \\ \sqcap\!\sqcup, & \text{если } x_4 = 0, \end{cases}$$

где x_1, x_2, x_3, x_4 – входные действительные переменные; y_1 – выходная действительная переменная; z_1, z_2, z_3 – входные булевы переменные; y_2, y_3 – выходные булевы переменные; $\sqcap\!\sqcup$ – меандр с периодом следования импульсов $T = 1$ с.

Этапы создания приложения в среде ISaGRAF

В общем виде работы по созданию приложений в среде ISaGRAF могут быть разбиты на следующие этапы:

1. Создание проекта.
2. Создание программ.
3. Объявление переменных.
4. Редактирование программы.
5. Конфигурация ввода/вывода.
6. Установка опций приложения и параметров связи с контроллером.
7. Компиляция программ и создание кода приложения.
8. Симуляция и отладка приложения.
9. Загрузка приложения в контроллер.

Дополнительные сведения о FBD

1. FBD-программа построена из стандартных элементарных функциональных блоков из библиотеки ISaGRAF. Каждый функциональный блок имеет фиксированное количество точек входа и выхода. Каждый вход или выход блока имеет определённый тип.

2. Входные и выходные переменные, входы и выходы функциональных блоков соединены линиями связи. Линии могут быть использованы для соединения двух логических точек диаграммы: входной переменной и входа функционального блока; выхода функционального блока и входа другого блока; выхода функционального блока и выходной переменной.

Связи ориентированы, это означает, что данные передаются с левого конца к правому. Левый и правый концы связи должны быть **одного типа**.

3. Использование оператора **RETURN**. Ключевое слово **RETURN** может быть выходом диаграммы. Оно должно быть связано с логическим выходом функционального блока. Оператор **RETURN** представляет собой условное завершение программы: если выход блока, связанного с оператором, имеет тип **TRUE**, остальная часть диаграммы не выполняется.

4. Использование прыжков и меток. Прыжки и метки используются для управления выполнением диаграммы. К правому краю символа метки или прыжка не может быть присоединено никаких других объектов. Используются следующие обозначения:

>>LAB..... прыжок на метку (имя метки "LAB")

LAB:..... определение метки (имя метки "LAB")

Если линия связи слева от символа прыжка находится в состоянии **TRUE**, исполнение программы переходит на соответствующую метку.

5. Логическое отрицание. Одиночная линия связи с правым концом, присоединённым к входу функционального блока, может заканчиваться **логическим отрицанием**. Отрицание представляется маленьким колечком. Когда используется логическое отрицание, левый и правый концы линии связи должны иметь тип **BOOLEAN.**

Порядок выполнения работы

Создание проекта. Создайте проект (под названием "Project1"), используя команду "New" из меню "File" или кнопку . В диалоговом окне:

- Введите имя проекта "Project1".
- Выберите конфигурацию ввода/вывода "None".
- Нажмите кнопку "Ok".

Открытие проекта. Программы проекта появляются при открытии окна Менеджера Программ ISaGRAF. Для перехода в окно Менеджера Программ щёлкните два раза мышью на имени нужного проекта или воспользуйтесь кнопкой .

Создание программ. Окно Менеджера Программ сейчас открыто и пусто (так как ни одна программа не определена). Первая программа создаётся при помощи команды "New" меню "File" или кнопки . В окне диалога:

- Введите имя программы "Program1".
- Выберите язык "FBD".
- Выберите раздел "Beginning of cycle".
- Нажмите кнопку "Ok" для создания программы.

Объявление переменных. Перед вводом программы должны быть объявлены переменные, используемые в данной программе. Это делается при помощи команды "Dictionary" меню "File" или кнопки .

Диалоговое окно "Dictionary" (словарь) имеет несколько закладок: "booleans", "Integer/Reals", "Timers", "Messages", "FB instances", "Defined words", в которых соответственно описываются булевы, целые и действительные, таймерные переменные, сообщения, экземпляры функциональных блоков, макроопределения.

Создайте все входные и выходные аналоговые (действительные или целые) и булевы переменные. Для задания значения таймерной переменной (в данном задании – период следования импульсов *T*) в поле ввода значения необходимо вначале ввести префикс *t#*, а вслед за ним величину времени, например: *t#*1s (1 секунда), *t#*1m (1 минута), *t#*500ms (500 милисекунд).

В завершение, покиньте редактор словаря, сохранив изменения.

Редактирование программ. Эта команда позволяет изменить содержание программы. Используемый редактор зависит от языка, выбранного для написания программы.

В окне редактирования программы произведите набор программы в соответствии с рис. 1.1.

