Лабораторная работа №_

Программа управления кодовым замком для контроллера ОВЕН ПЛК 150

Цель работы: знакомство с языком программирования стандарта МЭК 61131-3 ST (структурированный текст), разработка программы управления с визуализацией в реальном времени.

Перечень оборудования и приборов

1. Программируемый логический контроллер ПЛК-150 – 1 шт.
2. Персональный компьютер – 1 шт.

Порядок выполнения лабораторной работы

Язык программирования ST. ST представляет собой набор инструкций высокого уровня, которые могут использоваться в условных операторах (IF…THEN…ELSE) и в циклах (WHILE…DO).

Пример:

IF value < 7 THEN

WHILE value < 8 DO

value: = value + 1;

END_WHILE;

END_IF;

Выражение – это конструкция, возвращающая определенное значение после его вычисления. Выражение состоит из операторов и операндов. Операндом может быть константа, переменная, функциональный блок или другое выражение.

Вычисление выражений выполняется согласно правилам приоритета. Оператор с самым высоким приоритетом выполняется первым, оператор с более низким приоритетом – вторым и т. д., пока не будут выполнены все операторы.

Операторы с одинаковым приоритетом выполняются слева направо. Список ST операторов, расположенных в порядке приоритета, приведен в табл. 2.

Перед *оператором присваивания* находится операнд (переменная или адрес), которому присваивается значение выражения, стоящего после оператора присваивания:

Var1: = Var2 * 10;

Таблица 2

Список ST-операторов

Операция	Обозначение	Приоритет
Выражение в скобках	(Выражение)	Самый высокий
Вызов функции (список параметров)	Имя функции	
Возведение в степень	EXPT	
Замена знаков	–	
Числовое дополнение	NOT	
Умножение	*	
Деление	/	
Абсолютная величина	MOD	
Сложение	+	
Вычитание	–	
Сравнение	< , > ,< = , > =	
Неравенство	< >	
Равенство	=	
Логическое И	AND	
Логическое исключающее ИЛИ	XOR	
Логическое ИЛИ	OR	Самый низкий

После выполнения этой операции Var1 принимает значение в десять раз большее, чем Var2.

Функциональный блок вызывается с помощью имени экземпляра функционального блока и списка входных параметров с присваиванием данных в круглых скобках. В следующем примере вызывается таймер с параметрами IN и PT.

Значение выходной переменной Q присваивается переменной A. К выходной переменной можно обратиться с помощью имени экземпляра функционального блока, точки, следующей за ним и имени выходной переменной:

CMD_TMR (IN: = %IX5, PT: = 300);

A: = CMD_TMR.Q

Инструкция RETURN позволяет выйти из POU, например, в зависимости от условия. Используя инструкцию IF, можно проверить условие, и в зависимости от этого условия выполнить какие-либо действия.

Синтаксис:

IF <Boolean_expression1> THEN

<IF_instructions>

{ELSIF <Boolean_expression2> THEN

<ELSIF_instructions1>

.

.

.ELSIF <Boolean_expression n> THEN

<ELSIF_instructions n-1>

ELSE

<ELSE_instructions>}

END_IF.

Часть конструкции в фигурных скобках не обязательна.

Если <Boolean_expression1> возвращает истину, тогда <IF_Instructions> выполняется.

В противном случае будут выполняться остальные логические выражения одно за другим, пока одно из них не возвратит истину. Тогда выполняются инструкции, стоящие после этого логического выражения до следующего ELSIF или ELSE. Если все логические выражения ложны, то выполняются инструкции, стоящие после ELSE.

Пример:

IF temp < 17

THEN heating_on: = TRUE;

ELSE heating_on: = FALSE;

END_IF

В этом примере нагревание (heating) включается, когда температура опустится ниже 17^{o}, иначе оно останется выключенным.

С помощью инструкции CASE можно нескольким различным значениям целочисленной переменной сопоставить различные инструкции.

Синтаксис:

CASE <Var1> OF

<Value1>: <Instruction 1>

<Value2>: <Instruction 2>

<Value3, Value4, Value5>: <Instruction 3>

<Value6 .. Value10>: <Instruction 4>

...

<Value n>: <Instruction n>

ELSE <ELSE instruction>

END_CASE.

