И.В. ПЕТРОВ (Компания “Пролог”)

Отладка прикладных ПЛК программ в CoDeSys (часть 4)[1]

В двух предыдущих частях статьи мы сосредоточили внимание на методах программирования в МЭК 61131-3. Нам еще предстоит рассмотреть технику создания многозадачных проектов в CoDeSys и средства управления процессами в прикладных программах. Прежде чем перейти к этой теме, давайте остановимся подробнее на некоторых полезных приемах и инструментах отладки.

В мировой истории существует немало примеров ошибок в программах авторитетнейших компаний, приводивших к катастрофическим последствиям [1]. Ошибки допускают даже опытные программисты, поэтому у вас не должно быть какого-либо комплекса по этому поводу. Отладчик – это лишь инструмент, помогающий понять, что происходит. Выявление и анализ условий возникновения ошибки представляют собой творческий процесс.

Основную массу дефектов в контроллерных программах удается выявить простейшими средствами, описанными в первой части данного цикла статей. Это мониторинг значений переменных, фиксация их значений, останов и выполнение кода по рабочим циклам. Но на практике нередки ситуации, когда для выявления дефекта приходится проводить полноценное расследование с применением специальных приемов.

Давайте отложим анализ типов ошибок и причин, их вызывающих. Обратимся к практике. Допустим, в какой-то момент мы заметили, что наш ПЛК ведет себя несколько странно, совсем не так, как должен. Для примера предположим, что мы включаем в программе дискретный выход № 13, но он остается выключенным. Этого быть не может, но факт. Пора начинать детективное расследование, анализ улик и версий.

Версия 1. Выход неисправен

Для проверки исправности выхода ПЛК достаточно в режиме on-line остановить программу (режим стоп) и изменить значение выхода. Большинство контроллеров в CoDeSys поддерживают работу со встроенным конфигуратором ПЛК. С его помощью в режиме on-line можно наблюдать значения входов и изменять значения выходов. Это позволяет проверить исправность контроллера и внешних цепей системы даже без написания какой-либо программы.

Версия 2. Выход выключается в другом месте

Допустим, выход № 13 исправен. Открыв окно редактора в режиме on-line, мы видим команду:

```
bOut_13 := TRUE;
```

Ее смысл не вызывает сомнения. Программа запущена на выполнение, но выход № 13 упорно не желает включаться. Мониторинг значения переменной дает **FALSE** (рис. 1). Напомним, что значение переменной читается отладчиком в конце рабочего цикла. Отсюда можно предположить, что в некотором другом месте выполняется команда выключения выхода, перезаписывающая правильное значение. Как это проверить?

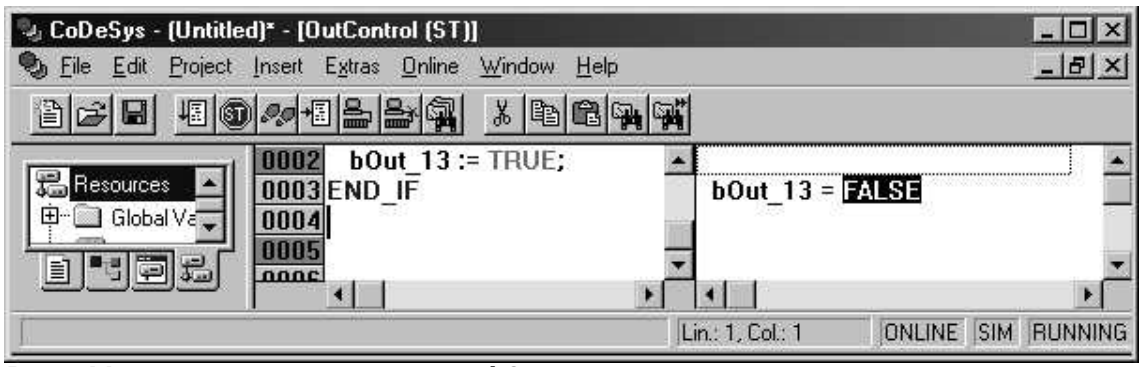

Рис. 1. Мониторинг значения переменной bOut_13.

Здесь может помочь глобальный поиск по имени переменной во всех компонентах проекта. В CoDeSys выполните команду “Project”, “Global Search”. В диалоге объектов поиска выберите все либо только “подозреваемые” объекты (рис. 2) и введите название искомой переменной. CoDeSys последовательно будет открывать соответствующие редакторы и указывать места, где использована данная переменная.

[1] Продолжение. Начало в № 2-4, 2006 г.

 © НАУЧТЕХИЗДАТ 2006г.

