Системы реального времени

Лекция 3

Стандарт программирования ПЛК

Использование даже простых, но полномасштабных компьютерных конфигураций на нижних уровнях промышленных систем для непосредственного управления оборудованием невыгодно как по экономическим соображениям, так и по причине избыточной сложности. В системах автоматизации для этого традиционно применяются более простые и дешевые устройства – *программируемые логические контроллеры (ПЛК)*. В первом поколении ПЛК представляли собой релейные схемы, вырабатывающие сигналы для оборудования по правилам булевой логики. Сегодня они стали более интеллектуальными [1].

Одним из главных требований к ПЛК всегда была и остается возможность его эксплуатации существующим техническим персоналом и возможность быстрой замены старого оборудования. Поэтому языки программирования компьютеров и встраиваемых микропроцессорных систем управления плохо подходят для программирования ПЛК. Здесь нужны более простые и наглядные языки, позволяющие излагать задачу в близких к применяемым технологиям категориях. Привлечение же к программированию специализированной фирмы неизбежно порождает зависимость, если реализация не является достаточно прозрачной. Сложный язык программирования ПЛК снижает его шансы на конкурентном рынке существенно больше, чем массогабаритные показатели [2].

С целью повышения скорости и качества разработки программ для ПЛК в 1979 году в рамках *Международной электротехнической комиссии (МЭК)* была создана специальная группа технических экспертов по проблемам ПЛК, включая аппаратные средства, монтаж, тестирование, документацию и связь. В 1982 году был выпущен первый вариант стандарта *МЭК 61131* (*IEC 61131* или *IEC 1131*), а всего на текущий момент времени было выпущено 3 редакции. Стандарт включает в себя 8 частей:

1) Общая информация.
2) Требования к оборудованию и тестам.
3) Языки программирования.
4) Руководства пользователя.
5) Спецификация сообщений.
6) Промышленные сети.
7) Программирование с нечеткой логикой.
8) Руководящие принципы применения и реализации языков ПЛК.

Рассмотрим более подробно 3-ю часть стандарта (МЭК 61131-3). В этой части описаны 5 языков программирования ПЛК. Среди них три графических и два текстовых. К графическим языкам относятся: язык релейных схем, язык функциональных блоков, язык диаграмм состояний. К текстовым – assembler-подобный язык Instruction List и паскале-подобный язык Structured Text.

Такое количество языков программирования ПЛК, определённое стандартом МЭК 61131-3, объясняется тем, что при разработке стандарта было обнаружено так много вариаций языков контроллеров, что оказалось невозможно выбрать какой-то один язык программирования как базовый [1].

Языки программирования ПЛК

Языки стандарта МЭК 61131-3 базируются на следующих принципах [3]:

- вся программа разбивается на множество функциональных элементов - Program Organization Units (POU), каждый из которых может состоять из функций, функциональных блоков и программ. Любой элемент программы может быть сконструирован иерархически из более простых элементов;
- стандарт требует строгой типизации данных. Указание типов данных позволяет легко обнаруживать большинство ошибок в программе до ее исполнения;
- имеются средства для исполнения разных фрагментов программы в разное время, с разной скоростью, а также параллельно. Например, один фрагмент программы может сканировать концевой датчик с частотой 100 раз в секунду, в то время как второй фрагмент будет сканировать датчик температуры с частотой один раз в 10 сек;
- для выполнения операций в определенной последовательности, которая задается моментами времени или событиями, используется специальный язык последовательных функциональных схем (SFC);
- стандарт обеспечивает совместное использование всех пяти языков, поэтому для каждого фрагмента задачи может быть выбран любой, наиболее удобный, язык;
- программа, написанная для одного контроллера, может быть перенесена на любой контроллер, совместимый со стандартом МЭК 61131-3.

Все языки, включенные в стандарт МЭК 61131-3, можно комбинировать. Полная реализация МЭК 61131-3 доступна как коммерческий продукт: например, это инструментальная графическая среда разработки прикладных программ для программируемых логических контроллеров ISaGRAF производства фирмы Rockwell Automation (Милуоки, штат Висконсин, США), инструментальный программный комплекс промышленной автоматизации CODESYS фирмы 3S-Smart Software Solutions GmbH (Кемптен, Германия), среда разработки MULTIPROG фирмы Advantech и другие [7].

