И.В. ПЕТРОВ (Компания “Пролог”)

Отладка прикладных ПЛК программ в CoDeSys (часть 7)[1]

В предыдущей части статьи мы рассмотрели базовые приемы программирования на языке SFC, определенном в стандарте МЭК 61131-3. Это исключительно мощный и выразительный язык. Действия стандартного SFC обладают значительной самостоятельностью по отношению к шагам. Действие может быть активировано в одном шаге и деактивировано в другом, оно может активироваться на заданный интервал времени либо с задержкой. Это напоминает работу бюрократического аппарата. Есть руководитель, который контролирует основные шаги, определяет их цели, признаки завершения и выбирает следующий шаг. Он владеет всей ситуацией и отвечает за слаженность работы. Он не обязан все делать сам и вникать в малосущественные детали. Для этого есть непосредственные исполнители (действия). Их задача состоит в четком выполнении порученного дела, в нужное время и заданный срок. При хорошей организации такой аппарат вне конкуренции по эффективности. Но для его организации нужны определенные усилия и опыт. Гораздо легче выделить этапы выполнения, каждому из которых сопоставить только одну работу. Очевидно, что в этом случае шагов будет больше. Но они получаются проще. Это типовой метод, применяемый обычными людьми в своих повседневных делах. В этом случае мощный аппарат распределения и контроля деятельности оказывается лишним.

Возвращаясь от этой абстракции в SFC, мы приходим к упрощенной модели программирования. Пускай каждый шаг сам выполняет соответствующую работу. То есть пока шаг активен, должна выполняться определенная подпрограмма. Для каждого шага она своя. Дополнительно желательно иметь возможность инициализации и завершения работы шага. В “минимизированном” SFC достаточно иметь одно текущее (основное), начальное и завершающее действия. Для такой упрощенной техники программирования нет необходимости изобретать новый язык. Достаточно в стандартном SFC ограничиться использованием только трех типов классификаторов (N, P1, P0). Однако с целью дальнейшего упрощения CoDeSys позволяет визуально не загромождать SFC диаграмму отображением действий, скрывая их внутри прямоугольника, соответствующего шагу. Поскольку все три действия вложены в один единственный шаг, то при его удалении вместе с ним удаляются и его действия. По вычислительной мощности стандартный и упрощенный языки равны. Выбор упрощенного или стандартного SFC определяется образом мыслей программиста, тем, как ему удобнее выполнить декомпозицию задачи.

Как мы увидим ниже, дополнительный плюс упрощения языка состоит в значительном внутреннем упрощении SFC исполнителя. Классификаторов действий нет, хранить и анализировать их не нужно. Исполнитель упрощенного SFC работает быстрее и расходует меньше памяти. Для ПЛК с низкой производительностью это может иметь существенное значение.

Текущее действие шага в упрощенном SFC вызывается всегда, пока активен шаг. Оно всегда жестко связано только с одним шагом. На диаграмме о его наличии говорит закрашенный верхний угол шага. В противном случае шаг не содержит действия. Для открытия редактора действия в CoDeSys необходимо щелкнуть левой кнопкой мышки по прямоугольнику шага. Как и в стандартном SFC, действия можно описывать на любом языке МЭК.

Начальное или входное действие вызывается однократно при активации шага, перед первым выполнением основного действия. На диаграмме входное действие отображается прямоугольником с буквой “E” (от слова Entry) внутри шага. Выходное действие вызывается однократно после деактивации шага. В этот момент активен уже другой шаг. Если он имеет входное действие, то оно будет вызвано непосредственно следом. На диаграмме выходное действие отображается прямоугольником с буквой “X” (от слова eXit) внутри шага (рис. 1).