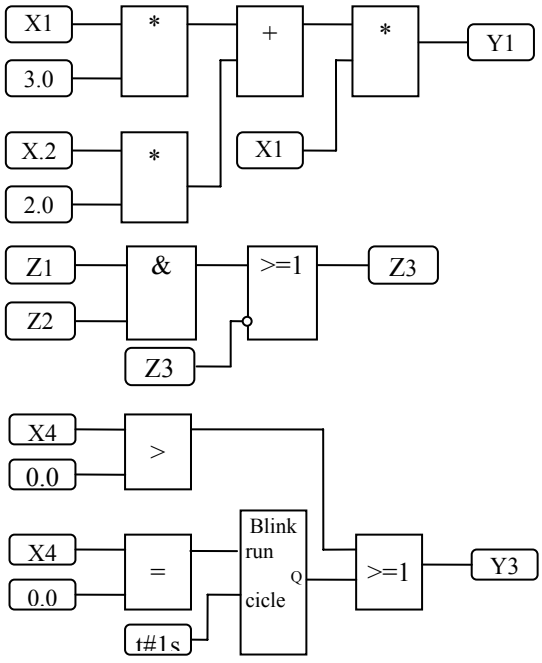


Рис. 1.1

***Привязка переменных ввода/вывода* (*конфигурация ввода/вывода*).** Команда **"**I/O connection**"** запускает редактор соединения переменных ISaGRAF. Этот инструмент используется для создания связей между объявленными в словаре проекта переменными ввода/вывода и соответствующей аппаратурой. Для данного задания выберите виртуальные платы ввода/вывода **xai8**, **xao8**, **xbi8**, **xbo8** и осуществите привязку переменных ввода/вывода, объявленных в вашей программе и входами/выходами плат.

Создание кода приложения. Для создания кода используйте команду "Make application" меню "Make" из окна Менеджера Программ или кнопку .

Симуляция. Для запуска ядра симулятора ISaGRAF используйте команду "Simulate" меню "Debug" из окна Менеджера Программ или кнопку . При появлении окна симулятора приложение может быть протестировано. Для выхода из симулятора закройте окно отладчика: меню "File" команда "Exit".

Контрольные задания

1. Разработать приложение на языке FBD, реализующее вычисление следующих арифметических и логических выражений (табл. 1.1).

Таблица 1.1

№	Выражения	№	Выражения
1	$y(x_1,x_2,x_3)=4{,}5x_1+1{,}2x_2x_3-0{,}5x_3$	5	$y(x_1,x_2,x_3)=\dfrac{x_1-1{,}7x_2}{x_3^2+1}$
2	$y(x_1,x_2,x_3)=\begin{cases}\lvert x_1+x_2\rvert, & \text{если} x_3<0;\\ -\lvert x_1+x_2\rvert, & \text{если} x_3>0;\\ 0, & \text{если} x_3=0\end{cases}$	6	$y(x_1,x_2)=\int\left(x_1+x_2^2\right)dt$
3	$y(x_1,x_2,x_3)=\begin{cases}x_1-x_2, & \text{если} x_2<x_3;\\ x_1+x_2, & \text{если} x_2>x_3;\\ \sin(x_1), & \text{если} x_2=x_3\end{cases}$	7	$y(x_1)=\begin{cases}1, & \text{если } x_1<0;\\ 0, & \text{если } x_1=0;\\ -1, & \text{если } x_1>0\end{cases}$
4	$y(x_1,x_2)=\begin{cases}x_2, & \text{если } x_1+x_2<1;\\ -1, & \text{если } x_1+x_2=1;\\ x_1, & \text{если } x_1+x_2>1\end{cases}$	8	$y(x_1,x_2,x_3)=\dfrac{2x_1+x_2}{x_3^2+5}$

2. Разработать приложение на языке FBD, реализующее звено со статической характеристикой, показанной на рис. 1.2.

3. Разработать приложение на языке FBD, реализующее:
- Стандартный ПИ регулятор.
- Стандартный ПД регулятор.
- Стандартный ПИД регулятор.

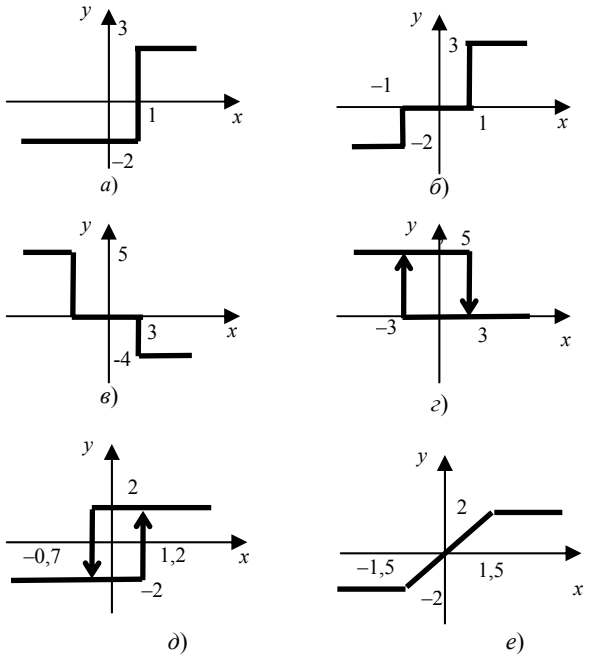


Рис. 1.2

СОЗДАНИЕ ПРОЕКТА В СРЕДЕ ISaGRAF НА ЯЗЫКЕ LD

Цель работы: знакомство с языком программирования LD.