Инструкция CASE выполняется согласно следующим правилам:

– если переменная <Var1> имеет значение <Value i>, то выполняется инструкция <Instruction i>;

– если <Var1> не принимает ни одного из указанных значений, то выполняется <ELSE Instruction>;

– чтобы одна и та же инструкция выполнялась при различных значениях переменной <Var1>, необходимо перечислить эти значения через запятую;

– чтобы одна и та же инструкция выполнялась для целого диапазона значений, необходимо указать начальное и конечное значения, разделенные двумя точками.

Пример:

CASE INT1 OF

1, 5: BOOL1 : = TRUE;

BOOL3 : = FALSE;

2: BOOL2 : = FALSE;

BOOL3 : = TRUE;

10..20: BOOL1: = TRUE;

BOOL3: = TRUE;

ELSE

BOOL1 : = NOT BOOL1;

BOOL2 : = BOOL1 OR BOOL2;

END_CASE;

С помощью цикла FOR можно программировать повторяющиеся процессы.

Синтаксис:

INT_Var :INT;

FOR <INT_Var> : = <INIT_VALUE> TO <END_VALUE> {BY <Step size>} DO

<Instructions>

END_FOR

Часть конструкции, заключенная в фигурные скобки, не обязательна. <Instructions> выполняются, пока счетчик <INT_Var> не больше <END_VALUE>. Это условие проверяется перед выполнением <Instructions>, поэтому раздел <Instructions> не выполняется, если <INIT_VALUE> больше <END_VALUE>.

Всякий раз, когда выполняются <Instructions>, значение <INIT_VALUE> увеличивается на <Step_size>. <Step_ size> может принимать любое целое значение. По умолчанию шаг устанавливается равным 1.

Пример:

FOR Counter: = 1 TO 5 BY 1 DO

*Var1: = Var1*2;*

END_FOR;

Erg: = Var1;

В этом примере предполагается, что начальное значение Var1 равно 1. После выполнения цикла эта переменная будет равна 32. Цикл WHILE может использоваться, как и цикл FOR, с тем лишь различием, что условие выхода определяется логическим выражением. Это означает, что цикл выполняется, пока верно заданное условие.

Синтаксис:

WHILE <Boolean expression>

<Instructions>

END_WHILE

Раздел <Instructions> выполняется циклически до тех пор, пока <Boolean_expression> дает TRUE. Если <Boolean_expression> равно FALSE уже при первой итерации, то раздел <Instructions> не будет выполнен ни разу. Если <Boolean_expression> никогда не примет значение FALSE, то раздел <Instructions> будет выполняться бесконечно.

Пример:

WHILE counter<>0 DO

*Var1: = Var1*2;*

counter : = counter-1;

END_WHILE

Цикл REPEAT отличается от цикла WHILE тем, что первая проверка условия выхода из цикла осуществляется, когда цикл уже выполнился 1 раз. Это означает, что независимо от условия выхода цикл выполняется хотя бы один раз.

Синтаксис:
REPEAT
<Instructions>
UNTIL <Boolean expression>
END_REPEAT

Раздел <Instructions> выполняется циклически до тех пор, пока <Boolean_expression> дает TRUE.

Если <Boolean_expression> равно FALSE уже при первой итерации, то раздел <Instructions> не будет выполнен один раз.

Если <Boolean_expression> никогда не примет значение FALSE, то раздел <Instructions> будет выполняться бесконечно.

Если в циклах FOR, WHILE, REPEAT встречается инструкция EXIT, то цикл заканчивает свою работу независимо от значения условия выхода.

Работа в реальном времени. Работа системы в режиме реального времени осуществляется с помощью программы CoDeSys HMI. CoDeSys HMI – это система исполнения визуализаций, созданных в среде программирования CoDeSys (HMI − human-machine interface, человеко-машинный интерфейс).

Если проект содержит визуализацию, то после запуска CoDeSys HMI она открывается в полноэкранном режиме. Пользователь может управлять ею посредством мыши или клавиатуры.

Пользователь не имеет возможности редактировать программу. Меню и панели управления CoDeSys не доступны в операционной версии. Если необходимо, то функции управления и контроля проекта должны быть сопоставлены элементам визуализации при ее создании.

CoDeSys HMI устанавливается стандартным установщиком CoDeSys. Безлицензионная, ограниченная по времени демонстрационная версия входит в типовой комплект.

