И.В. ПЕТРОВ (Компания “Пролог”)

Отладка прикладных ПЛК программ в CoDeSys (часть 6)[1]

Самые сложные в мире программы – это рецепты из поваренной книги. Как можно солить по вкусу и варить до готовности? Научить этому практически нельзя. Даже очень горький (в прямом смысле) опыт оказывается бесполезен. К счастью, на современных упаковочных пакетах есть нормальные пошаговые инструкции с указанием точного количества компонентов и времени этапов приготовления. В действительности очень многие технологические процессы описываются подобными последовательностями элементарных операций: замена узла в автомобиле, этапы работы стиральной машины, операции сборочного автомата, процессы дозирования, маршруты движения, испытания изделий и т.д. и т.п. Разбираясь в порядке их выполнения, мы интуитивно берем ручку и начинаем ставить галочки, обводить их кружками и иногда даже соединять стрелками. В итоге получается программа, доступная для повторения любым разумным человеком, желающим получить хороший результат проверенным способом. Это именно то, что нужно для ПЛК. Идеальный язык программирования ПЛК должен быть понятен не только программисту, но и любому человеку, умеющему читать. Лучше всего этому требованию среди языков МЭК 61131-3 удовлетворяет язык последовательных функциональных схем – SFC.

Некоторые профессиональные программисты, имеющие многолетнюю привычку выражать свои мысли на “классических” языках программирования, называют SFC сложным в освоении. Специально для них в CoDeSys даже существует упрощенная версия SFC. Считается, что изучение языка SFC лучше начинать именно с него. Нередко SFC называют неполноценным языком, на котором нельзя ничего запрограммировать, поскольку все переходы и действия необходимо писать на других языках. В качестве примеров для пояснения SFC часто используют задачи, близкие к практике и более сложные по описанию и терминологии, чем алгоритмические. Давайте отбросим авторитетные высказывания и составим собственное мнение.