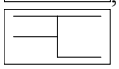
Задание на лабораторную работу: в ходе работы разработать приложение на языке LD для виртуального контроллера, реализующее вычисление арифметических и логических выражений, представленных в задании на лабораторную работу 1.

Краткие сведения о языке LD

Язык релейных диаграмм (LD) – это графическое представление логических уравнений, комбинирующее **контакты** (входы) и **витки** (выходы). Язык LD позволяет описывать работу с **булевыми** данными, помещая **графические символы** в схему программы. Графические символы LD организованы внутри схемы так же, как электрическая схема. Справа и слева LD диаграмма должна соединяться с вертикальными силовыми шинами. Основные компоненты LD диаграммы:


– левая и правая вертикальные силовые шины;


– горизонтальная и вертикальная линии связи;


– множественные линии соединения;


– контакт и виток, связанные с переменными.

Для представления контактов используются символы:

- Прямой контакт ⊣ ⊢
- Инвертированный контакт ⊣/⊢
- Контакт с определением переднего ⊣P⊢ и заднего ⊣N⊢ фронтов.

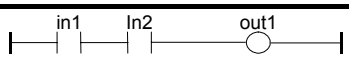
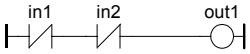
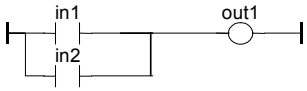
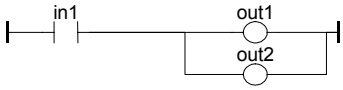
Для представления витков используются символы:

- Прямой виток —○—
- Инвертированный виток —⊘—
- SET виток —Ⓢ—
- RESET виток —Ⓡ—
- Виток с определением фронтов —Ⓟ—, —Ⓝ—

Имя переменной пишется над этими графическими символами.

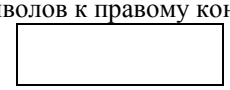
Примеры использования компонентов LD диаграммы приведены в табл. 2.1.

Таблица 2.1

Математическая запись	Программная реализация
$out1 = in1 \wedge in2$	
$out1 = \overline{in1} \wedge \overline{in2}$	
$out1 = in1 \vee in2$	
$out1 = in1$ $out2 = in1$	

В языке LD может быть использован оператор RETURN, а также метки и безусловные переходы.

Метка RETURN может быть использована как выход, чтобы представить условное завершение программы. Никаких символов к правому концу RETURN подключать нельзя.



В редакторе LD можно подключать функциональные блоки к логическим линиям. Так как блоки не всегда имеют логические входы и/или логические выходы, введение блоков в LD диаграммы приводит к добавлению нескольких новых параметров EN, ENO в интерфейс блока (рис. 2.1).

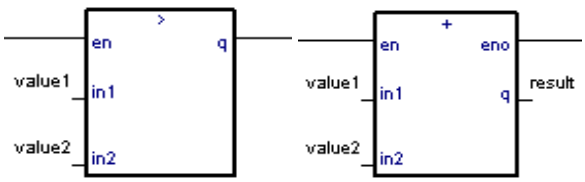


Рис. 2.1

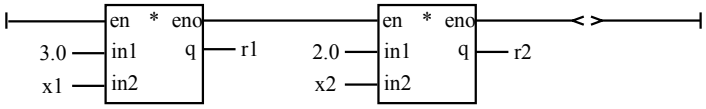
Так как первый выход всегда должен быть подключен к шине, на первую позицию автоматически вводится другой выход, называемый "ENO". Выход ENO всегда имеет то же значение, что и первый вход блока.

Так как первый вход всегда должен быть подключён к шине, на первую позицию автоматически вводится другой вход, называемый "EN". Блок выполняется только тогда, когда вход EN равен TRUE.

Порядок выполнения работы

1. Создать новый проект.
2. Создать новую программу. При выборе языка указать язык LD.
3. Объявить используемые переменные.
4. Отредактировать программу в соответствии с рис. 2.2.

(***задание 1***)



(* *)

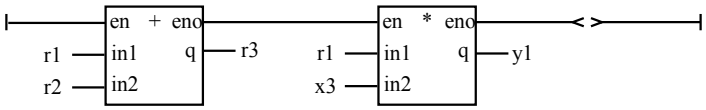
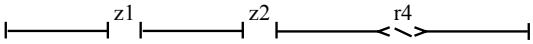
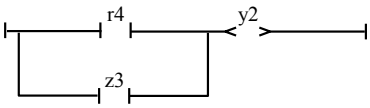


Рис. 2.2

(***задание 2***)



(* *)



(*задание 3*)

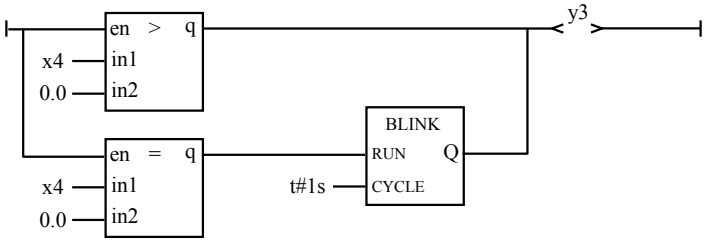


Рис. 2.2. Продолжение

5. Настроить конфигурацию ввода/вывода и осуществить привязку входных и выходных переменных проекта.
6. Создать код приложения.
7. Провести отладку приложения в режиме симуляции

Выполнение п. 1 – 3, 5 – 7 подробно рассмотрено в лабораторной работе 1.