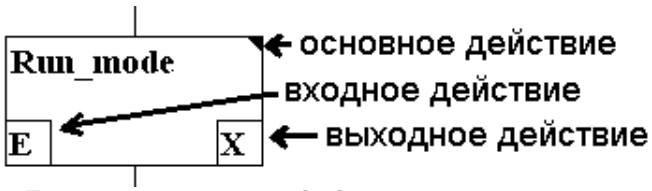

Рис.1 Действия упрощенного SFC

В стандартном SFC также можно использовать действия упрощенного варианта языка. Обратите внимание, что в CoDeSys не поддержаны стандартные МЭК классификаторы действий P1 и P0. В них нет необходимости, поскольку их полностью заменяют входное и выходное действия. В CoDeSys стандартный SFC служит надстройкой над упрощенным языком. Работая в нем, вы в любое время можете включить режим использования МЭК действий. При этом вам не придется переделывать уже написанные и отлаженные фрагменты.

Для иллюстрации упрощенного SFC создадим функциональный блок таймера, полностью аналогичный стандартному блоку TON, но дополненный входом приостановки отсчета времени. В некоторых системах программи-

[1] *Продолжение. Начало в №№ 2-5, 7, 9, 2006 г.*

рования контроллеров подобные блоки присутствуют, но в стандарте МЭК их нет. Обратите внимание, что программные таймеры в CoDeSys отмеряют время по аппаратным часам контроллера. Поэтому даже если мы временно запретим вызов экземпляра функционального блока таймера в программе, то после его разрешения он первым делом обновит свой внутренний счетчик в соответствии с реально прошедшим временем. То есть таким простым способом решить нашу задачу нельзя. Единственный выход – это создание нового функционального блока, способного суммировать интервалы времени.

Работу блока можно описать четырьмя шагами.

1. Инициализация (Init), выполняющая сброс таймера в исходное состояние. Выполняется при переходе значения входа IN в FALSE.

2. Рабочий режим (Run_mode) отсчета времени. Входное действие запускает новый отсчет заданного интервала (или его остатка после паузы) времени с помощью стандартного блока TON. Причем в качестве интервала ему задается остаток времени. Выходное действие запоминает отсчитанное значение на случай продолжения.

3. Окончание счета (End_mode). Это состояние соответствует успешному полному отсчету заданного интервала времени. Когда вызывающая программа соизволит обратить на это внимание и отреагирует снятием входа IN, происходит переход к инициализации. Никаких действий в этом состоянии делать не нужно.

В состояние Пауза (Pause_mode) блок попадает из рабочего режима, если вход PAUSE получает значение TRUE.

Полная реализация примера показана на рис. 2. Вы можете самостоятельно переделать блок в стандартном SFC и попробовать реализовать его в других языках. Это не так просто, как кажется.