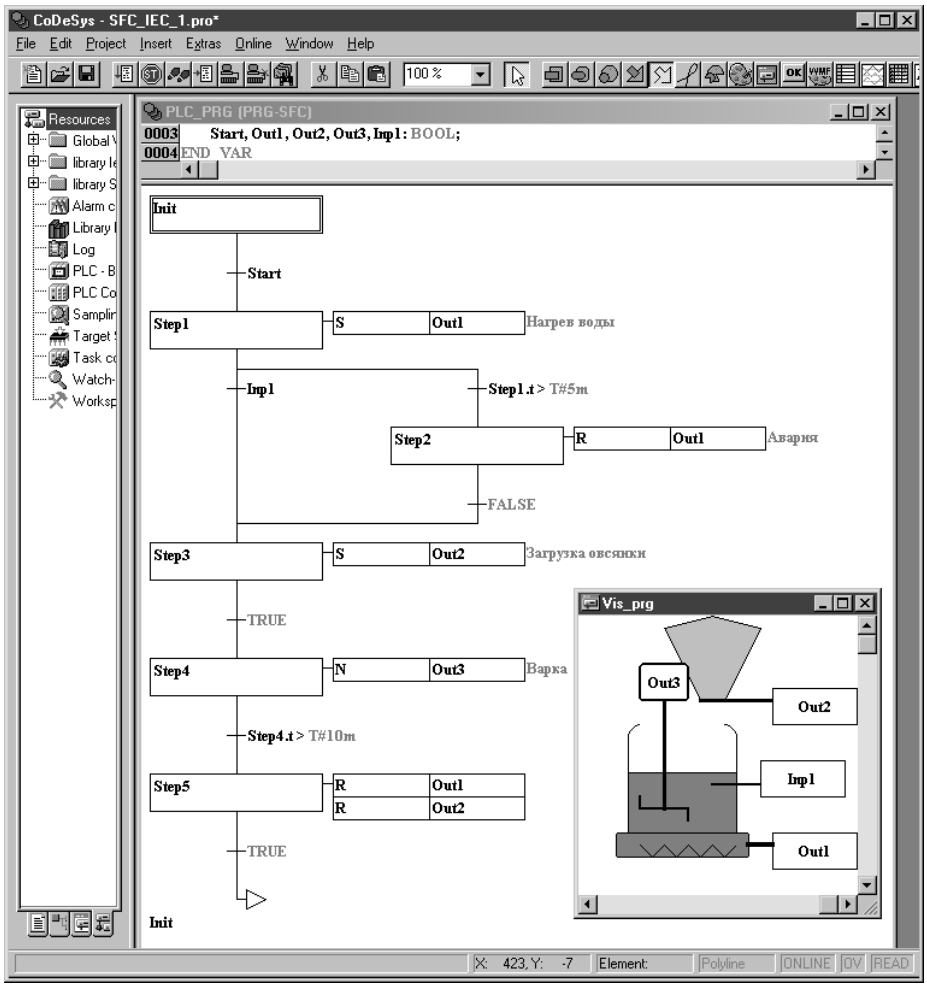

Рис.1 Полное решение примера 1 на SFC

[1] *Продолжение. Начало в № 2-5, 7. 2006 г.*

 © НАУЧТЕХИЗДАТ 2006г.

За примером не нужно ходить далеко. На коробке с овсянкой описана вполне достойная для выражения на SFC технологическая задача. В ней опущены отдельные очевидные для человека детали. Для контроллера их придется запрограммировать. Полный пример решения показан на рис. 1.

Начальный шаг: пока нет команды начинать варку (кнопка Start), нужно просто ждать. На SCF диаграмме начальный шаг расположен вверху и обозначается как прямоугольник с двойной рамкой (рис. 1). От него вниз уходит линия связи с горизонтальной черточкой (как бы закрытый шлагбаум). Это так называемый переход. Ему соответствует определенная логическая переменная или выражение: условие перехода. Когда условие перехода принимает значение TRUE, текущий шаг прекращает работу и передает эстафету (маркер) следующему шагу.

Первый шаг: должен нагреть до кипения литр воды. Для этого он выполняет одно единственное действие: включает нагреватель (выход Out1). Действия, если они есть, показываются в прямоугольниках справа от шага. Они собственно и определяют полезную работу, связанную с шагом. Условием перехода в этом шаге служит включение датчика кипения (Inp1). По нему маркер передается третьему шагу.

Второй шаг: это дань нашему предыдущему опыту программирования контроллеров. Допустим, датчик кипения не сработал после 10 мин нагревания. Налицо аварийная ситуация. Ее обрабатывает шаг 2. Он отключает нагреватель и просто стоит в ожидании рестарта программы. Иным способом уйти из него невозможно, поскольку в условии его перехода задана логическая константа FALSE. Чтобы после первого шага попасть в данный шаг, в программе предусмотрен альтернативный переход, начинающий отдельную ветвь. В альтернативных ветвях маркер может двигаться только по одной из них.

Третий шаг: должен высыпать стакан овсянки. Он открывает соответствующий клапан (Out2) и безусловно (константа TRUE в условии) передает управление дальше.

Четвертый шаг: должен варить 10 мин. Время и будет единственным условием следующего перехода. Варить нужно помешивая. Для этого действие шага удерживает выход мешалки включенным (Out3), пока шаг имеет активность.

Пятый шаг: выключает все выходы, которые ранее были включены, и передает управление начальному шагу. Редактор SFC не позволяет рисовать пересекающиеся или идущие вверх переходы. Для этих целей используется Jump – переход на произвольный шаг (как goto).

Данную схему можно сделать более понятной, если дать осмысленные наименования шагов и переменных. Но даже сделанная на “скорую руку”, она позволяет легко проследить этапы выполнения процесса. В режиме online CoDeSys выделяет цветом активные шаги, еще более упрощая анализ схемы. Рассмотренный выше пример призван продемонстрировать базовые принципы SFC. Сам язык и приемы работы в редакторе подробно описаны в документации на CoDeSys [1]. Далее мы сосредоточимся исключительно на моментах, требующих особого внимания, типовых затруднениях и технике отладки SFC программ.