Контрольные задания

Разработать приложение на языке LD, реализующее вычисление следующих логических выражений:

1) $f(x_1, x_2, x_3) = (x_3 \vee x_1) \wedge (x_1 \vee \bar{x}_2) \wedge (\bar{x}_2 \vee \bar{x}_3)$;
2) $f(x_1, x_2, x_3) = (\bar{x}_1 \vee x_3) \wedge (x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2)$;
3) $f(x_1, x_2, x_3) = \overline{\bar{x}_1 \wedge x_3} \vee x_2 \wedge \bar{x}_3 \wedge (\bar{x}_1 \vee x_2)$;
4) $f(x_1, x_2, x_3) = \overline{(\bar{x}_1 \vee x_2)} \wedge (x_2 \vee \bar{x}_3) \vee \overline{(\bar{x}_1 \vee x_2)}$;
5) $f(x_1, x_2, x_3) = \overline{(x_1 \vee \bar{x}_2)} \wedge (x_2 \vee \bar{x}_3) \vee \bar{x}_1 \wedge \bar{x}_3$;
6) $f(x_1, x_2, x_3) = \overline{\overline{(\bar{x}_1 \vee x_2)} \wedge (\bar{x}_1 \vee \bar{x}_3)} \vee \bar{x}_1 \wedge \bar{x}_3$;
7) $f(x_1, x_2, x_3) = x_3 \wedge x_1 \vee x_1 \wedge \bar{x}_2) \vee \overline{(\bar{x}_2 \vee x_3)}$;
8) $f(x_1, x_2, x_3) = (\bar{x}_3 \vee x_1) \wedge \overline{(x_1 \vee \bar{x}_2)} \vee \bar{x}_2 \wedge x_3$.

СОЗДАНИЕ ПРОЕКТА В СРЕДЕ ISaGRAF НА ЯЗЫКЕ ST

Цель работы: знакомство с языком программирования ST.

Задание на лабораторную работу: в ходе работы разработать приложение на языке ST для виртуального контроллера, реализующее вычисление арифметических и логических выражений, представленных в задании на лабораторную работу 1.

Краткие сведения о языке ST

ST (Structured text) – это структурный язык высокого уровня. Этот язык, в основном, используется для создания сложных процедур, которые не могут быть легко выражены при помощи графических языков. По умолчанию ST является языком для описания действий внутри шагов и условий языка **SFC.**

ST-программа – это список ST-**операторов**. Каждый оператор заканчивается точкой с запятой (;). Основные операторы языка ST: оператор **присвоения** (variable := expression); вызов **подпрограммы** или **функции**;
вызов **функционального блока**; операторы **выбора** (IF, THEN, ELSE, CASE); **итеративные** операторы (FOR, WHILE, REPEAT); **управляющие** операторы (RETURN, EXIT); специальные операторы для связи с такими языками, как **SFC.**

Вызов функций в ST осуществляется в соответствии со следующим синтаксисом:

<rezult>:=<function>(<par1>, <par2>,…);

где **<rezult>** – имя переменной, которой присваивается результат выполнения функции; **<function>** – имя вызываемой функции; **<par1>, <par2>,…** – список операндов. Например:

```
y := sin(x1);
z := real(100*x2);
```

Использование в ST функциональных блоков производится в соответствии со следующим синтаксисом:

```
<blockname> ( <p1>, <p2> ... );
<result_1> := <blockname>.<ret_param1>;
... <result_N> := <blockname>.<ret_paramN>;
```

где **<blockname>** – имя экземпляра вызываемого функционального блока (перед использованием экземпляр функционального блока должен быть определён в словаре); **<result_1>,** … **<result_N>** – переменные, которым присваиваются результаты выполнения блока; **<par1>, <par2>,…** – список операндов; **>; <ret_param1>,** …**,<ret_paramN>** – выходные поля функционального блока. Например:

```
blink1(run, t#2s);
y := blink1.q;
```

Здесь **blink1** – экземпляр функционального блока **blink**; **q** – выходное поле блока **blink** (наименование выходных полей следует смотреть в описании функционального блока).

Оператор **IF** предназначен для организации ветвлений, при этом выполняются 1 или 2 списка ST-операторов. Выбор осуществляется в соответствии со значением булевского выражения. Синтаксис оператора IF имеет вид:

```
IF < boolean_expression > THEN
        <statement > ;
        <statement > ;
        …
ELSIF <boolean_expression> THEN
        <statement> ;
        <statement> ;
        …
ELSE
        <statement> ;
        <statement> ;
        …
END_IF;
```

Операторы ELSE и ELSIF – дополнительные. Если ELSE опущен и условие равно FALSE, то никаких инструкций не выполняется.