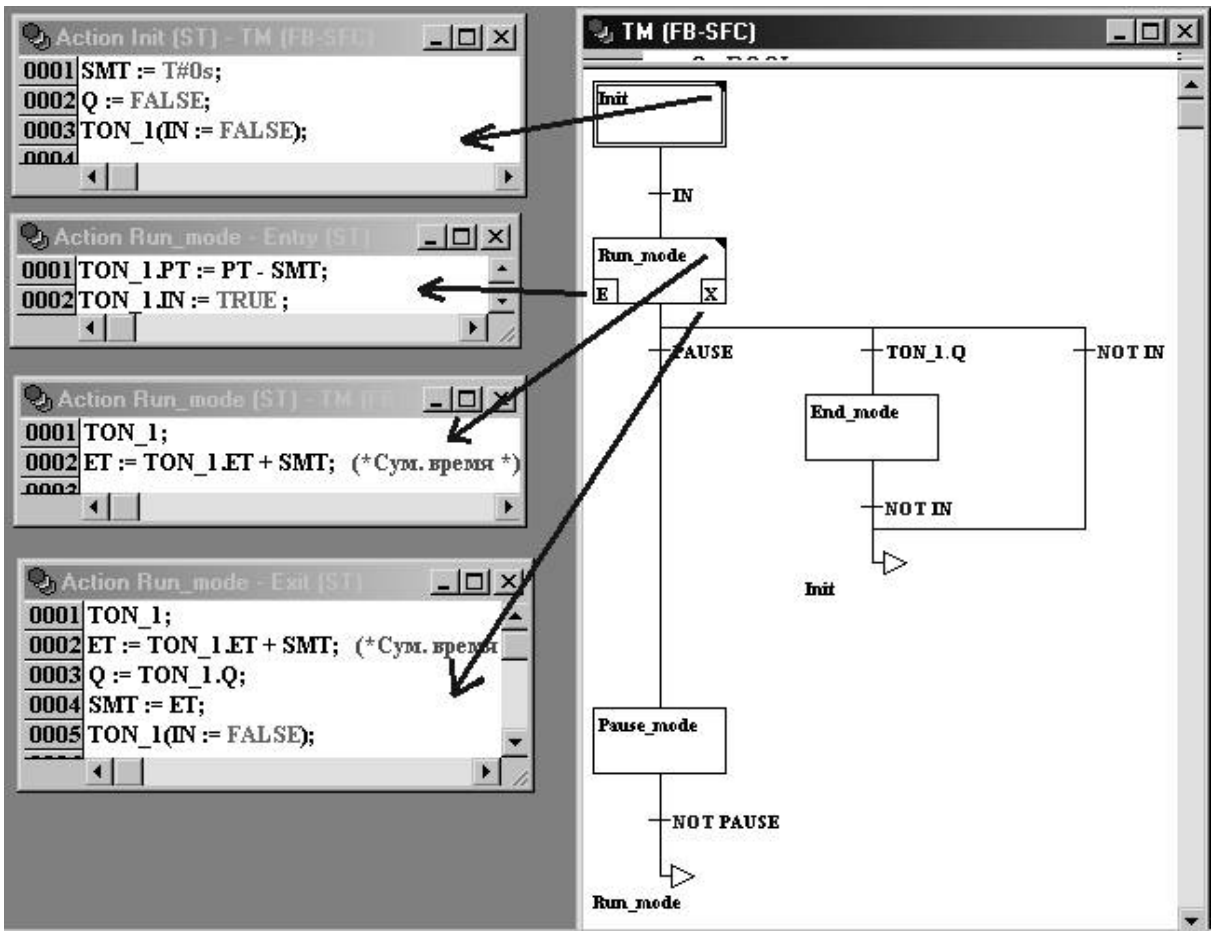

Рис.2 Полное решение примера на упрощенном SFC

Отладка SFC

Как и для любого другого языка, основным средством выявления ошибок в SFC диаграммах является прогон программы в режиме Online. Обратите внимание на следующие моменты.

- Вы можете использовать встроенный эмулятор CoDeSys. Естественно, при этом надо учитывать возможное взаимодействие POU с внешним оборудованием и имитировать его работу, изменяя соответствующие переменные вручную.

• В режиме Online редактор SFC CoDeSys подсвечивает синим цветом шаги и переходы, являющиеся активными. Для простейшей проверки логической правильности работы компонента необходимо запустить программу на выполнение и “пройти” все его ветви путем последовательного изменения значений переменных, участвующих в соответствующих условиях переходов. Поскольку постоянно разрешенных (TRUE в условии) или определяемых задержкой времени переходов, как правило, не много, то SFC компоненты достаточно легко сразу проверять в режиме выполнения в реальном времени.

• Если отдельные шаги выполняются слишком быстро для визуального контроля, то их можно временно замедлить путем задания минимального времени активности или тактируемого выполнения. Оба приема мы далее рассмотрим подробно.

• Для более детального анализа последовательности вызова действий удобно использовать режим выполнения по рабочим циклам.

• Вы можете установить точку останова на нужном шаге или внутри действия. При этом будут доступны обычные Online функции редактора языка, на котором написано данное действие.

