И.В. ПЕТРОВ (Компания “Пролог”), Р. ВАГНЕР (Компания “3S-Smart Software Solutions GmbH”)

Отладка прикладных ПЛК программ в CoDeSys (часть 3)[1]

Компьютерная программа выполняет то,
что вы ей приказали делать,
а не то, чтобы вы хотели, чтобы она делала.
Третий закон Грида

Вы могли предположить, что такой эпиграф выбран авторами специально для апрельского номера журнала. Это правда. Тем не менее, он абсолютно точно выражает стержневую мысль данного цикла статей.

В предыдущей части мы остановились на обещании получить статистику популярности языков МЭК 61131-3 на основе изучения данных службы технической поддержки 3S-Smart Software Solutions GmbH. Специалисты 3S предоставили нижеследующие данные и краткие комментарии к ним (рис. 1).

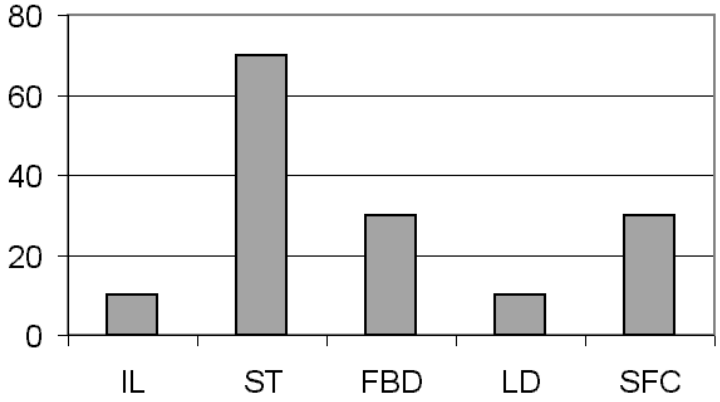

Рис.1. Сравнение популярности языков МЭК

IL регулярно используют менее 10 % пользователей. Главным образом, это бывшие пользователи Siemens S5/S7. Но из-за достаточно большого числа отличий в привычных для них конструкциях все они со временем осваивают другие языки.

ST регулярно используют до 70 % пользователей, и их число постоянно растет. Как правило, это люди, имеющие высшее образование. В ST пишутся наиболее сложные проекты на ПЛК, включающие элементы технического зрения, биометрию и т.п. Ряд важнейших составных частей самого комплекса CoDeSys написаны на языке ST (стандартные библиотеки, библиотеки CANopen и SoftMotion, модуль целевой визуализации).

FBD и CFC регулярно применяют около 30 % пользователей. FBD – это самый популярный язык у начинающих программистов. Все пользователи CoDeSys знают FBD и используют его с разной интенсивностью. CFC, по сути, отличается от FBD только редактором. Он широко применяется для программирования регуляторов непрерывного действия.

LD регулярно используют не менее 10 % пользователей CoDeSys. Он является старейшим языком ПЛК. В багаже каждого опытного специалиста присутствуют изящные приемы на языке LD. Он активно применяется всеми пользователями, особенно в задачах логического управления. Наиболее популярен язык LD в США и Великобритании.

SFC применяют около 30 % пользователей. Диаграммы SFC являются самым выразительным графическим средством. Но их эффективное использование требует специальной подготовки. Ее имеют далеко не все наши пользователи. SFC-исполнитель требует определенных ресурсов процессора, что несколько ограничивает его применение в самых массовых и дешевых контроллерах. Чаще всего SFC используют совместно с ST.

Как вы наверняка заметили, сумма составляет 150 %. Из этого следует, что от 12,5 до 50 % пользователей постоянно используют в своих проектах два и более языков МЭК 61131-3.

Пожалуйста, не делайте обобщений на основе этих данных. Они относятся только к пользователям, обращавшимся в 3S с вопросами. По причине того, что язык LD популярен среди опытных специалистов, мало обращающихся за консультациями, его рейтинг вероятно существенно занижен.

[1] *Продолжение. Начало в №№ 2, 3 2006 г.*

Языки МЭК можно попытаться отсортировать по выразительной мощности и применимости на основных этапах проектирования. Соответствующая иерархическая диаграмма (рис. 2) была приведена в книге [1]. За счет специальных расширений стандарта МЭК в CoDeSys область применимости ST и FBD оказывается несколько шире.