Тонкости редактирования SFC

Базовым элементом SFC служит связка – шаг и переход. Поэтому в графическом редакторе CoDeSys вы не найдете команд вставки в ветвь одного шага или одного перехода. Вставить можно только шаг вместе с переходом. Аналогичное правило действует и при удалении. CoDeSys не позволит удалить из ветви только один шаг. Для этой цели нужно выделить шаг вместе с переходом. Это логично, хотя и не столь очевидно (рис. 2).

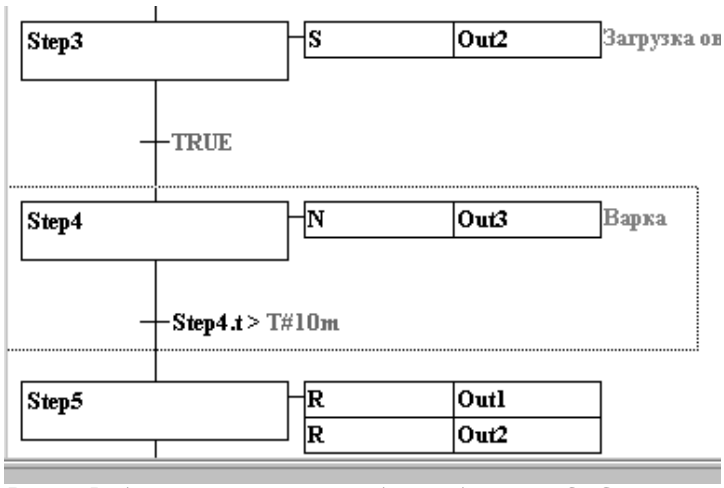

Рис. 2. Выделение шага и перехода в редакторе SFC

Альтернативные ветви всегда требуют разветвленных условий. Поэтому при вставке альтернативной ветви нужно выделить фрагмент, начинающийся и заканчивающийся переходом.

Стандарт МЭК допускает не изображать на диаграммах действия рядом с шагами, а прятать их внутри шага. Такая компактная форма отображения послужила основой упрощенного SFC в CoDeSys, который мы рассмотрим позднее. По умолчанию SFC редактор CoDeSys работает именно в таком режиме. Чтобы он переключился в режим ввода стандартного SFC, необходимо активировать команду *Use IEC-Steps* в меню *Extras*. Кроме того, в проект обязательно должны быть включены библиотеки *Standard.lib* и *IecSfc.lib*.

Условие можно писать непосредственно в диаграмме справа от перехода в виде логического выражения на языке ST. Как правило, эта техника применяется для примитивно простых условий, состоящих из одной логической переменной или константы. Альтернативный вариант состоит в описании условий перехода в отдельном окне. Мы просто щелкаем мышкой по переходу, и CoDeSys автоматически открывает отдельное окно редактора, выбранного нами для описания условия языка. По умолчанию условиям автоматически даются порядковые наименования. Естественно, эти имена можно поменять. Условие пишется практически, как функция типа BOOL, но в нем нельзя присваивать значения переменным. Условное выражение должно быть предельно лаконичным и понятным. Если нужны сложные вычисления, то лучше создать для них отдельное вспомогательное действие. Пример описания условия на LD показан на рис. 3.

