И.В. ПЕТРОВ (Компания “Пролог”)

Отладка прикладных ПЛК программ в CoDeSys (часть 2)[1]

Продолжим исследования в области программирования и отладки прикладных программ на языках МЭК 61131-3 в CoDeSys. Рассмотрим несколько способов решения одной несложной задачи. В данном цикле статей изложение материала идет снизу вверх. То есть не от теории к практике, а наоборот. Пока мы только продолжаем набор высоты, необходимой для широкого обзора.

Внимательные читатели, обладающие знаниями в области программирования, наверняка заметят, что за каждым решением задачи спрятана демонстрация разных вариаций известных парадигм программирования (автоматное и императивное структурное).

В случае затруднений реализации примеров вы можете загрузить готовый проект в формате CoDeSys V2.3. Вы найдете его в разделе публикаций сайта www.CoDeSys.ru.

Используйте POU

В серьезных проектах самой плохой идеей является реализация всего алгоритма непосредственно в главной программе. Для того чтобы проект можно было легко понять и отладить, необходимо использовать подпрограммы. В МЭК это программные компоненты – POU. Даже если вы используете POU только один раз, создание компонента с четко описанным назначением является оправданным. Время на вызов POU крайне мало, и его можно не учитывать. Времени же для отладки компактных “обозримых” компонентов требуется значительно меньше, чем для объемных, алгоритм которых сложно понять. Кроме того, компоненты МЭК решают проблему разделения областей видимости имен переменных. Например, вы можете безопасно объявить свою независимую переменную X в любом компоненте. При отладке CoDeSys использует полные имена переменных, включающие имена компонентов через точку (например, PLC_PRG.x). По умолчанию, PLC_PRG является в CoDeSys главной программой для однозадачных проектов. Но даже главную программу вашего проекта есть смысл сразу заключать в отдельный POU и уже его вкладывать в PLC_PRG. В ходе работы может выясниться необходимость ввести специальную секцию инициализации либо главный аварийный выключатель, либо несколько параллельных задач. Если ваша программа заключена в отдельный компонент, то добавить управление ее поведением не составит проблемы.

Выбирайте оптимальный тип POU

Типов POU в МЭК 61131-3 всего 3: функция (FUNCTION), функциональный блок (FUNCTION BLOCK) и программа (PROGRAM).

Результат работы функции всегда однозначно определяется значениями ее параметров без учета предыстории работы. Функция не может иметь внутренней памяти и возвращает единственное значение (оно может быть составным). Функцию можно использовать в выражениях, например: *Y := Sin(X) – 1;*. Функция не имеет возможности запомнить этапы своего выполнения (нет памяти) между вызовами, поэтому она всегда выполняется синхронно. Функция не может иметь локальных энергонезависимых переменных. В функции нельзя использовать язык SFC. Если это возможно, то используйте функцию, в противном случае – функциональный блок.

Функциональный блок представляет собой подпрограмму, объединенную с собственной структурой данных. Как и при объявлении структуры, создание нового функционального блока создает только новый тип. Для практического использования необходимо объявить экземпляр соответствующего функционального блока. Функциональный блок может иметь произвольное число входных и выходных переменных, доступных извне (интерфейс компонента). Внутренние переменные не доступны другим компонентам (к отладчику это не относится). Объявление дополнительного экземпляра функционального блока приводит к выделению памяти в области данных. Исполняемый код для разных экземпляров один и тот же.

Программа в CoDeSys аналогична функциональному блоку, но не имеет экземпляров. Программа – это глобальный объект. Традиционно программы применяются в CoDeSys для очень крупных программных модулей в многозадачных проектах.

Окружайте компонент линией обороны

При создании нового POU необходимо сразу уделить внимание его входным и выходным переменным. Продумайте их состав и типы данных. Постарайтесь учесть возможные способы применения POU, не задумываясь пока о внутреннем устройстве. Должен ли POU иметь вход разрешения работы (по уровню или фронту), выходы готовности и диагностики, возможность расширения области определения данных? Если есть сомнения, то лучше сразу предусмотреть избыточные возможности. Как и всегда, особое внимание нужно уделить выбору типов данных.

Интерфейс – это единственный доступ к компоненту извне. Нарушить работу компонента можно только через его входные данные. Качественно сделанный POU должен включать необходимые ему проверки допустимости значений данных так, чтобы даже самые злостные враги, задав плохие комбинации значений входов, не смогли бы

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

 © НАУЧТЕХИЗДАТ 2006г.