CoDeSys HMI (CoDeSysHMI.exe) запускается из командной строки. Как минимум, в командной строке должен указываться желаемый проект CoDeSys. Если строка вызова не содержит никаких других параметров, CoDeSys HMI стартует с визуализации PLC_VISU.

Кроме того, в вашем распоряжении имеются следующие специальные параметры: /simulation или /target. По умолчанию проект запускается в режиме, который был установлен, когда проект сохранял-

ся последний раз. Ключи /simulation или /target явно указывают, должен ли проект запускаться в режиме эмуляции или на целевой системе /visu <visualization POU>. Если проект содержит модуль визуализации (POU) с именем PLC_VISU, визуализация автоматически стартует с него. В противном случае в командной строке необходимо указать имя стартового компонента: «/visu <name of visualization POU>» /visudownload.

Блокировка загрузки. В стандартном случае при входе в систему с проектом, который отличается от версии в контроллере, пользователь может решать, проводить ли загрузку нового проекта или нет. Запись «visudownload = no» в файле codesys.ini блокирует запрос загрузки. Эту блокировку можно отключить ключом «/visudownload». Ключ /visucompactload служит оптимизации запуска проекта, для которого не требуется загрузка. Если она затребована опцией «/visudownload», то «/visucompactload» игнорируется.

Пример командной строки:

D:\PROGRAMME\CoDeSysHMI /simulation

D:\PROJECTS\PROJECT.PRO /visu overview

Проект project.pro запускается в режиме эмуляции с «overview» визуализацией.

Внимание, при вводе путей в командной строке: если в пути содержатся символы пробела, он должен заключаться в кавычки (").

Проект запускается в полноэкранном режиме. Дальнейшее управление проектом происходит посредством клавиатуры и мыши через элементы визуализации.

Если не предусмотрен специальный элемент визуализации для окончания работы, то CoDeSys HMI можно закрыть в любое время с помощью <Alt><F4>.

Порядок выполнения лабораторной работы

В данной работе создается простейшая программа управления кодовым замком двери для контроллера Овен ПЛК 150. Замок открывается активацией первого дискретного выхода контроллера. Терминал эмулируется на экране персонального компьютера как объект визуализации (рис. 46).

«Терминал» содержит 9 кнопок для ввода кода, кнопку «start» для сброса и индикатор открытия двери. Для простоты примем, что код состоит из пяти неповторяющихся цифр.

Создайте новый проект CoDeSys для контроллера ПЛК 150. Проект будет содержать единственный POU PLC_PRG на языке ST и объект визуализации.

Рис. 46. «Терминал» ввода кода

Объявление переменных. В PLC Configuration объявим переменную enter, «привязав» ее к первому дискретному выходу контроллера. Эта переменная будет отвечать за открытие двери.

В Global Variables объявим следующие глобальные переменные:

– s1, s2, *N*s9 типа BOOL, которые будут сигнализировать о нажатии цифровых кнопок «терминала»;

– start типа BOOL – будет сигнализировать о нажатии кнопки «start»;

– status типа INT – будет отражать ход процесса набора кода.

Все логические переменные инициализируем значением FALSE, а переменную status − нулевым значением.

Создание объекта визуализации. Создайте объект визуализации. Разместите в нем кнопки и индикатор так, как показано на рис. 46.

Назначьте переменные s1, s2, Ns9 соответствующим кнопкам, как изображено на рис. 47.

Аналогично поступите с переменной и кнопкой «start». Теперь при нажатии кнопок переменные будут принимать значение TRUE, при отпускании − FALSE.