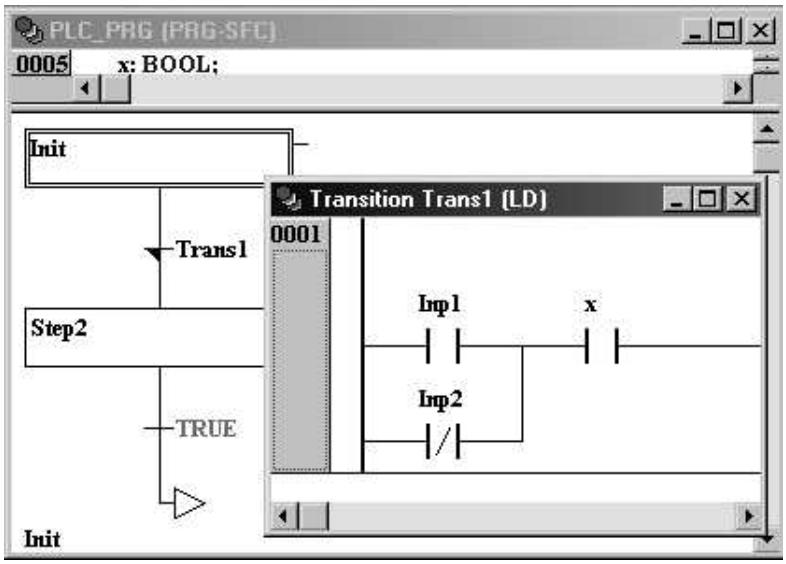

Рис.3.Описание условия перехода на LD

Действия МЭК SFC принадлежат целиком к программному компоненту (POU), а не к отдельному шагу. Поэтому они создаются (команда Add Action) и отображаются в менеджере проекта, под соответствующим POU. Действие может быть написано на любом языке. В том числе на SFC. Чтобы связать действие с шагом, нужно выбрать шаг и дать команду *Associate Action*. Командой *Clear Action/Transition* можно удалить привязку действия к шагу и описание условия из перехода.

Мы уже использовали функциональный блок с действиями в примере третьей части статьи. Возможность добавлять действия в функциональные блоки и вызывать их на любом языке – это уникальное расширение CoDeSys. В других МЭК системах действия применяются только в SFC.

Порядок выполнения шагов SFC

Важнейшее отличие SFC от других языков состоит в порядке выполнения программы. В любом другом языке вся программа полностью выполняется при каждом вызове программного компонента (POU). Если мы хотим пропустить выполнение определенных фрагментов текста в ST и IL или цепей в FBD и LD, то должны самостоятельно запрограммировать ветвления по необходимым условиям. При вызове POU на SFC выполняются только действия, связанные с активными шагами (имеющими маркер). Даже если переход имеет значение TRUE, то следующий шаг начнет работу при следующем вызове POU. Таким образом, если мы пишем на SFC главную программу PLC_PRG, то разрешение переходов будет происходить в каждом рабочем цикле один раз.

При первом вызове SFC компонента маркер всегда имеет начальный шаг. Далее маркеры могут расходиться по параллельным ветвям, делая активными произвольное число шагов. Кроме привилегии получения маркера первым, начальный шаг не обладает абсолютно никакими дополнительными свойствами. Если из некоторой ветви принудительно передать маркер на начальный шаг, то это не будет автоматически обозначать сброс всего компонента в исходное состояние. В самом языке SFC команды для сброса нет. При крайней необходимости это можно сделать при помощи специальных системных флагов CoDeSys, управляющих работой SFC исполнителя. На практике многие программисты предпочитают создавать программу PLC_PRG на ST или LD и вкладывать в нее вызов программ на

SFC. Это позволяет прозрачно отключить выполнение SFC программы. Например, по команде главного выключателя.

Для альтернативных ветвей хорошим стилем является указание взаимоисключающих условий. В принципе, нет запрета на ввод частично пересекающихся условий. Например, мы имеем право задать в одной ветке условие X > 10, а в другой – X > 100. Но в этом случае направление перехода будет зависеть от порядка расположения ветвей на экране. Оценка альтернативных переходов происходит слева направо. Маркер всегда пойдет только по одной ветке. Однако при модификации диаграммы можно легко забыть об этих тонкостях и создать ветвь, которая не получит управление никогда. Поэтому лучше однозначно указать условие для первой ветви: X > AND < 100.