заставить ваш компонент дать абсурдный результат или зависнуть. Это так называемая идеология “защитного программирования”. Весьма вероятно, вы будете аккуратно и правильно использовать свои POU. Но часто, даже в случае аппаратных сбоев защитное программирование обеспечивает ПЛК самостоятельный выход из кризисной ситуации. Правильный выбор типов данных позволяет избежать лишних проверок. Используйте типы данных, которые принципиально не могут содержать опасные для компонента значения. Для этого в CoDeSys есть целые без знака (USINT, UINT, UDINT) перечисления, переменные с ограниченным диапазоном значений (пример: *ADCInt : INT (-4095..4095);*). Избежать повторения сложных объявлений вам помогут псевдонимы типов.

Как вы уже знаете, редакторы CoDeSys умеют автоматически объявлять переменные. Однако интерфейс компонента слишком ответственен. Правильнее всегда описывать его “руками” до того, как перейти к разделу реализации.

Задача №1

Задача такова: в разных концах коридора установлены два выключателя с одним замыкающим контактом в каждом (Swithc1 и Switch2). Контакты подключены на входы ПЛК. Выход ПЛК включает освещение. Необходимо сделать так, чтобы переключение любого выключателя позволяло включить или выключить свет при любом положении другого. Главный общий выключатель (Main Switch) также подключен к входу ПЛК. Задача заимствована из программы базовых учебных курсов 3S-Software.

Решение 1

Многие могут сказать, что задача №1 столь элементарна, что решается без применения ПЛК (например, как показано на рис. 1). Но это неточно, поскольку мы имеем выключатели, а не переключатели. То есть для практической реализации схемы нам понадобятся дополнительные реле, а это уже классическая область применения ПЛК. В любом случае схема на рис. 1 дает нам очевидный метод решения задачи. Безусловно, реализация на языке релейно-контактных схем (РКС, он же LD) будет самым наглядным решением.

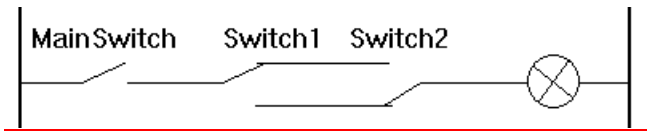

Рис. 1. Схема управления освещением

Реализуем решение в виде отдельного программного компонента. Мы имеем три входных переменных и один выход. Очевидно, все они должны иметь тип данных BOOL. Значение выхода здесь всегда однозначно определяется значением входов. Мы имеем дело с комбинационной логической схемой. Внутренняя память не нужна. Решить эту задачу можно практически моментально, растягивать вычисления во времени и предусматривать асинхронное выполнение смысла нет. Соответственно нет необходимости вводить выход готовности. Исходя из этого, будем использовать функцию.

Добавим новый объект в наш проект (Project, Object, Add). Назовем его “Example2”. Вариант реализации, соответствующий схеме на рис. 1, показан на рис. 2. Правила ввода LD диаграмм вы найдете в руководстве по программированию CoDeSys и в интерактивной подсказке.

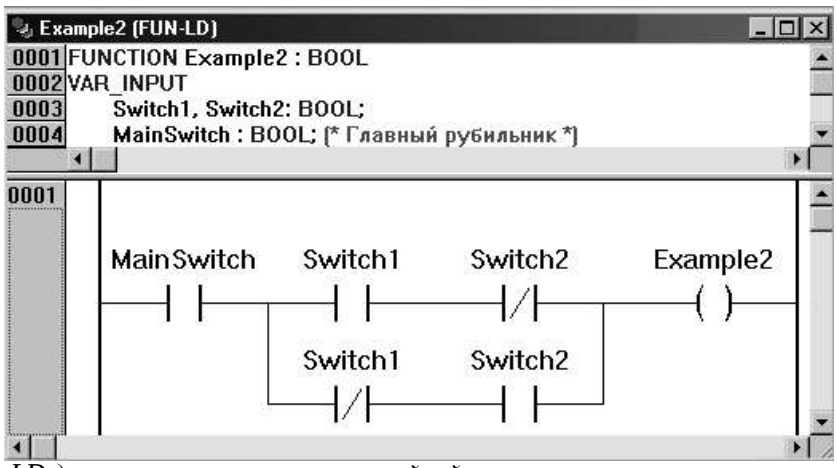

Рис. 2. LD диаграмма, полученная из релейной схемы

Для проверки работы нашей функции включите ее вызов в программу PLC_PRG (мы создали ее на языке ST).