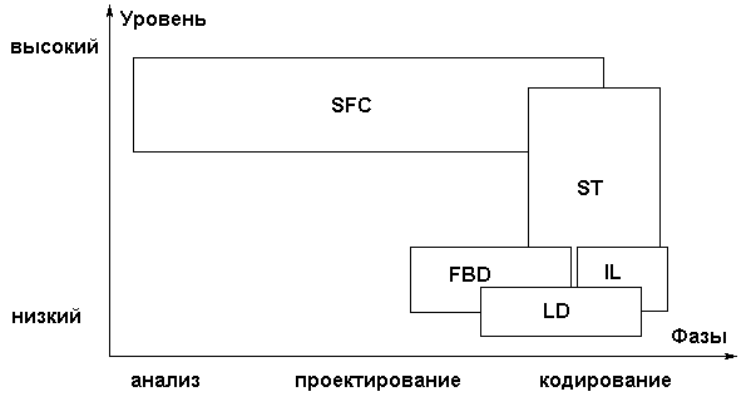

Рис. 2. Применимость языков МЭК.

Далеко не все пользователи имеют возможность свободно выбирать язык программирования ПЛК. В крупных компаниях существуют специальные правила, оговаривающие язык программирования, способы создания идентификаторов и документирования проектов. Например, все машины крупнейшего пользователя CoDeSys компании Robert Bosch GmbH запрограммированы на SFC и ST. Компания тратит большие средства на обучение своих сотрудников. Ограничение свободы выбора языков снижает эти расходы. Жесткие правила обеспечивают понимание текстов программ разными людьми. Это особенно важно для технического персонала, обязанного оперативно вносить изменения в “старые” программы. Редакторы и инструменты программирования для разных языков могут иметь собственные тонкости. Постоянное использование только двух языков повышает надежность программ.

Приверженцев IL мало. Тем не менее, исключить его из CoDeSys нельзя. Существует большое число контроллеров, для программирования которых применяются собственные фирменные системы. Язык IL в них очень распространен, а в некоторых даже сертифицирован. Таким образом, IL позволяет пользователям таких контроллеров перенести свои тщательно разработанные и проверенные временем проекты в CoDeSys без переработки. Для программ S5/S7 в CoDeSys включен специальный конвертер.

Еще одна редкая, но важная роль IL состоит в следующем: язык IL наиболее прозрачно транслируется в ассемблер целевого процессора. Он оптимален для низкоуровневого кодирования (рис. 2). Опытным программистам это позволяет писать быстрые и переносимые “ассемблерные вставки” непосредственно в CoDeSys. Для некоторых типов ПЛК в CoDeSys разрешено писать обработчики аппаратных прерываний. Разрешить или запретить пользователям такую возможность решает изготовитель контроллера. Это же относится и к возможности включения в проект внешних библиотек, написанных на С или Ассемблере.

Решение 5

В предыдущей части статьи мы рассмотрели несколько способов решения простейшей задачи. Каждое решение служило иллюстрацией одной из методик программирования. Чтобы сконцентрировать ваше внимание именно на этом, а не на деталях задачи, она была намеренно выбрана примитивно простой. Суть ее состоит в создании автомата, попеременно включающего и выключающего свет по нажатию нескольких кнопок.

Технология

Одним из широко известных методов программной реализации автоматов является метод, использующий внутреннее многозначное кодирование [2]. Суть состоит в задании состояния автомата значением единственной целочисленной переменной (State). В нашей задаче всего два состояния. В первом состоянии свет выключен, назовем его State_1. Во втором состоянии (State_2) свет включен. Находясь в любом из состояний, автомат должен выполнять как минимум 2 очевидных действия. Первое действие анализирует условия переходов и изменяет соответствующим образом переменную State. В результате, уже в следующем рабочем цикле активным может стать другое состояние. В нашем случае, переход из одного состояния в другое должен происходить по переднему фронту любого входа, связанного с кнопками управления. Второе действие (шаговое) выполняет полезную работу, соответствующую состоянию. В нашем случае, оно должно поддерживать соответствующее значение на выходе.

В CoDeSys есть замечательное свойство, расширяющее стандарт МЭК, – возможность добавлять в функциональный блок действия. Это вложенные подпрограммы, имеющие доступ ко всем переменным функционального блока, включая внутренние. Без них нам пришлось бы весь код писать в теле функционального блока.