Язык релейных схем (Ladder Diagram, LD)

Язык релейных или *релейно-контактных схем (Ladder Diagram, LD)* – графический язык, реализующий структуры электрических цепей. Впервые появился в виде электрических схем, которые состояли из контактов и обмоток электромагнитных реле. Такие схемы использовались в автоматике конвейеров для сборки автомобилей до эры микропроцессоров. Язык релейной логики был интуитивно понятен людям, слегка знакомым с электротехникой и поэтому оказался наиболее распространенным в промышленной автоматике. Обслуживающий персонал легко находил отказ в оборудовании, прослеживая путь сигнала по релейной диаграмме (рисунок 1).

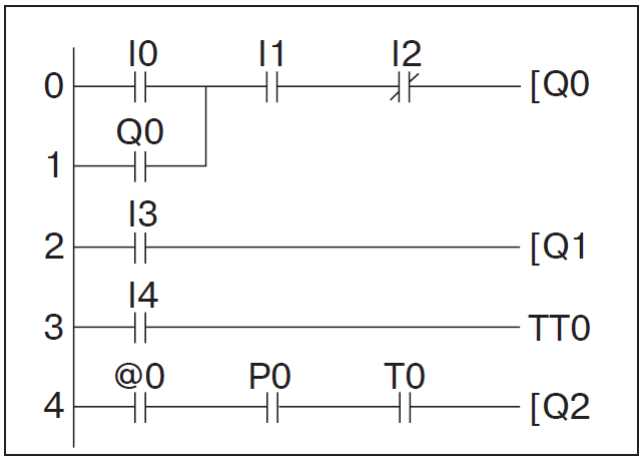

Рисунок 1 – Пример программы на языке LD

Основными элементами языка являются *контакты* и *катушки*. Различаются *нормально замкнутые* и *нормально разомкнутые контактные элементы*, которые можно сопоставить с нормально замкнутыми и нормально разомкнутыми кнопками в электрических цепях.

- ─┤ ├─ Нормально разомкнутый контакт.
  Разомкнут при значении ложь, назначенной ему переменной и замыкается при значении истина.
- ─┤ / ├─ Нормально замкнутый контакт.
  Замкнут, если переменная имеет значение ложь, и разомкнут, если переменная имеет значение истина.
- ─( )─ Катушка.
  Итог логической цепочки копируется в целевую переменную, которая называется катушка (англ. coil). Это слово имеет обобщенный образ исполнительного устройства, поэтому в русскоязычной документации обычно говорят о выходе цепочки, хотя можно встретить и частные значения термина, например катушка реле.

Конкретные версии языка реализуются обычно в рамках программных продуктов, для работы с определенными типами ПЛК. Часто такие реализации содержат команды, расширяющие множество стандартных команд языка, что вызвано желанием производителя полнее учесть желания заказчика, но в итоге приводят к несовместимости программ, созданных для контроллеров различных типов.

Однако язык LD проблематично использовать для реализации сложных алгоритмов, поскольку он не поддерживает подпрограммы, функции, инкапсуляцию и другие средства структурирования программ с целью повышения качества программирования. Эти недостатки затрудняют многократное использование программных компонентов, что делает программу длинной и сложной для обслуживания. Сложные вычисления в этом языке невозможны. Недостатком является также то, что только маленькая часть программы умещается на мониторе компьютера или панели оператора при программировании.

Несмотря на указанные недостатки, язык LD относится к наиболее распространенным в мире [3], хотя чаще всего используется для программирования только простых задач.

Язык функциональных блоков (Function Block Diagram, FBD)

Язык функциональных блоков (Function Block Diagram, FBD) является графическим языком и наиболее удобен для программирования процессов прохождения сигналов через функциональные блоки. Язык FBD удобен для схемотехников, которые легко могут составить электрическую схему системы управления на «жесткой логике», но не имеют опыта программирования.

Функциональные блоки представляют собой фрагменты программ, написанных на IL, SFC или других языках, которые могут быть многократно использованы в разных частях программы и которым соответствует графическое изображение, принятое при разработке функциональных схем электронных устройств (см. рисунок 2).