Например, так:

```
PROGRAM PLC_PRG
VAR
    res, sw1, sw2, sw_main: BOOL;
END_VAR
res := Example2(sw1,sw2,sw_main);
```

Работа с отладчиком LD

Перейдите в режим *online*. Поменяйте значения переключателей и выполните один цикл или запустите программу в реальном времени. Вы сразу обратите внимание на то, что CoDeSys выделяет ярким синим цветом (на черно-белом рис. 3 жирные линии) активные соединения и контакты. То есть те, которые передают значения TRUE. Рассмотренные в первой части статьи [1] инструменты отладки полностью применимы и для LD диаграмм.

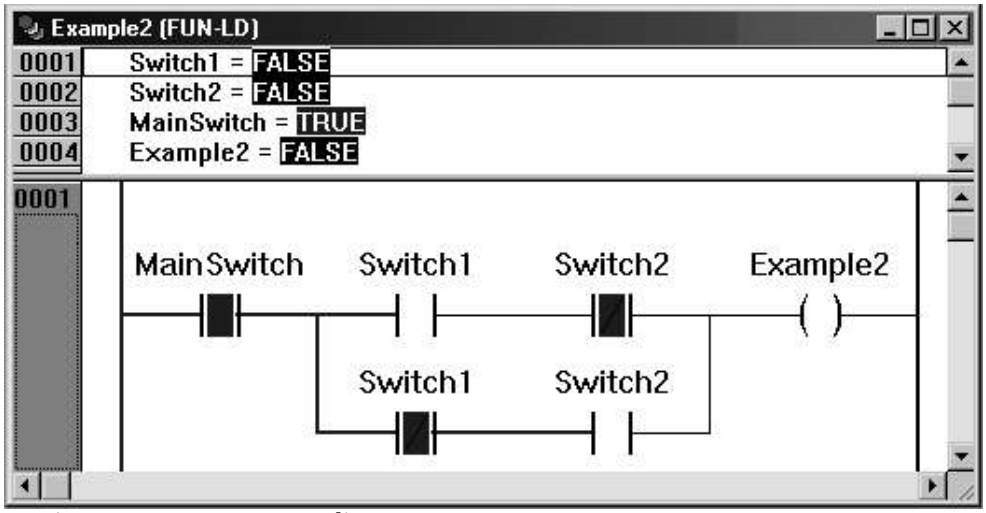

Рис. 3. LD диаграмма в режиме online

На диаграмме вы не увидите определений переменных и, естественно, комментариев к ним. Но достаточно подвести указатель мыши к любому контакту, и CoDeSys покажет всплывающую подсказку с определением переменной. При анализе сложных схем удобно включить режим отображения объявлений переменных в каждой цепи (рис. 4): *Extras, Options, Show variables comments per rung in printout*. Для изменения значений переменной, связанной с контактом, достаточно щелкнуть мышкой по контакту. Как обычно, изменение запоминается, но не передается в ПЛК до команды записи или фиксации значений переменных (рис. 4).