В больших программах это привело бы к тексту, просматривая который до конца, уже забываешь, что было в начале, либо нужно было “открыть” внутренние переменные для доступа извне и программировать действия обычными POU. Действия позволяют “спрятать” детали реализации внутри функционального блока. В языках объектно-ориентированного программирования (ООП) это называется “инкапсуляцией”. В организаторе объектов CoDeSys действия показаны в виде раскрывающегося списка под функциональным блоком (рис. 3). Действия можно вызывать как снаружи, так и изнутри функционального блока. Если у вас возникает необходимость в POU, неразрывно связанных с функциональным блоком, то используйте действия.

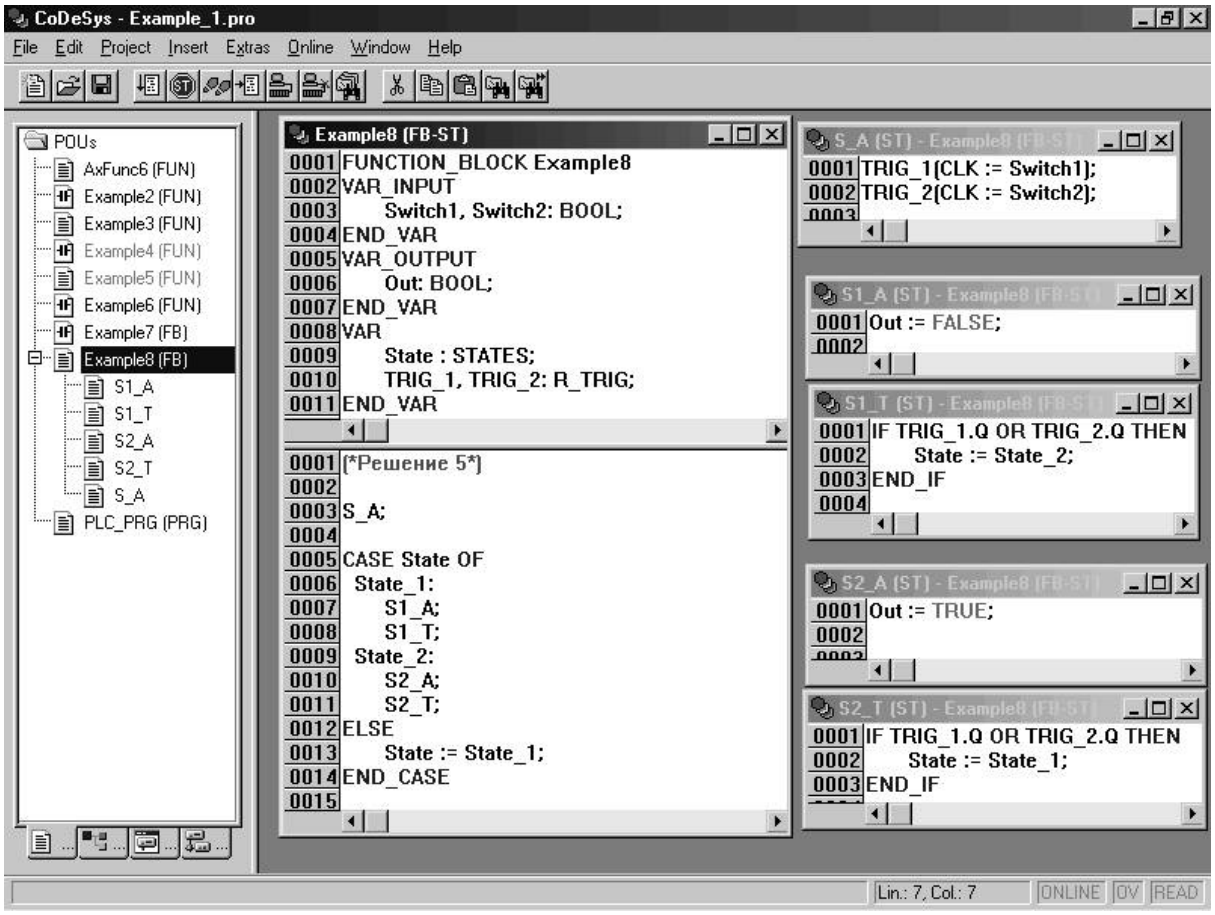

Рис. 3. Решение 5.