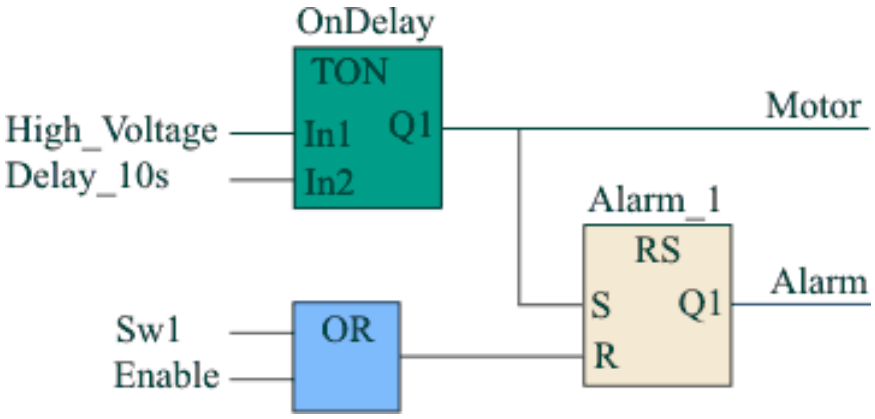

Рисунок 2 – Пример программы на языке FBD

Язык FBD может быть использован для программирования функций, функциональных блоков и программ, а также для описания шагов и переходов в языке SFC. Функциональные блоки инкапсулируют данные и методы, чем напоминают объектно-ориентированные языки программирования, но не поддерживают наследование и полиморфизм.

Типичным применением языка FBD является описание «жесткой логики» и замкнутых контуров систем управления. Язык функциональных блоков является удобным также для создания и пополнения библиотеки типовых функциональных блоков, которую можно многократно использовать при программировании задач промышленной автоматизации. К типовым блокам относятся блок таймера, ПИД-регулятора, блок секвенсора, триггера, генератора импульсов, фильтра, и т.п. [4]

Язык диаграмм состояний (Sequential Function Chart, SFC)

Графический язык диаграмм состояний (Sequential Function Chart, SFC) создан на базе математического аппарата сетей Петри, описывающий последовательность состояний и условий переходов. Для создания программы используются следующие структурные элементы: шаг (и начальный шаг), переход, блок действий, прыжок и связи типа дивергенция и конвергенция (рисунок 3).