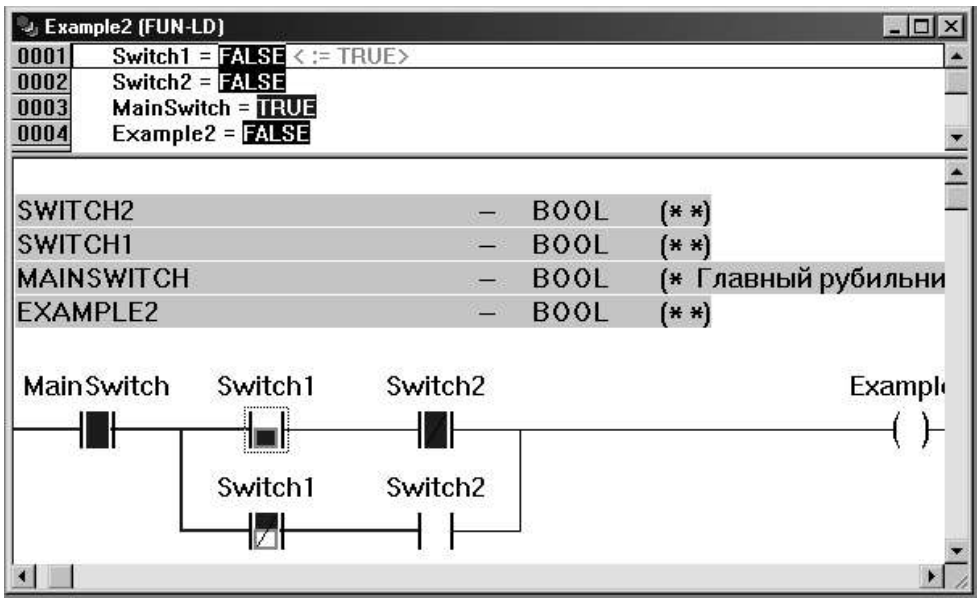

Рис. 4. Модификация значений переменных в LD

CoDeSys позволяет снабжать LD цепи комментариями (рис. 5). Не забывайте об этой полезной возможности.

Решение 2

Можно взглянуть на данную задачу по-иному. Давайте полностью забудем о схеме на рис. 1. Вернемся к исходному условию и составим алгоритм. Если главный выключатель выключен, то следует выключить свет. Иначе, если выключатели находятся в разных положениях, то следует включить свет. Для проектирования таких алгоритмов применяются блок-схемы.

Для программирования блок-схемы достаточно иметь возможность ставить в программе метки и осуществлять переходы (он же jump, он же goto) на них. Эту возможность имеют все МЭК языки. В языке LD метку можно поставить на любую цепь. Переход можно связать только с обмоткой реле. Соответствующее решение задачи №1 показано на рис. 5.

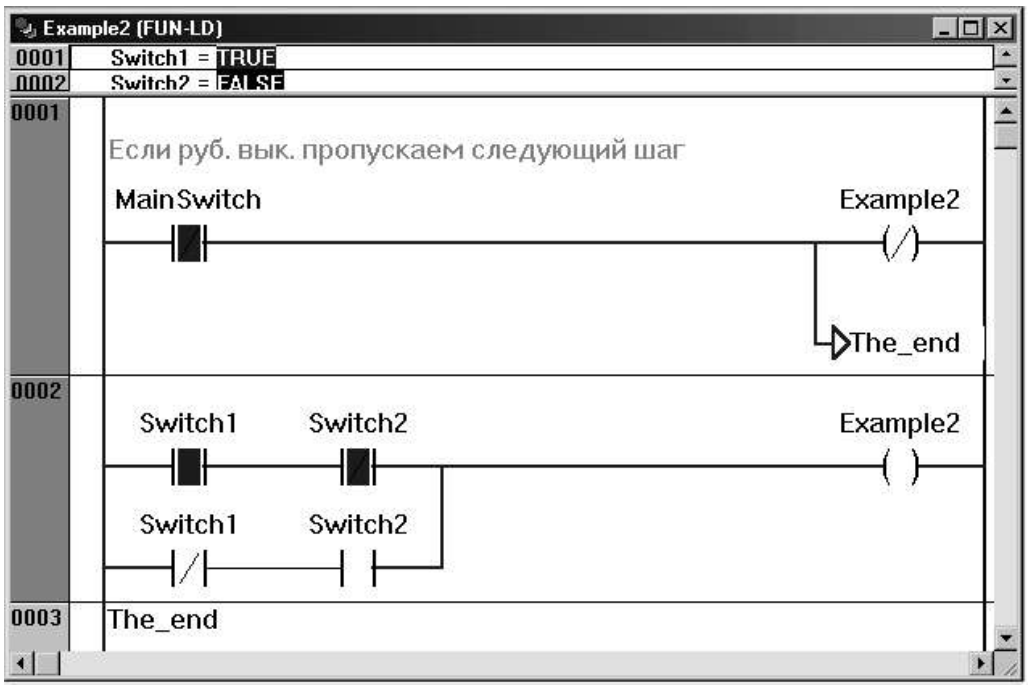

Рис. 5. LD диаграмма, полученная из блок-схемы

Отладка подобных программ в CoDeSys имеет одну тонкость, точнее паразитный эффект.

Посмотрите внимательно на рис. 5. Ранее выход уже был включен, соответствующая ветвь цепи 2 выделена (жирным) синим цветом. Позднее мы выключили главный выключатель. Теперь цепь 2 не активна, то есть она вообще не выполняется. Вопрос, как правильно отображать ее состояние в отдатчике, спорный. CoDeSys показывает состояние проводников пропущенной цепи таким, каким оно было перед потерей активности этой цепи! Это разумно, поскольку если цепь не выполняется, то она не меняет свое состояние. Пропуск работы цепи и ее отключение – это разные вещи.