В параллельных ветвях маркер распадается на несколько независимых маркеров, каждый из которых будет работать в своей ветке. Для завершения параллельности должны закончить работу все ветви. После чего маркеры опять сливаются в один, передаваемый последующему шагу. Если одна из параллельных ветвей закончит свою работу раньше других, то ее последний шаг будет продолжать работу до окончания других ветвей, даже если общее условие следующего перехода имеет значение TRUE. Здесь нужно проявлять аккуратность с действиями этого шага. Если в такой ситуации их нужно отключить, то просто добавьте в эту ветвь пустой шаг. Как уже было показано в первом примере, действия не обязательно должны состоять из какого-либо кода на МЭК языках. В простейшем случае вместо действия можно задать логическую переменную. Использование таких переменных для синхронизации параллельных ветвей является стандартным приемом.

Размножение маркера

Крайне аккуратно в параллельных ветвях нужно использовать Jump. Если из одной параллельной ветки сделать переход за пределы параллельных ветвей, то оставшиеся в других ветвях маркеры спокойно продолжат работу. Сверх того, отправленный за пределы параллельных ветвей маркер будет функционировать независимо. Например, на рис. 4 видно, что шаг Step5 отдал свой маркер вверх на начальный шаг. В следующем цикле выполнения этот маркер снова придет в параллельные ветви и повторно разделится.

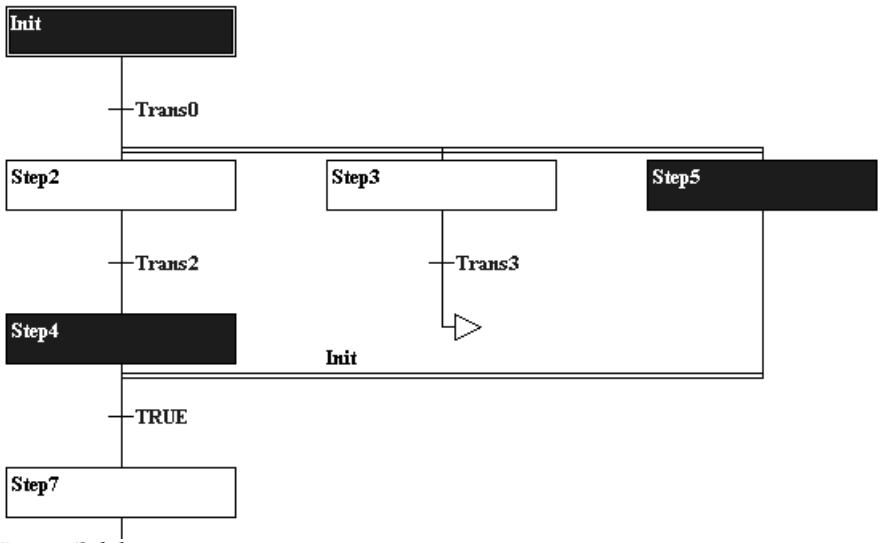

Рис.4. Эффект размножения маркера в параллельных ветвях

Можно придумать ситуации, когда передача маркера между параллельными ветвями или несколькими группами параллельных ветвей не вызовет размножения маркера. Поэтому Jump из параллельных ветвей стандартом МЭК не запрещен. Но анализировать такие диаграммы очень сложно. Лучше всего просто не использовать Jump в параллельных ветвях. Эффект размножения маркера можно получить, если осуществить Jump из вложенного действия, написанного на SFC, в основную диаграмму. Такой прием также нужно исключить, если вы хотите создавать безопасные SFC диаграммы. Размножения маркера в SFC допускать нельзя. Это не вредит работе системы исполнения CoDeSys, но, скорее всего, программа будет работать далеко не так, как было задумано.