Оператор **CASE** выполняет один или несколько списков ST-операторов, выбор осуществляется в соответствии с целым выражением. Синтаксис оператора **CASE** имеет вид:

```
CASE <integer_expression> OF
    <value>: <statements> ;
    <value> , <value>: <statements> ;
    …
ELSE
    <statements> ;
END_CASE;
```

Порядок выполнения работы

1. Создать новый проект.
2. Создать новую программу. При выборе языка указать язык ST.

3. Объявить используемые переменные и объявить экземпляр функционального блока **blink**, например присвоить имя экземпляру **blink1**.

4. Отредактировать программу в соответствии с представленным ниже текстом:

```
y1:=(3.0*x1+2.0*x2)*x3;
y2:=NOT(z1 AND z2) OR z3;
IF x4<0.0 THEN y3:=false;
ELSIF x4>0.0 THEN y3:=true;
ELSE
blink1(x4=0.0, t#1s);
y3:= blink1.q;
END_IF;
```

5. Настроить конфигурацию ввода/вывода и осуществить привязку входных и выходных переменных проекта.
6. Создать код приложения.
7. Провести отладку приложения в режиме симуляции.

Выполнение п.1 – 3, 5 – 7 подробно рассмотрено в лабораторной работе 1.

Контрольные задания

1. Разработать приложение на языке ST, реализующее вычисление арифметических и логических выражений, представленных в табл. 1.1.

2. Разработать приложение на языке ST, реализующее нелинейное звено со статической характеристикой, представленной на рис. 1.2:

3. Разработать приложение на языке ST, реализующее:

- Стандартный ПИ регулятор.
- Апериодическое звено первого порядка.
- Стандартный ПИД регулятор.
- Апериодическое звено второго порядка.
- Колебательное звено.
- Интегрирующее звено.
- Реально-дифференцирующее звено.
- Интегро-дифференцирующее звено.

Лабораторная работа 4

СОЗДАНИЕ ПРОЕКТА В СРЕДЕ ISaGRAF НА ЯЗЫКЕ SFC

Цель работы: знакомство с языком программирования SFC.

Задание на лабораторную работу: в ходе работы разработать приложение на языке SFC для виртуального контроллера, реализующее нижеприведённую последовательность действий:

1. При нажатии кнопки "ПУСК" начать процесс, при этом открыть клапан *A* и загрузить компонент *A* в течение 10 с в ёмкость *1*.
2. По окончании загрузки компонента *A*, при работающей мешалке *M* начать загрузку компонента *B*, открыв клапан *B*.
3. Загрузку осуществить в течение 15 с.
4. По окончании загрузки компонента *B* смесь продолжить перемешивать ещё в течение 10 с.
5. Затем смесь перекачать в ёмкость *2*, включив насос H1. Об опорожнении ёмкости *1* свидетельствует срабатывание датчика реле уровня LS1.
6. В ёмкость *2* произвести загрузку компонента *C* в течение 10 с, открыв клапан *C*.
7. Дать смеси выдержаться в течение 15 с, после чего открыть клапан *D* и выгрузить готовый продукт из ёмкости *2*. Об опорожнении ёмкости *2* свидетельствует срабатывание датчика реле уровня LS2.
8. Подготовить линию для приготовления новой партии продукта.

Краткие сведения о языке SFC

Язык **SFC** (Sequentiad Function Chart) – это **графический** язык, который используется для описания **последовательных операций**. Процесс представляется в виде набора определённых шагов, связанных **переходами**. К каждому переходу прикреплено **логическое условие**. **Действия** внутри шагов описаны более детально при помощи других языков (как правило – ST).

SFC программа – это графический набор **шагов** и **переходов**, соединённых вместе направленными связями. Для обозначения схождений и расхождений используются множественные связи. Некоторые части программы могут быть отделены и представлены в основной схеме одним символом – **макрошагом**. Вот основные **графические правила** для SFC:

1. Шаги не могут следовать подряд.
2. Переходы не могут следовать подряд.

Основные компоненты SFC:

 .....Начальный шаг

.....Шаг

 .....Переход

↓.....Прыжок на шаг

.....Макрошаг

.....Начальный макрошаг

.....Конечный макрошаг

Шаг представляется одиночным **квадратом**. Каждому шагу присваивается номер, написанный внутри квадрата. Основное описание шага пишется внутри прямоугольника, присоединённого к символу шага. Это **свободный комментарий,** который не является частью языка. Вышеприведённая информация называется **Уровнем 1** шага (рис. 4.1).

Во время работы **активный** шаг помечается маркером (выделяется).

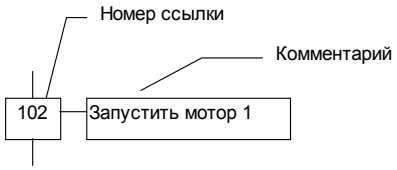


Рис. 4.1

Начальная ситуация программы SFC описывается **начальными шагами**. Начальный шаг обозначается графическим символом **с двойной рамкой**. После запуска программы маркер автоматически устанавливается на каждый начальный шаг (рис. 4.2).