• Основная сложность отладки SFC программ состоит в том, что для проверки определенных ситуаций приходится предварительно приводить компонент к соответствующему состоянию. Воспроизвести нужное состояние повторно вручную не так просто. Очень полезным здесь будет уже знакомый нам *Менеджер рецептов* (*Watch and Receipt Manager*). С его помощью не представляет сложности сформировать полный список всех переменных компонента, запомнить их мгновенные значения и восстановить одной командой (“*Extras*” ”*Write Receipt*”). Списки могут включать и неявные переменные (см. ниже), отвечающие за активность шагов и действий (рис. 3).

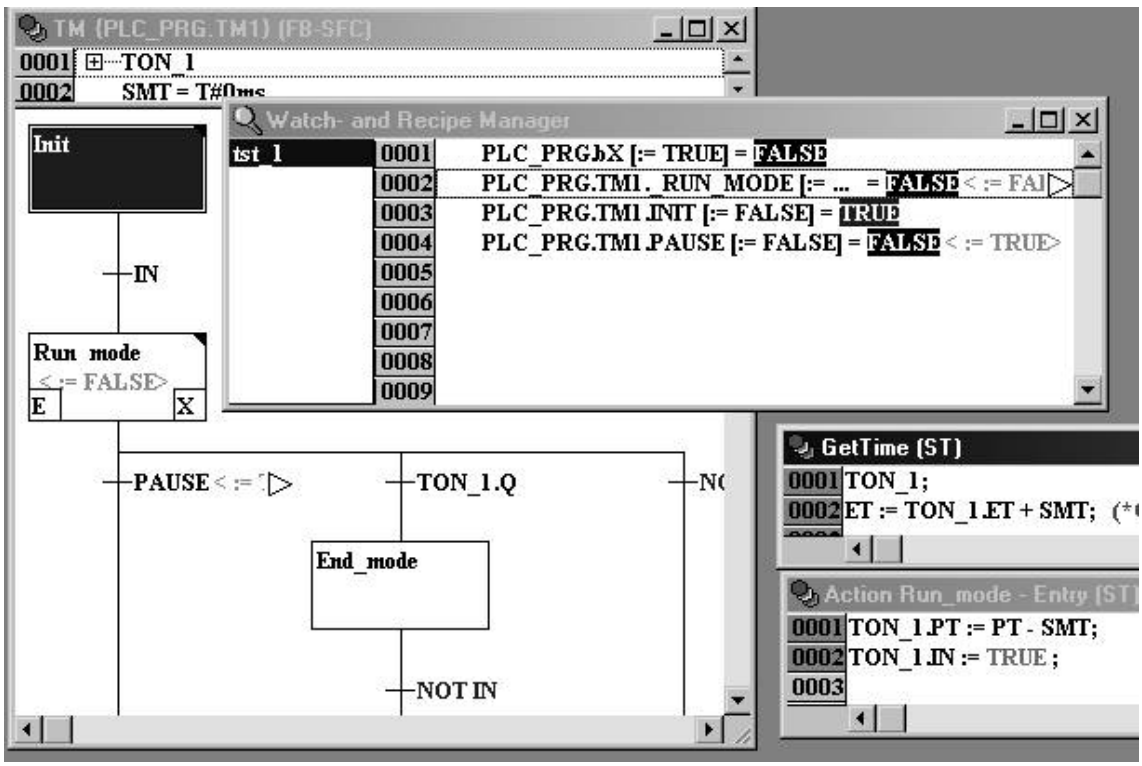

Рис.3 Использование Менеджера рецептов при отладке SFC программ

• При необходимости процесс подготовки к работе можно автоматизировать. Командные файлы (*cmdfile*) позволяют выполнить заданную последовательность команд, включая загрузку нужного проекта и значений переменных из указанного списка, при запуске CoDeSys. Поместив соответствующие ярлыки на рабочий стол, мы можем создать несколько вариантов запуска среды программирования с автоматическим приведением контроллера в требуемое состояние. Это также удобно для проверки оборудования на этапе сопровождения. Описание команд cmdfile вы найдете в справочной системе CoDeSys.