Классификаторы действий

На рис. 1 видно, что прямоугольники, включающие наименования действий, имеют левую часть, содержащую букву. Это так называемый классификатор действия. Он определяет, как выполнение действия связано с шагом. Наиболее часто применяются следующие классификаторы:

- N – неохраняемое. Действие вызывается всегда, пока активен шаг. Если вместо действия задать логическую переменную, то она будет иметь значение TRUE, пока шаг активен;
- S – сохраняемое. Действие активируется шагом и остается активным до сброса;

- R – сброс. Деактивация действия. Вместе с S дает механизм, аналогичный set/reset обмоткам в LD;
- P – импульс. Действие выполняется по одному разу при активации и деактивации шага;
- L – ограниченное по времени. Действие вызывается, пока активен шаг, но не дольше заданного времени. Требует указания константы типа TIME;
- SL – сохраняемое и ограниченное по времени. Активируется вместе с шагом и отрабатывает данное время, даже если шаг потерял активность.

Подробно классификаторы действий описаны в [1] и [2]. Для пояснения на рис. 5 показаны графики трассировки активности шага *SL_step* и действия *SL_action* с классификатором SL.

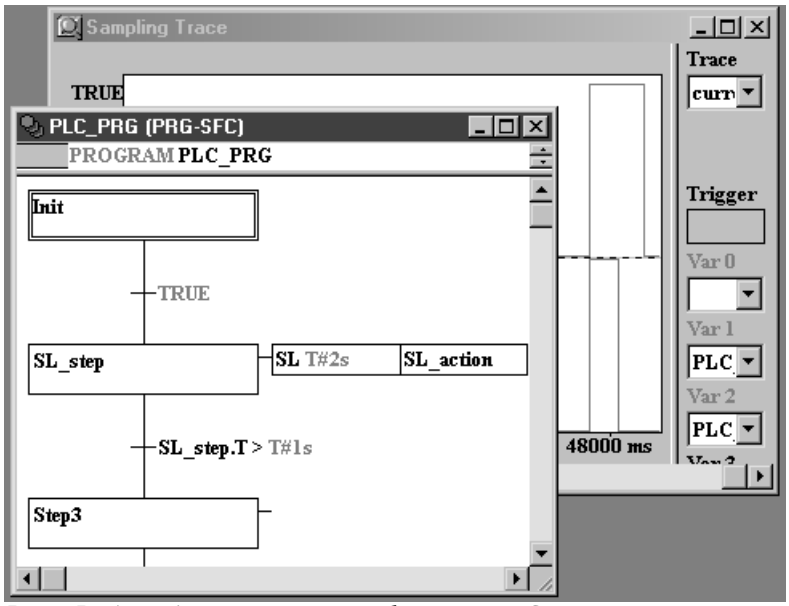

Рис.5.Работа действия с классификатором SL

Устройство SFC исполнителя

Какие шаги будут работать при очередном вызове POU на языке SFC, определяется в процессе работы динамически по положению маркеров. SFC программу невозможно транслировать в машинный код, просто расположив вызов шагов в некотором фиксированном порядке. Каждый шаг обязан иметь внутренний признак активности (владения маркером). По нему система исполнения сможет узнать о необходимости выполнения шага. В результате мы приходим к тому, что для SFC компонента помимо кода нужна оперативная память для запоминания текущего состояния. Сверх того, в систему исполнения должен быть вложен некий автомат, контролирующий выполнение SFC компонентов. Он называется SFC исполнитель. К счастью, он скрыт в системе исполнения CoDeSys SP. Мы могли бы вообще не упоминать об этом, а просто изложить правила работы с SFC без объяснения причин. Но гораздо проще один раз понять, как выполняется SFC изнутри, и не учить правила. Все станет на свои места, и вы будете легко использовать SFC, не обращаясь к справочникам.

Например, поскольку SFC компоненте необходима внутренняя память для запоминания текущего положения маркеров, можно догадаться, что функции (по определению не имеющие внутренней памяти) на SFC писать нельзя. Действительно, для SFC нужно использовать либо программу, либо функциональный блок.