Решение 3

Возможно, что однажды в процессе программирования вы вдруг встретитесь с упорным сопротивлением со стороны инструментальной среды. Вам удается добиться результатов, но очень мучительно. Стандартные элементы не имеют нужных возможностей и отказываются стыковаться нужным образом. Создается впечатление, что тысячи пользователей CoDeSys – весьма странные люди, и только вы первый – нормальный. Это 100 % симптом ошибочного выбора метода решения задачи. В этом случае лучше всего начать все сначала, изменить структуру данных или выбрать более подходящий язык реализации. В нашем случае разумно выделить пропускаемые цепи в отдельный POU (AxFunc6) и вставить его в LD цепь, используя вход разрешения работы (EN). Результат показан на рис. 6.

Давайте вернемся к условиям задачи и посмотрим на нее с большей высоты. То есть не только на программу, а на систему целиком, включая внешние цепи.

Данное устройство в целом очень похоже на автомат. Оно обладает двумя состояниями (включено или выключено) и четко определенными условиями переходов между ними. Однако программа на рис. 2 выглядит, как комбинационная схема. В ней нет ни триггеров, ни переменных состояний, вообще никаких элементов памяти, необходимых для программирования автоматов. Тем не менее, это конечный детерминированный автомат. Дело в том, что элементами памяти здесь служат выключатели. То есть, это автомат с внешним кодированием состояний.

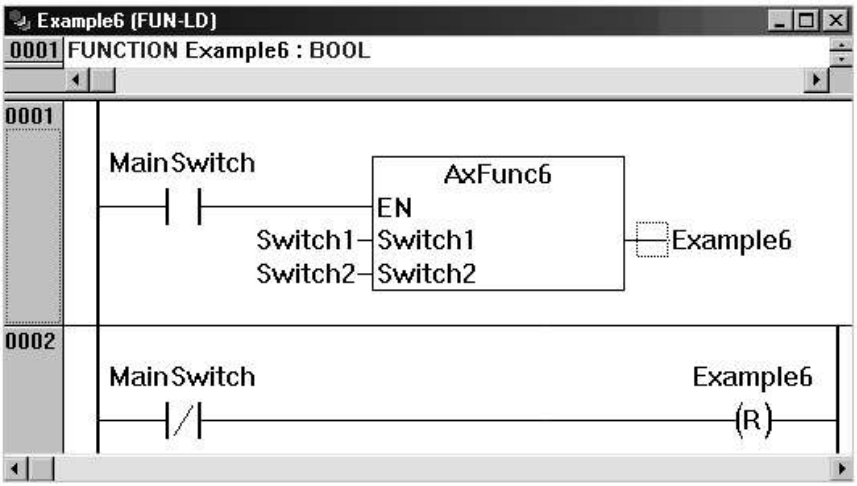

Рис. 6. Автомат с внешним кодированием (LD)

Эта идея иллюстрируется яснее на рис. 6. Здесь явно видны 2 состояния, соответствующие положению главного выключателя. В реальных задачах обычно получается несколько десятков цепей с условиями, разрешающими выполнение соответствующих POU. Забегая вперед, отметим, что такой подход является одним из лучших для автоматизации сборочных автоматов и других механизмов с четко выделенными состояниями. Визуальное отображение LD диаграммы не только дает ясное представление о работе машины, но и здорово упрощает диагностику внешних цепей. В каком бы состоянии не была машина, она сразу по включению начинает работать правильно, без специальных действий по приведению ее в исходное состояние. Не возникает нужда и в энергонезависимой памяти. Структура программы здесь жестко связана со структурой объекта управления. Часто для промышленного оборудования это именно то, что нужно.

Решение 4

Но в нашем примере это не так. Мы всегда должны думать несколько вперед. На практике мы можем прогнозировать следующие сюрпризы: требование добавить еще пару выключателей, требование заменить выключатели кнопками. В обоих случаях это полная переделка программы. Можно ли сразу построить программу более гибко? Конечно, да. Сформулируем задачу шире: любое изменение уровня по любому входу должно изменять состояние выхода.