• Наиболее сложными для обнаружения являются динамические ошибки, возникающие только при выполнении в реальном масштабе времени. Для локализации подобных ошибок приходится моделировать определенные последовательности воздействий синхронно с работой компонента. Самым естественным и гибким способом такого моделирования является программирование непосредственно в контроллере. Идея проста. Необходимо написать вспомогательный программный компонент, который будет воздействовать на испытуемый и контролировать его работу. Для этого не всегда достаточно стандартных элементов языков МЭК, нужны специальные средства, позво-

ляющие грубо вмешиваться в работу POU так, как это делает отладчик. Такие средства для SFC в CoDeSys есть, но предназначены для людей, хорошо понимающих их назначение.

Неявные переменные

Мы уже имели дело с неявными переменными в стандартном SFC. Это признак активности шага “x” и время его работы “t”. Если заглянуть глубже, то можно заметить еще две переменные с аналогичными названиями, но начинающиеся с символа подчеркивания. Каждое МЭК действие имеет в CoDeSys аналогичный набор переменных плюс свой экземпляр управляющей структуры *SFCActionControl* (рис. 4).

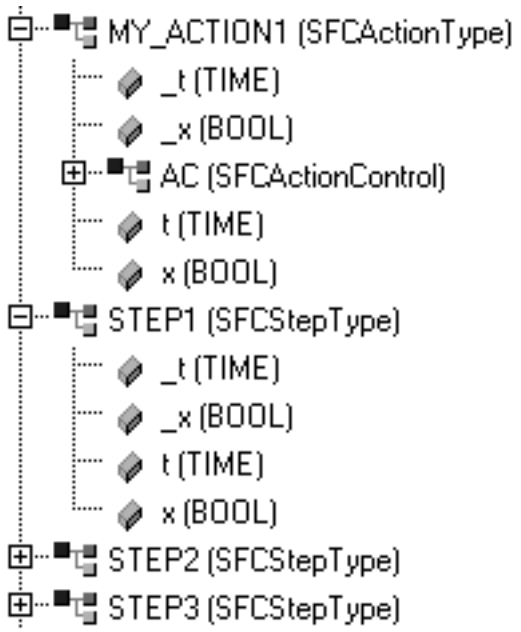

Рис.4 Дерево управляющих структур SFC исполнителя

В упрощенном SFC на каждый шаг неявно объявляется логическая переменная, отвечающая за активность шага. Ее имя совпадает с названием шага: <*StepName*>. Еще одна переменная имеет имя, начинающееся с символа подчеркивания: <*_StepName*> . Ее значение изменяется на один цикл вызова раньше. Например, когда шаг получает активность, переменная <*_StepName*> сразу приобретает значение TRUE, а <*StepName*> повторяет ее с запаздыванием. Аналогично при деактивации шага. Комбинация значений этих переменных образует четыре возможных состояния шага: пассивен, активация, активен, деактивация (рис. 5). Принудительно задав соответствующую комбинацию при отладке, мы можем выполнить входное, основное или выходное действие. Как вы могли заметить, устройство исполнителя упрощенного SFC существенно проще. Переменные времени шагов и структуры управления действиями для работы упрощенного SFC не нужны.

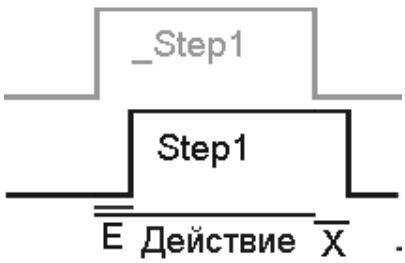

Рис.5 Диаграмма состояний шага

Но если включить механизм контроля времени выполнения шага, описанный ниже, то CoDeSys все же создает внутреннюю переменную для отсчета времени его активности. Имя переменной образуется так: *_time<StepName>*. По умолчанию она не видна. Чтобы получить к ней доступ, ее нужно объявить. В нашем примере (рис. 2) объявление для шага *Run_mode* будет выглядеть так: *_timeRun_mode : TIME*.