В теле функционального блока мы будем писать только текст, реализующий “голый” селектор состояний, включающий вызовы шаговых действий и действий переходов. Все детали прячем в соответствующие действия. Шаговые действия будем называть Sn_A (action), действия переходов – Sn_T (transaction). Часто возникает нужда в некоторой служебной работе (фильтрация, контроль входов, обслуживание таймеров и т.д.), которую нужно делать всегда, независимо от состояния. С этой целью логично ввести общее действие S_A.

Реализация

Давайте, наконец, возьмем самый популярный язык ST и реализуем нашу разобранную на атомы задачу. Вполне возможно, это будет проще и компактнее, чем словесное описание.

Прежде всего, перейдите на вкладку Data Types в организаторе объектов CoDeSys. В контекстном меню (правая клавиша мышки) дайте команду Add Object. Присвойте новому объекту имя STATES. Описываться он должен так:

TYPE STATES: (State_1,State_2);
END_TYPE

Это не обязательно, но полезно. Мы создали собственный специализированный тип данных для переменной состояния. Тем самым резко снижаются наши шансы случайно присвоить ей ошибочное значение. Мы переложили эту лишнюю ответственность на CoDeSys. Создаем функциональный блок на языке ST. В разделе внутренних переменных объявите пару экземпляров R_TRIG и переменную State: STATES.

Вставьте 5 действий в функциональный блок. Назовите их в соответствии с нашими правилами. Для вставки действия в CoDeSys дайте команду Add action в контекстном меню.

Действие S_A реализовано так:

```
TRIG_1(CLK := Switch1);
TRIG_2(CLK := Switch2);
```

Действие S1_A:

```
Out := FALSE;
```

Действие S2_A:

```
Out := TRUE;
```

Действие S1_T:

```
IF TRIG_1.Q OR TRIG_2.Q THEN
    State := State_2;
END_IF
```

Действие S2_T:

```
IF TRIG_1.Q OR TRIG_2.Q THEN
    State := State_1;
END_IF
```

Раздел реализации функционального блока:

```
S_A;
CASE State OF
  State_1:
    S1_A;
    S1_T;
  State_2:
    S2_A;
    S2_T;
ELSE
    State := State_1;
END_CASE
```

Последняя ветка ELSE алгоритмически избыточна. Это дань концепции “защитного программирования”. Если произойдет сбой в работе и переменная State примет недопустимое значение, то за один такт наш автомат самостоятельно исправит проблему. Полностью решение представлено на рис. 3.

Заключение

Рассмотренное решение тривиальной задачи может показаться громоздким. В действительности же мы “отлили пушку”, пригодную для взятия крепостей, а не только уничтожения мух. Сформулированные жесткие правила позволяют достаточно формально (не задумываясь) составить программу для любого детерминированного автомата, описанного любым способом (например, графом переходов). Справедливо и обратное. По программе можно формально получить описание автомата. Каждое действие выражено компактным текстом, имеющим однозначно определенный смысл. Автомат полностью реализуется одним функциональным блоком. Причем его раздел реализации практически не зависит от решаемой задачи.

На самом деле, мы использовали композицию автоматов, поскольку стандартный функциональный блок R_TRIG – это тоже автомат, имеющий 2 состояния. Мы можем легко закодировать его по той же технологии. Для “продвинутых” пользователей заметим, что данный метод основан на автомате Мура [3]. Он является вариацией на тему switch технологии [2, 4] с применением ООП возможностей CoDeSys.

Есть в нем и один мощный плюс с точки зрения отладки. Нам достаточно организовать трассировку значений единственной переменной в каждом рабочем цикле. Синхронизировав результат с наступлением критической ситуации, мы сможем восстановить последовательность предшествующих ей событий. Для этого в CoDeSys есть очень удобный готовый инструмент. Впрочем, речь об инструментах отладки и моделирования CoDeSys еще впереди.

Игорь Викторович Петров – технический директор "Пролог".
Роланд Вагнер – директор по маркетингу "3S-Smart Software Solutions GmbH".

Продолжение следует.

Список литературы

1. *F. Bonfatti, P.D. Monari, U. Sampieri*. IEC 1131-3 Programming methodology // CJ International. 1999.
2. *Шалыто А.А.* Логическое управление. Методы аппаратной и программной реализации алгоритмов. СПб: Наука, 2000.
3. *Карпов Ю.Г.* Теория автоматов. СПб: Питер, 2002.
4. *Альтерман И.З., Шалыто А.А.*. Формальные методы программирования логических контроллеров // Промышленные АСУ и контроллеры. 2005. № 10.