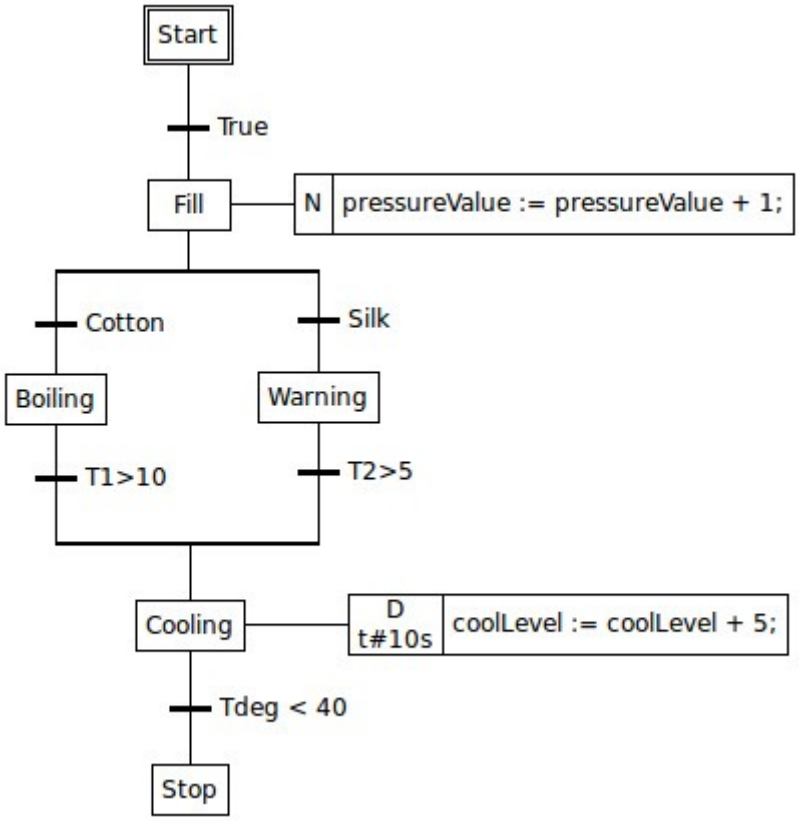

Рисунок 3 – Пример программы на языке SFC

Программа состоит из шагов и условий переходов. Шаги показываются на схеме прямоугольниками, а условия переходов – жирной перечеркивающей линией. Программа выполняется сверху вниз. Начальный шаг на схеме показывается в виде двойного прямоугольника. Условия переходов записываются рядом с их обозначениями. Каждый шаг программы может представлять собой реализацию сложного алгоритма, написанного на одном из языков стандарта МЭК 61131-3 [4].

При выполнении программы каждый шаг активен до тех пор, пока не активен переход, следующий за ним. Как только активируется переход, текущий шаг отключается и активируется следующий шаг в последовательности. Переход также может иметь код, который образует необходимые условия для выполнения перехода.

Язык удобно использовать в системах с повторяющимися многошаговыми процессами или последовательностью повторяющихся процессов. Язык достаточно нагляден, а сегментация кода упрощает поиск неисправностей.

К недостаткам относится то, что такой стиль программирования подходит не для всех систем, т.к. структура, которая накладывается на программу, может ее значительно усложнить. Дополнительные ресурсы, требующиеся для такого программирования, приводят к замедлению процесса написания программы по сравнению с другими языками. Также язык очень сложно конвертировать в другие языки [5].

Assembler-подобный язык Instruction List (IL)

Язык Instuction List (IL) – один из текстовых языков программирования стандарта МЭК 61131-3. Синтаксис языка очень похож на Assembler.

Текст программы состоит из нескольких строк, каждая из которых содержит одну операцию (инструкцию процессора). Т.о. программа представляет собой пошаговый список инструкций, который легко приводим к последовательности математических формул. В таблице 1 представлены инструкции языка IL.

Таблица 1 – Список инструкций языка IL

№	Инструкция	Модификатор	Тип данных операнда	Действие
1	**LD**	**N**	Любой	Присвоить аккумулятору значение операнда
2	**ST**	**N**	Любой	Присвоить значение аккумулятора операнду
3	**S**		Логический	Присвоить лог. операнду значение ИСТИНА
4	**R**		Логический	Присвоить лог. операнду значение ЛОЖЬ
5	**AND**	**N, (**	Логический	Логическое И
6	**OR**	**N, (**	Логический	Логическое ИЛИ
7	**XOR**	**N, (**	Логический	Исключающее логическое ИЛИ
8	**ADD**	**(**	Любой	Сложение
9	**SUB**	**(**	Любой	Вычитание
10	**MUL**	**(**	Любой	Умножение
11	**DIV**	**(**	Любой	Деление
12	**GT**	**(**	Любой	Сравнение с содержимым аккумулятора: >
13	**GE**	**(**	Любой	Сравнение с содержимым аккумулятора: >=
14	**EQ**	**(**	Любой	Сравнение с содержимым аккумулятора: =
15	**NE**	**(**	Любой	Сравнение с содержимым аккумулятора: <>
16	**LE**	**(**	Любой	Сравнение с содержимым аккумулятора: <=
17	**LT**	**(**	Любой	Сравнение с содержимым аккумулятора: <
18	**JMP**	**C, N**	Имя метки	Переход на метку
19	**CAL**	**C, N**	Имя функции	Вызов функции
20	**RET**	**C, N**		Возврат из функции или блока
21	**)**			Окончание вложенной операции

Пример программы на языке IL приведён на рисунке 4:

```
LD X1    (* загрузить в аккумулятор значение X1 *)
         SUB X2         (* из X1 вычесть X2 *)
         ST Y           (* загрузить в Y результат выполнения X1-X2 *)
         MUL Y          (* умножить Y на Y *)
         ST Result      (* сохранить результат операций в Result *)
         GT Max         (* значение Result > Max? *)
         JMPC TEST_OK   (*  *)
TEST_OK: LD 100         (*  *)
```

Рисунок 4 – Пример программы на языке IL

Перед оператором может находиться метка, оканчивающаяся двоеточием « **:** ». Комментарий должен быть последним элементом в строке.