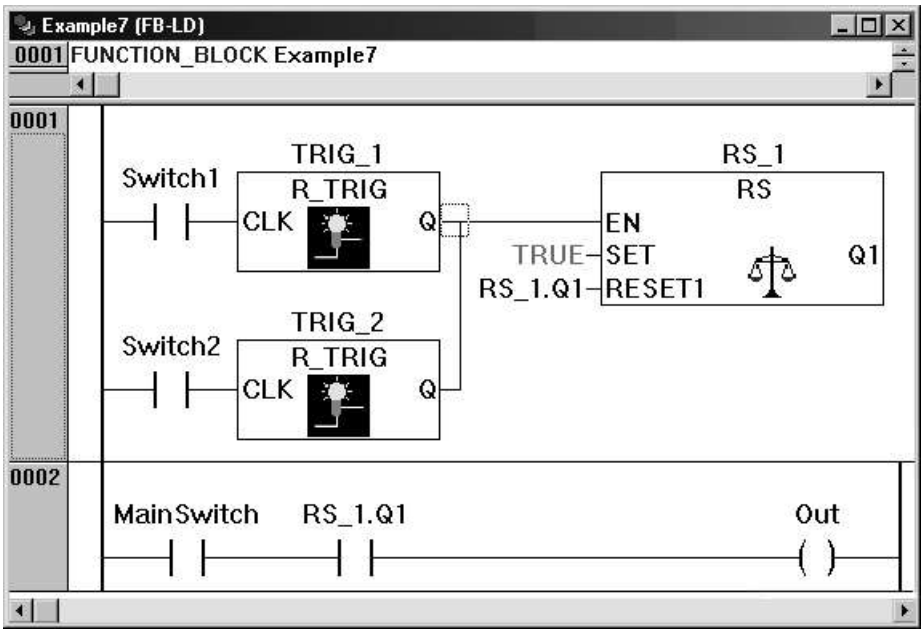

Рис. 7. Автомат с внутренним кодированием (LD)

В аппаратной реализации мы применили бы детекторы импульсов по входам и счетный триггер. Рис. 7 показывает, что применение стандартных функциональных блоков МЭК позволяет прозрачно перейти к эквивалентной программе. Счетный триггер реализуется на основе доминантного переключателя RS с помощью обратной связи.

Одна тонкость: состояние триггера должно сохраняться между вызовами POU. Пора переделать функцию в функциональный блок.

Заключение

В следующий раз мы продолжим изучение методов программирования, поскольку главным образом они и определяют методы отладки. С точки зрения вычислительной мощности все языки МЭК равны. Любую задачу можно решить на любом языке. Например, на языке ST решение 1 записывается в одну строку:

```
res := MainSwitch AND Switch1 XOR Switch2;
```

Решение 2 будет выглядеть так:

```
IF MainSwitch THEN
    res := Switch1 XOR Switch2;
ELSE
    res := FALSE;
END_IF
```

На ST оно выглядит лучше[2].

ST – это самый сложный язык с точки зрения реализации транслятора. Из двух десятков наиболее распространенных систем МЭК программирования (известных автору) только в двух (не считая CoDeSys) он реализован так, что позволяет забыть о практике создания расширенных функциональных блоков внешним С компилятором. К сожалению, в большинстве распространенных МЭК инструментов язык ST “самостоятельного значения не имеет: используется только совместно с SFC” [2].

Наличие в МЭК 61131-3 пяти языков дает свободу выбора и, естественно, приносит постоянные сомнения в правильности этого выбора. Большой опыт работы с конкретным языком, инструментом или определенной методикой проектирования приводит к тому, что некоторые языки кажутся странными, лишними или лишенными практической ценности. Вы обязательно встретитесь с подобного рода обобщениями. Но даже просто из любопытства интересно было бы узнать: какой язык наиболее популярен в мире? Но как получить такую статистику?

Компания 3S имеет службу технической поддержки. В ней работают очень отзывчивые люди. По долгу службы они ежедневно анализируют до 20 чужих ПЛК программ со всего мира. На нашу удачу, специалисты 3S имеют привычку аккуратно фиксировать результаты своей работы.

Продолжение следует.

Список литературы

1. *Петров И.В.* Отладка прикладных ПЛК программ в CoDeSys // Промышленные АСУ и контроллеры. 2006. № 2.
2. *Зюбин В.Е.* Программирование ПЛК: языки МЭК 61131-3 и возможные альтернативы // Промышленные АСУ и контроллеры. 2005. № 11.

[2] Было бы странно, если бы язык ST (структурированный текст) плохо подходил для структурного программирования.

 © НАУЧТЕХИЗДАТ 2006г.