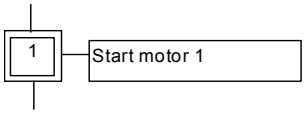


Рис. 4.2

У каждого шага есть атрибуты. Они могут быть использованы в любом другом языке в любом месте программы:

GSnnn.x...активность шага (логическая переменная);

GSnnn.t...продолжительность активного состояния шага (таймер), где **nnn** – номер шага.

Переходы представлены горизонтальными полосками (┼), которые пересекают линии связи. Каждому переходу присвоен номер, следующий за символом перехода. Описание перехода (комментарий) располагается справа от символа перехода. Вышеприведённая информация называется **Уровнем 1** перехода (рис. 4.3).

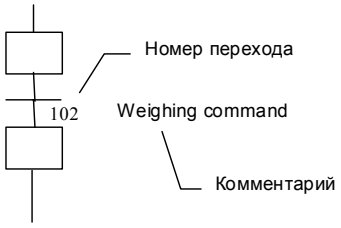


Рис. 4.3

Для связи шагов и переходов используются одиночные линии. Это ориентированные связи. Когда ориентация не задана явно, связь ориентирована сверху вниз (рис. 4.4).

Символ прыжка ↓ может быть использован, чтобы определить линию связи от перехода к шагу, не рисуя линию. Символ прыжка должен иметь номер шага назначения (рис. 4.5).

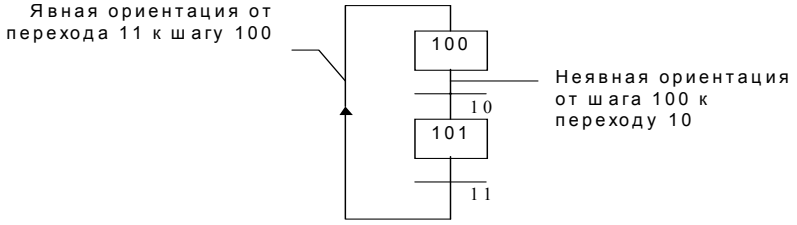


Рис. 4.4

Связь от шага к переходу нельзя представить с помощью символа прыжка.

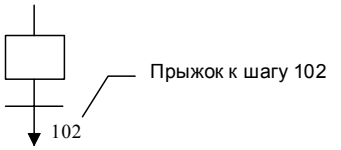


Рис. 4.5

Расхождения – это **множественные связи** от одного символа SFC (шага или перехода) ко многим. **Схождение** – это множественные связи от более чем одного символа SFC к одному другому символу. Схождения и расхождения могут быть одиночными или двойными.

Одиночное расхождение – это множественная связь от одного шага к нескольким переходам. Оно позволяет маркеру активности пройти по одной из множества ветвей. Одиночное схождение – это множественная связь от нескольких переходов к одному и тому же шагу. Одиночное схождение, обычно используется для того, чтобы объединить несколько ветвей SFC, начавшихся из одиночного расхождения. Одиночные расхождения и схождения обозначаются одиночными горизонтальными линиями (рис. 4.6).

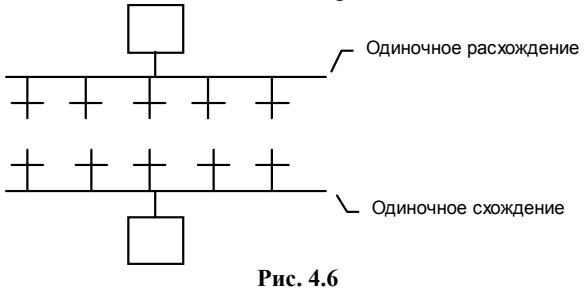


Рис. 4.6

Следует обратить внимание, что условия, присоединённые к переходам, не являются **взаимоисключающими**. Для того, чтобы программа пошла по одной ветке, надо явно определить исключительность условий перехода.

Двойное расхождение – это множественная связь от одного перехода к нескольким шагам. Она соответствует параллельной работе процесса. Двойное схождение – это множественная связь от нескольких шагов к одному и тому же пере-

ходу. Двойное схождение, обычно, используется для того, чтобы объединить несколько ветвей SFC, начавшихся из двойного расхождения. Двойные расхождения и схождения обозначаются двойными горизонтальными линиями (рис. 4.7).

Уровень 2 шага SFC представляет собой детальное описание **действий** в период **активности шага**. Это описание может использовать **текстовые дополнения языка SFC** или язык **ST**. Основные типы действий: булевские действия; импульсные действия, описанные на ST (действия типа "P"); не сохраняемые действия, описанные на ST (действия типа "N"); SFC действия.

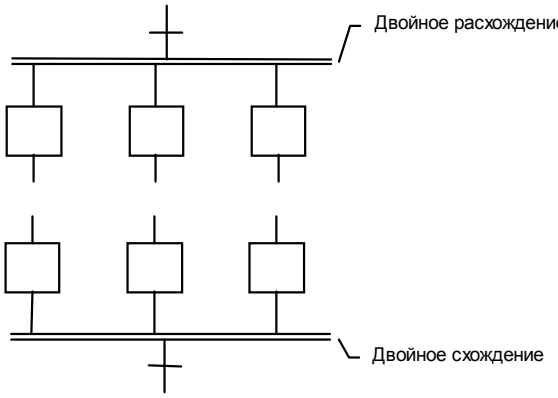


Рис. 4.7

Булевские действия присваивают значение логической переменной при активизации шага. Логические переменные могут быть выходными или внутренними. Им присваивается значение каждый раз, когда шаг становится активным или перестаёт быть активным. Синтаксис основных логических действий показан в табл. 4.1.