В IL можно использовать следующие модификаторы (см. таблицу 2):

Таблица 2 – Применение модификаторов с операторами

Модификатор с инструкциями	Описание
C с **JMP**, **CAL**, **RET**	инструкция выполняется только тогда, когда результат аккумулятора ИСТИНА
N с **JMPC**, **CALC**, **RETC**	инструкция выполняется тогда, когда результат аккумулятора ЛОЖЬ
N	в других случаях: отрицание операнда

Т.к. IL является языком нижнего уровня, то программы, написанные на нем, работают быстрее аналогов, созданных с помощью графических языков. Также программа занимает меньше памяти по сравнению с остальными языками.

Несмотря на вышеперечисленные достоинства, язык имеет ряд существенных недостатков. Во-первых, язык ненагляден по сравнению с графическими языками, и поэтому сложно понять, что делает программа и какие ошибки могут возникнуть. Во-вторых, по мере усложнения ПЛК в списке инструкций могут возникнуть сложности при вводе некоторых функций, например, ПИД-регулирование [5].

Использование языка IL постепенно снижается. К тому же он был исключен из списка языков МЭК 61131-3 в 3-й редакции стандарта.

Паскале-подобный язык Structured Text (ST)

Язык Structured Text (ST) – второй из текстовых языков программирования стандарта МЭК 61131-3. Его синтаксис очень похож на Паскаль.

В таблице 3 представлены основные языковые конструкции.

Таблица 3 – Основные языковые конструкции языка ST

№	Выражение	Пример

1	Присваивание	`A:= B;`
		`CV:= CV+1;`
		`C:= SIN(X);`

2	Вызов функции или процедуры	`CMD_TMR(IN:= %IX5, PT:= T#300ms);` `A:= CMD_TMR.Q;`
3	Возврат RETURN	`RETURN;`

4	Конструкция IF	`D:= B * B – 4 * A * C;`
		`IF D < 0.0 THEN NROOTS:= 0;`
		`ELSIF D = 0.0 THEN`
		`  NROOTS:= 1;`
		`  X1:= -B / (2.0 * A);`
		`ELSE`
		`  NROOTS:= 2;`
		`  X1:= (-B + SQRT(D)) / (2.0 * A);`
		`  X2:= (-B - SQRT(D)) / (2.0 * A);`
		`END_IF;`

5	Конструкция CASE	`TW:= BCD_TO_INT(THUMBWHEEL);`
		`TW_ERROR:= 0;`
		`CASE TW OF`
		`  1,5: DISPLAY:= OVEN_TEMP;`
		`  2: DISPLAY:= MOTOR_SPEED;`
		`  3: DISPLAY:= GROSS - TARE;`
		`  4,6..10: DISPLAY:= STATUS(TW - 4);`
		`  ELSE DISPLAY:= 0;`
		`  TW_ERROR:= 1;`
		`END_CASE;`
		`QW100:= INT_TO_BCD(DISPLAY);`

6	Цикл FOR	`J:= 101 ;`
		`FOR I:= 1 TO 100 BY 2 DO`
		`  IF WORDS[I] = 'KEY' THEN`
		`  J:= I;`
		`  EXIT;`
		`  END_IF;`
		`END_FOR;`

7	Цикл WHILE	`J:= 1;`
		`WHILE J <= 100 & WORDS[J] <> 'KEY' DO`
		`  J:= J + 2;`
		`END_WHILE;`

8	Цикл REPEAT (аналог DOWHILE)	`J:= -1;`
		`REPEAT`
		`  J:= J + 2 ;`
		`UNTIL J = 101 OR WORDS[J] = 'KEY'`
		`END_REPEAT;`

9	EXIT (выход)	`EXIT;`
10	Пустой оператор	`;`

На рисунке 5 представлен пример программы на языке ST.

```
Quantity := FromLimitA + FromLimitB + FromLimitC;

IF (Quantity<= (100000-MainQtyPrevious) ) THEN
      IF ( FromProcA AND FromProcB AND FromProcC) THEN
            EnableA := True;
            EnableB := True;
            ENableC := True;
            IF Level[0]< FromLimitA THEN
                  Level[0] := Level[0] + 1;
                  MainQuantity := MainQuantity + 1;
                  IF (Level[0]=FromLimitA) THEN
                        EnableA := False;
                        Level[0] := 0;
                  END_IF;
            END_IF;

            IF Level[1]< FromLimitB THEN
                  Level[1] := Level[1] + 1;
                  MainQuantity := MainQuantity + 1;
                  IF (Level[1]=FromLimitB ) THEN
                        EnableB:=False;
                        Level[1]:=0;
                  END_IF;
            END_IF;
```