Переменные с символом подчеркивания используются для работы внутренних механизмов системы исполнения. Тем не менее, доступ к ним не закрыт. Опытные программисты могут использовать их в отладочных целях.

В стандартном SFC переменные “t” содержат длительность активного состояния. Переменные “_t” хранят абсолютное время момента активации. Это необходимо для поддержки механизма отложенных действий. Как мы уже

отмечали, в стандартном SFC переменные “x” можно использовать в программах для целей синхронизации переходов. В упрощенном SFC для этого служат переменные <*StepName*>.

Вы можете выполнять графическую трассировку неявных переменных SFC с целью динамического анализа работы диаграммы. Встроенная трассировка CoDeSys позволяет отслеживать изменения активности шагов и действий по рабочим циклам синхронно с выполнением.

В диагностических целях можно изменять неявные переменные либо средствами отладчика (рис. 3), либо программно. Однако не используйте неявные переменные на запись в рабочих программах. Это нарушает связь реального хода выполнения шагов с визуальным отображением на диаграмме. Таким образом, SFC диаграмма теряет смысл. Грамотно реализованный программный компонент на SFC может иметь несколько уровней вложений. Отдельные действия могут переставлять собой вложенные диаграммы. Компонент может вызывать другие POU. Но в любом случае последовательность работы компонента должна выражаться средствами языка SFC без применения трюков с неявными переменными.

Атрибуты шага

Наиболее очевидным признаком того, что программа на SFC выполняется не так, как нам нужно, является подозрительно долгое выполнение определенных шагов. Это бывает вызвано как программными ошибками, так и проблемами с внешним оборудованием. Точный контроль тайм-аутов шагов в CoDeSys осуществляется с помощью атрибутов шага. Для выбранного шага дайте команду “*Extras*” “*Step attributes*”. В диалоге атрибутов шага задайте максимальное допустимое и/или минимально необходимое время активности шага. Время работы шага можно наблюдать визуально в режиме Online (рис. 6). Для этого в опциях SFC редактора (“*Extras*” “*Options*”) в разделе *Display at steps* должен быть включен режим отображения *Time Limits*. Команда “*Extras*” “*Time Overview*” позволяет открыть окно, содержащее полный список шагов диаграммы с указанием их временных пределов. Это удобно при работе с объемными SFC диаграммами.

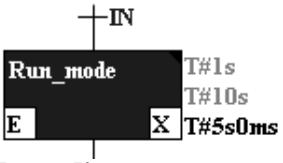

Рис. 6 Контроль времени работы шага в Online

Механизм контроля времени активности шагов встроен в систему исполнения контроллера. Указание минимального времени приведет к тому, что шаг будет активен заданное время, если даже последующий переход уже разрешен. В упрощенном SFC этот простейший прием часто используется для формирования программных задержек по времени. С позиции упрощения переносимости МЭК программ более правильно использовать стандартные функциональные блоки таймеров. Это решение более гибкое, поскольку позволяет изменять время паузы программно. Задание максимального времени шага не влияет на условия переходов. Но если шаг остается активным больше заданного времени, то система исполнения CoDeSys возбуждает специальный флаг ошибки по тайм-ауту. Он доступен программно.

Флаги SFC

В отличие от неявных переменных, по умолчанию флаги SFC не доступны. Флаг необходимо объявить в разделе объявлений SFC компонента. После чего его можно использовать наравне с обычными переменными.