Таблица 4.1

Логическое действие	Описание
<boolean_variable> (N) ; **<boolean_variable> ;**	Присвоить переменной сигнал активности шага
/ <boolean_variable> ;	Присвоить переменной отрицание сигнала активности шага
<boolean_variable> (S) ;	Присваивает переменной значение TRUE, когда шаг становится активным
<boolean_variable> (R) ;	Присваивает переменной значение FALSE, когда шаг становится активным

Импульсное действие – это список инструкций ST или IL, которые выполняются только **однажды** при **активизации** шага. Инструкции пишутся в соответствии с синтаксисом, показанным на рис. 4.8.

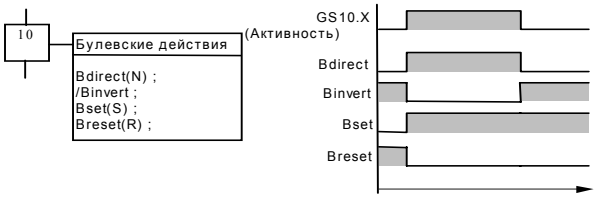


Рис. 4.8

ACTION (P) :
(*ST операторы *)
END_ACTION ;

Не сохраняемое действие – это список инструкций ST или IL, которые выполняются **на каждом цикле** в течение всего периода активности шага. Инструкции пишутся в соответствии со следующим синтаксисом:

ACTION (N) :
(* ST операторы *)
END_ACTION ;

SFC-действие – это дочерняя последовательность SFC, стартующая и убивающаяся в соответствии с изменением сигнала активности шага. SFC-действие может иметь признак **N** (не запоминаемый), **R** (установить), **S** (сбросить). Синтаксис основных SFC действий показан в табл. 4.2.

Таблица 4.2

SFC действие	Описание
<child_prog> (N); **<child_prog> ;**	Запустить дочернюю последовательность, когда шаг становится активным, и убить ее, когда шаг становится пассивным
<child_prog> (S);	Запустить дочернюю последовательность, когда шаг становится активным, и ничего не делать, когда шаг становится пассивным

<child_prog> (R);	Убить дочернюю последовательность, когда шаг становится активным, и ничего не делать, когда шаг становится пассивным

SFC-последовательность, определённая как действие, должна быть **дочерней SFC программой** редактируемой программы.

Порядок выполнения работы

1. Создать новый проект.
2. Создать новую программу. При выборе языка указать язык SFC.
3. Объявить используемые переменные.
4. Отредактировать программу в соответствии с рис. 4.9.
5. Настроить конфигурацию ввода/вывода и осуществить привязку входных и выходных переменных проекта.
6. Создать код приложения.
7. Провести отладку приложения в режиме симуляции.

Выполнение п. 1 – 3, 5 – 7 подробно рассмотрено в лабораторной работе 1.

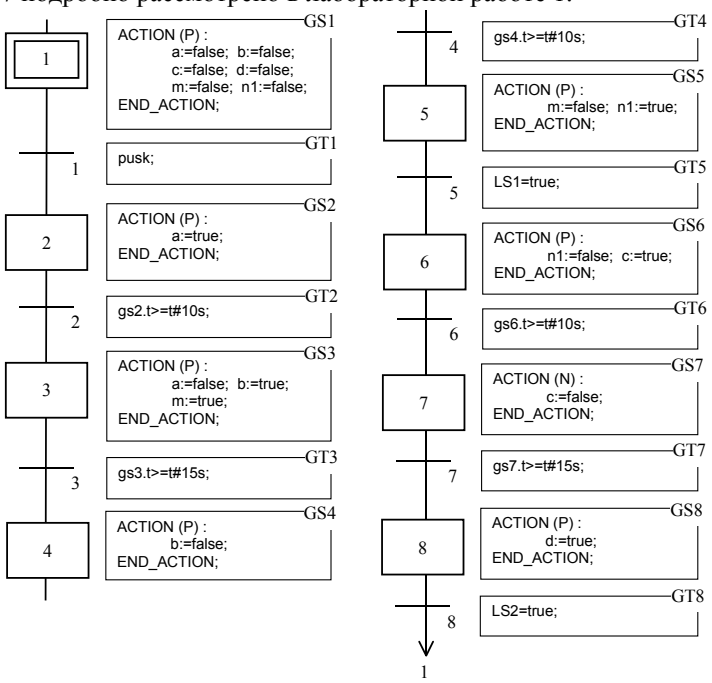


Рис. 4.9

Контрольные задания

1. Разработать приложение на языке SFC, реализующее следующий алгоритм управления.