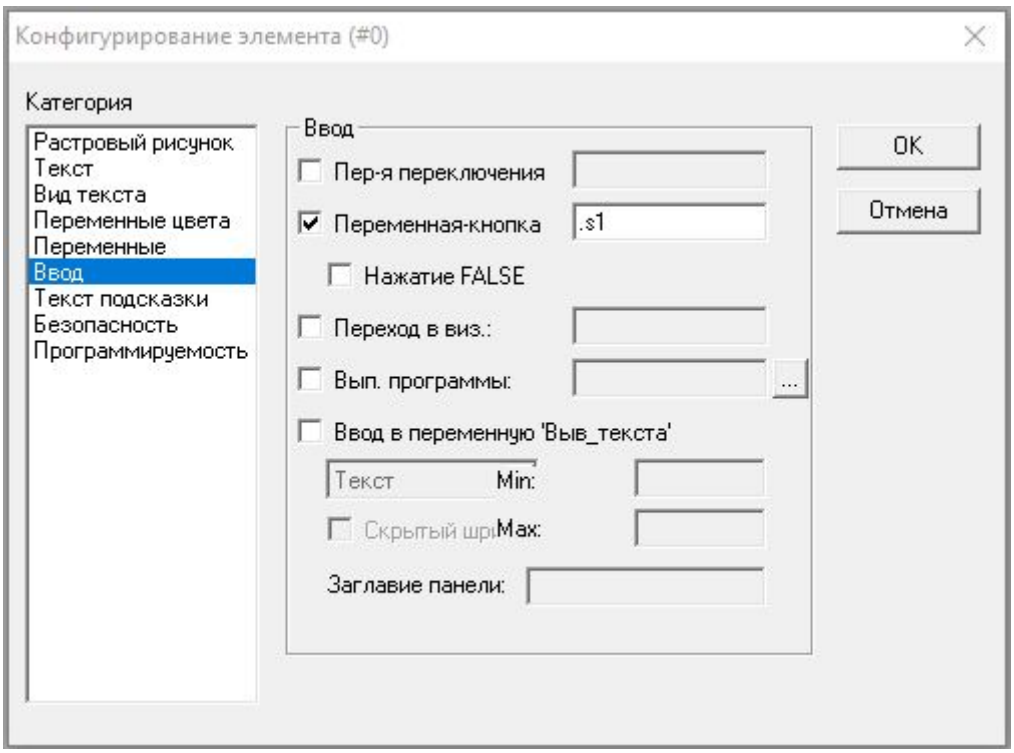

Рис. 47. Назначение переменных цифровым кнопкам

Индикатор должен изменять цвет с красного (замок закрыт) на зеленый (замок открыт) при изменении значения переменной enter с FALSE на TRUE.

PLC_PRG. Напишите программу управления кодовым замком на языке ST, воспользовавшись следующими указаниями.

1. Вначале программа должна проверять нажатие кнопки «start». Если кнопка нажата (start = TRUE), необходимо установить в FALSE переменную enter и обнулить переменную status. Используйте конструкцию IFNEND_IF.

2. Далее производится работа с переменной status. С каждым нажатием «правильной» кнопки производится увеличение этой переменной на единицу. При нажатии «неправильной» кнопки переменная сбрасывается в нуль. Если в ситуации, когда status = 4, нажата «правильная» кнопка (последняя) устанавливаем enter = TRUE.

Реализация описанной логики легко осуществляется с помощью конструкции CASENEND_CASE.

Пусть, например, используется код 29167, тогда фрагмент, связанный с вводом цифры «9» (изменение переменной status с единицы на двойку) может выглядеть следующим образом:

```
CASE status OF
0: NN
NN
1: IF s9 = TRUE THEN
status: = 2;
ELSIF (s1 OR s3 OR s4 OR s5 OR s6 OR s7 OR s8) = TRUE THEN
status: = 0;
END_IF
2: NN
NN
3: NN
NN
4: NN
NN
END_CASE
```

Обратите внимание, что в секции ELSIF не проверяется переменная s2, связанная с кнопкой «2». Эта кнопка нажималась на предыдущем шаге и в момент выполнения инструкции может еще удерживаться пользователем.

Не забудьте на последнем шаге открыть дверь!

По окончанию ввода программы, проверьте ее работу в режиме симуляции, после чего загрузите ее в контроллер.

Работа в реальном времени. Создайте ярлык для запуска программы CoDeSys HMI с разработанной визуализацией.

После загрузки программы в контроллер, отключите связь с контроллером (Logout). С помощью ярлыка запустите программу CoDeSys HMI.

Протестируйте работу системы в реальном времени.

Содержание отчета

1. Программа, реализованная в ходе выполнения работы, с комментариями ко всем блокам и операторам.
2. Описание объекта визуализации.

Контрольные вопросы и задания

1. Дайте краткую характеристику языка ST.
2. Приведите и опишите синтаксис операции условного перехода в языке ST.
3. Приведите и опишите синтаксис цикла FOR в языке ST.
4. Приведите и опишите синтаксис цикла WHILE в языке ST.
5. Приведите и опишите синтаксис цикла REPEAT в языке ST.
6. Опишите способ и параметры запуска визуализация CoDeSys в реальном времени.
7. Какова логика работы программы управления кодовым замком?