Для контроля времени шагов применяются следующие переменные-флаги:

- *SFCError mun BOOL*. Переменная принимает значение TRUE, если один из шагов POU превысил заданный в атрибутах шага тайм-аут. Обнаружение ошибки не влияет на ход выполнения программы. Значение TRUE остается до его сброса с помощью SFCQuitError. Если сброс вовремя не сделан, то следующие ошибки по тайм-ауту не будут обнаружены.
- *SFCQuitError mun BOOL* (доступна на запись). Установка этой переменной в TRUE сбрасывает ошибку и приостанавливает выполнение POU. После возврата значения в FALSE выполнение возобновляется.
- *SFCEnableLimit mun BOOL* (доступна на запись). Если данная переменная объявлена и имеет значение FALSE, то контроль тайм-аутов будет отключен. Значение TRUE разрешает контроль.
- *SFCErrorStep mun STRING*. В эту переменную записывается имя шага, в котором обнаружен тайм-аут.
- *SFCErrorPOU mun STRING*. Содержит имя компонента, в котором обнаружен тайм-аут.

Выполнить контроль исполнения POU позволяют флаги:

- *SFCTrans mun BOOL*. Принимает значение TRUE в рабочем цикле вызова SFC POU, когда его состояние изменяется. Другими словами, разрешается один или более переходов.

- *SFCCurrentStep mun STRING*. Содержит имя активного шага. В случае выполнения параллельных ветвей включает имя самого правого шага диаграммы.

Следующие флаги дают возможность изменить работу SFC компонента:

- *SFCInit mun BOOL* (доступна на запись). Если записать в эту переменную значение TRUE, то программный компонент сбрасывает все флаги ошибок, переходит на шаг Init и останавливается перед его выполнением. После записи значения FALSE, работа компонента будет продолжена. Обычно этот флаг применяется для сброса SFC компонента из вызывающей его программы. Для этого удобно объявить переменную SFCInit как входную.
- *SFCReset mun BOOL* (доступна на запись). Аналогична по смыслу SFCInit. Единственное отличие состоит в том, что выполнение приостанавливается после шага Init. Ее применение дает лазейку для выполнения сброса компонента изнутри. Вы можете взвести этот шаг в любом действии и сбросить его в FALSE в шаге Init.
- *SFCPause mun BOOL* (доступна на запись). Выполнение SFC компонента приостанавливается, пока эта переменная имеет значение TRUE. Естественно, разрешить выполнение компонента можно только извне, поскольку в режиме паузы ничего сделать нельзя.

Все перечисленные выше флаги предназначены только для чтения, если не указано иное. Никаких иных ограничений на способы их использования нет. Наиболее часто в практических программах встречается флаг *SFCInit*. Программисты, не имеющие соответствующего опыта, всегда начинают работу с написания и отладки основных алгоритмов, откладывая обработку нестандартных ситуаций. Добавить в разветвленную SFC диаграмму переходы на начальный шаг и полноценную инициализацию внутренних переменных исключительно сложно. Здесь на помощь и приходит флаг сброса. Это очень удобный прием, но злоупотреблять им не стоит. Столь сильный механизм контроля и управления SFC, как вышеописанные флаги, прежде всего предназначен для диагностики и обработки ошибок исполнения. Его наличие является данью ориентации CoDeSys на использование профессиональными программистами. Даже если на практике нет времени на переделки, всегда стоит обращать внимание на затруднения в программах и продумывать, как их избежать в дальнейшем без использования нестандартных средств. Это улучшит переносимость ваших программ и упростит их последующее сопровождение.

Заключение

В этой статье мы бегло рассмотрели упрощенный язык SFC. Он действительно очень прост в сравнении со стандартным вариантом. Тем не менее позволяет создавать красивые и выразительные программы. Освоение языка SFC требует больших усилий, чем других МЭК языков, но результат стоит того. Как правило, программисты, не имеющие опыта работы с МЭК языками, начинают с упрощенного SFC и постепенно осваивают стандартный вариант. Благодаря четко выраженной последовательности работы, отладка программ на SFC обычно не вызывает затруднений даже у новичков. Специальные расширения CoDeSys дают возможность оснастить программу средствами самодиагностики. Флаги и неявные переменные позволяют определить, в каком программном компоненте и на каком именно шаге возникла проблема. Остается только подумать и определить причину ее возникновения. Здесь будет полезен еще один нехитрый прием.

Продолжение следует.