Рисунок 5 – Пример программы на языке ST

Ещё один короткий пример программы ST представлен на рисунке 6:

```
WHILE A >= B DO
  A:= A - B
END_WHILE;
Result:= A;
```

Рисунок 6 – Ещё один пример программы на языке ST

Этот язык лучше всего подходит для сложного программирования ПЛК. Тригонометрические функции, математические вычисления и анализ данных на этом языке можно реализовать легче, чем на языке релейно-контактных схемах или языке списка инструкций. Текстовый, неграфический характер языка ST, похожего на язык IL,

позволяет создавать программы, которые работают гораздо быстрее, чем программа созданные на графических языках. Дополнительным преимуществом языка ST является то, что он ближе других языков программирования подошел к достижению переносимости, обещанной стандартом МЭК 61131. Окончательным преимуществом является то, что многие студенты инженерных специальностей лучше владеют компьютерными языками, чем основами электротехники, и поэтому лучше владеют языком ST, чем, например, LD.

Недостаток языка ST заключается в том, что для многих старых специалистов в области программирования и отладки среда языка ST является чем-то незнакомым и неудобным. В определенном смысле, код и структура необходимые, чтобы сделать поддержку этого кода удобной, снижают преимущества, связанные с компактностью программ. В результате основной тенденцией использования языка ST является его использование так сказать «за сценой». Например, МЭК 61131 позволяет программисту реализовать функции на одном языке, а затем использовать их в другом языке. Таким образом, программист, скорее всего, включит программу на языке ST внутрь команды, вызываемой на языке LD. Это не обязательно является недостатком, но программисту понадобится тщательно протестировать любой «скрытый» код, и удостовериться в отсутствии ошибок, поскольку у других, кто будет использовать этот код, возможно доступа к данному коду не будет [5].

Будущее стандарта МЭК 61131-3 связывается со стандартизацией форматов программных файлов, что обеспечит возможность обмена пакетами функциональных блоков между различными платформами и позволит реально перейти к модульному программированию – созданию больших систем из готовых пакетов. Второе направление развития – использование в распределенных управляющих системах функциональных блоков за пределами программируемых контроллеров. С этой целью был создан стандарт IEC TC65/WG6, в котором концепции IEC 61131-3 применены к стандарту Fieldbus и который определяет, каким образом средства связи могут объединяться с прикладным ПО.

Вопросы для самоконтроля:

1)

Список использованных источников

1) Системы реального времени: учебное пособие / А.М. Сулейманова. – Уфа: Уфимск. гос. авиац. техн. ун-т. Уфа, 2004. – 292 с.
2) Петров И.В. Программируемые контроллеры. Стандартные языки и приемы прикладного проектирования / Под ред. проф. В.П. Дьякова – М.: СОЛОН-Пресс, 2004.- 256 с: ил.
3) Lewis R.W. Programming industrial control systems using IEC 113-3 Revised edition. - The Institution of Electrical Engineers, London, 1998. - 329 с.

4) Системы программирования на языках МЭК 61131-3 / Энциклопедия АСУ ТП [Электронный ресурс]. URL: http://www.bookasutp.ru/Chapter9_3.aspx (дата обращения: 17.05.2014)
5) Понимание языков программирования IEC61131-3/ Про АСУ ТП. Профессионально и в деталях [Электронный ресурс]. URL: http://proasutp.com/articles/plc/understanding_the_iec61131_3_programming_languages.html (дата обращения: 18.05.2014)
6) Системы реального времени: обзорный курс лекций / К.Е. Климентьев. – Самара: Самар. гос. аэрокосм. ун-т. Самара, 2008. – 45 с.
7) IEC 61131-3 / Википедия [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/IEC_61131-3 (дата обращения: 26.11.2017)