Для каждого шага IEC SFC существуют 2 специальные переменные. Объявлять их не нужно. Они определены неявно и не отображаются в разделе объявлений. Однако доступ из программы к ним открыт. Первая переменная типа BOOL называется *'x'*. Она содержит признак активности шага. Вторая переменная типа TIME называется *'t'*. Она содержит длительность активности шага. Обращение к этим переменным происходит по имени шага, через точку. Например, *'Step2.x'*. Мы уже использовали переменную *t* в первом примере для разрешения перехода по времени работы шага. Это типовой прием в SFC. Переменная *x* применяется не менее широко. Она нужна для синхронизации шагов. Обе переменные предназначены только для чтения. Ими управляет исполнитель SFC. Отладчик CoDeSys позволяет изменить их значения для целей тестирования. Но делать это из программы категорически не следует.

Действия в стандартном SFC работают как вложенные программные компоненты. Их поведение определяется классификаторами и привязкой к шагам. Действия могут включаться, выключаться и даже работать по времени. Из этого следует, что им тоже необходима некоторая внутренняя память. Как и для шагов, работой действий управляет SFC исполнитель.

При вызове POU на SFC исполнитель начинает свою работу с перебора активных шагов. Для каждого из них оцениваются условия перехода, и соответственно изменяются признаки активности. После чего для каждого связанного действия по правилам, определяемым классификаторами, формируется внутренний флажок – выполнять или нет данное действие. После того, как исполнитель разберется со всеми шагами и переходами, он берется за вызов действий. Для каждого из них уже сформированы соответствующие флажки. Остается взять список действий и по порядку вызвать те, которые имеют активный флажок. Порядок вызова действий соответствует тому, как они перечислены в менеджере проекта.

Это не упрощенное изложение, а точное описание порядка работы исполнителя. Из него вполне можно выводить правила.

Первое очевидное наблюдение состоит в том, что шаг не вызывает связанные с ним действия. Он просто управляет их активностью. В результате одно МЭК действие не удастся вызвать дважды при одном цикле работы SFC компонента. Если даже действие включается в нескольких активных шагах, то его вызов произойдет всего один раз. Если в одном из шагов действие включается (S), затем в другом шаге оно выключается (R), то оно вообще не будет вызвано. Таким образом, SFC исполнитель автоматически оптимизирует исполнение программы, не допуская избыточного повторения одинаковых операций.

Второе важнейшее наблюдение состоит в том, что когда переход фактически произошел и шаг уже отдал маркер, то действия шага все же выполняются в последний раз. При необходимости это дает возможность корректно завершить работу действия. Так, непосредственно внутри действия можно обнаружить обнуление переменной *x* и выполнить завершающие операции. Мало того, стандарт МЭК требует обязательного однократного выполнения всех действий шага при его деактивации.

При желании вы можете еще глубже проникнуть в механизмы работы исполнителя. Модуль управления действием реализован в CoDeSys в виде функционального блока *SFCActionControl*. Его экземпляры неявно включаются в каждое действие. Их данные доступны из программы и из отладчика. Реализация самого блока открыта и содержится в библиотеке *IecSfc.lib*.

Заключение

В данной статье мы изучали SFC методом “от сложного к простому”. Действительно, программировать в SFC очень легко. Вряд ли этот язык может вызвать затруднения, но на всякий случай все типовые сложности описаны выше. Если же SFC программа не захочет работать должным образом, в CoDeSys имеются специальные средства.

Продолжение следует.

Список литературы

1. *Руководство* пользователя по программированию ПЛК в CoDeSys 2.3. 3S – Smart Software Solutions GmbH.
2. *Петров И.В.* Программируемые контроллеры. Стандартные языки и приемы прикладного проектирования /Под ред. проф. В.П. Дьяконова. – М.: СОЛОН-Пресс, 2004.