В ёмкость залить вещество *A* до уровня 1,5 м. Включить мешалку. Затем залить вещество *C* до уровня 2,6 м. Выдержать смесь в аппарате в течение 1 часа. Выключить мешалку и слить полученную смесь *B* (рис. 4.10).

2. Необходимо реализовать одноконтурную АСР температуры в реакторе (рис. 4.11, *а*).

При этом необходимо загружать вещество *A* в течение 5 минут, затем включить мешалку и поддерживать температуру по заданной программе (рис. 4.11, *б*).

3. Необходимо реализовать одноконтурную АСР температуры в реакторе (рис. 4.12, *а*).

Открыть клапан *A* на 30 с. Затем включить мешалку и поддерживать температуру по заданной программе в течение 1 часа (рис. 4.12, *б*). Далее открыть клапан *B*.

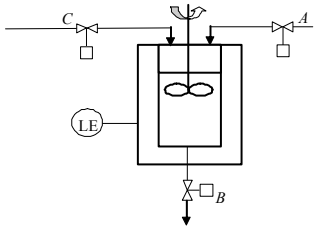
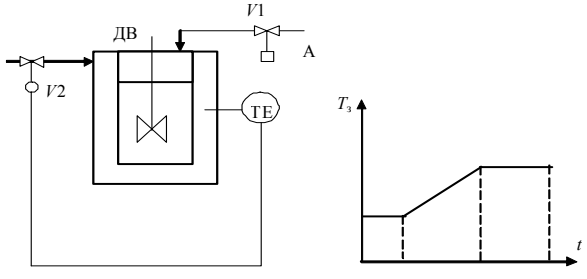


Рис. 4.10



а) *б*)

Рис. 4.11

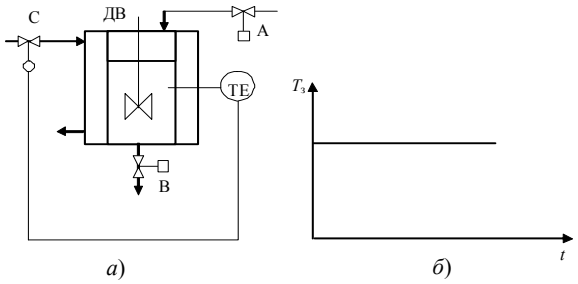


Рис. 4.12

4. Разработать приложение на языке SFC, реализующее следующий алгоритм управления темперирующей машиной (рис. 4.13).

Залить компоненты в аппарат. Включить мешалку, нагреть массу до 60 °С. При этой температуре вести обработку 4 часа. После этого выключить мешалку и перекачать массу.

5. Разработать приложение на языке SFC, реализующее следующий алгоритм управления. Предварительно аппарат *A* (рис. 4.14) просушить в течение 10 мин сжатым воздухом. Затем закачать реагент, включить мешалку, нагреть до температуры 80 °С, выключить мешалку, перекачать в аппарат *B*. Одновременно с просушкой аппарата *A* в аппарат *B* закачать воду и включить мешалку. Перед закачкой реагента в аппарат *B* из него слить воду.

6. Разработать приложение на языке SFC, реализующее следующий алгоритм управления.

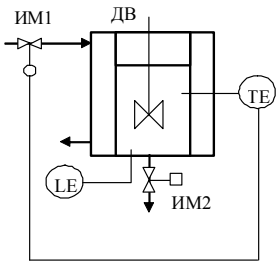


Рис. 4.13

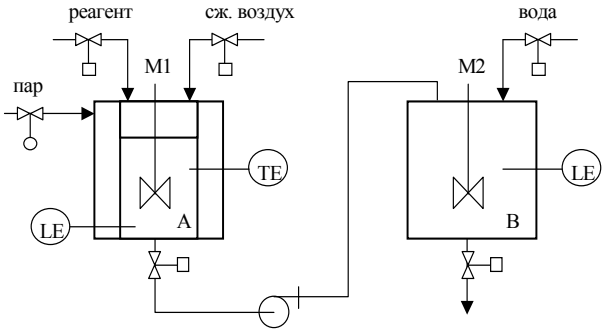


Рис. 4.14

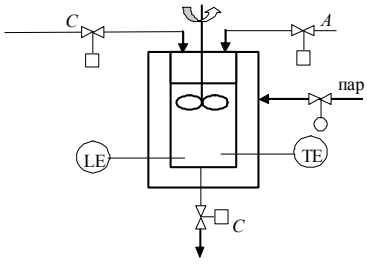


Рис. 4.15

В ёмкость загрузить вещество *A* (рис. 4.15) до уровня 1,5 м. Включить мешалку. Нагреть вещество до 70 °С. Затем загрузить вещество *B* до уровня 2,6 м. Выдержать смесь в аппарате в течение 1 часа при температуре 70 °С. Выключить мешалку и слить полученную смесь *C*.

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. ISaGRAF: версия 3.4. Руководство пользователя. CJ International, 2003. – 430 с.

2. Технические средства автоматизации. Программно-технические комплексы и контроллеры / И.А. Елизаров, Ю.Ф. Мартемьянов, А.Г. Схиртладзе, С.В. Фролов. – М. : Изд-во Машиностроение-1, 2004. – 180 с.