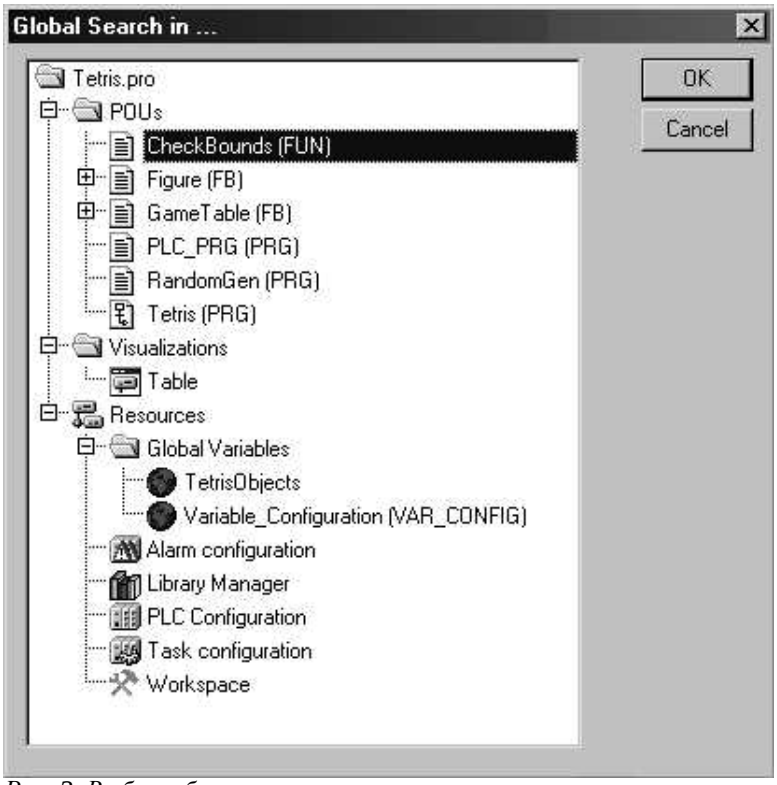

Рис. 2. Выбор объектов поиска.

В CoDeSys есть и более удобный специализированный инструмент для подобных случаев (рис. 3). Команда “Project”, “Show Cross Reference” открывает диалог контроля перекрестных ссылок. Введя интересующую нас переменную, мы можем проконтролировать, в каких POU и с каким типом доступа она используется. Для работы данного инструмента проект должен быть откомпилирован.

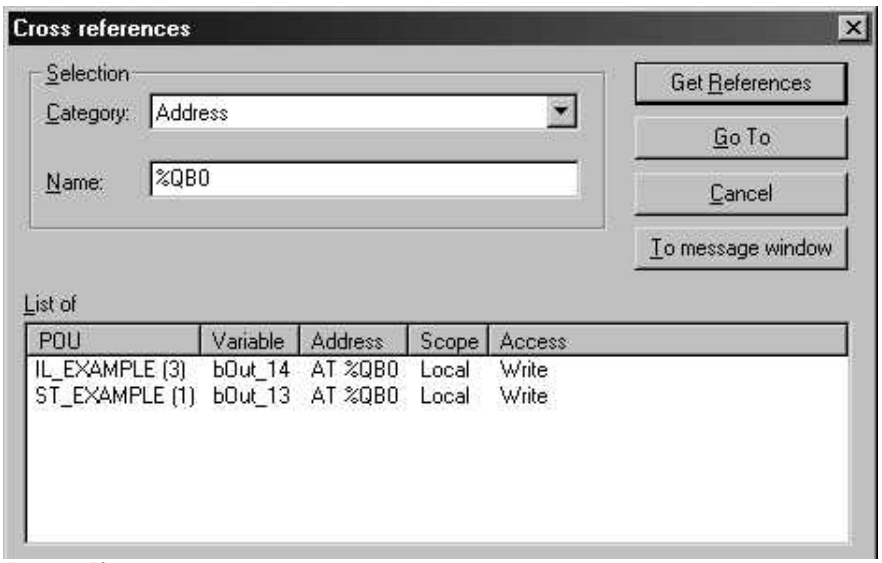

Рис. 3. Контроль перекрестных ссылок.

В изменении значения одного выхода в разных местах проекта нет ничего страшного. Но, в силу возможных паразитных эффектов, лучше избегать такого приема. Включите в опциях генератора кода проекта (“Project”, “Options”, “Build”) флажок “Multiple write access on output”. Теперь CoDeSys будет автоматически сообщать о таких фактах при каждой компиляции.

Здесь же присутствует еще несколько полезных флажков, ужесточающих контроль кода в сомнительных ситуациях. Например, “Unused Variables” включает сообщения о переменных, которые объявлены в проекте, но нигде не

используются. Флажок “Overlapping memory areas” позволит проверить третью версию неверной работы нашего выхода.

Версия 3. Перекрытие областей памяти

Допустим, мы более нигде явно не изменяем значение выхода. Тем не менее, оно изменяется. Это может быть вызвано ошибочным использованием прямоадресуемой памяти. В МЭК-программах к областям памяти входов и выходов можно обращаться напрямую. То есть в проекте могут иметь место запись по прямому адресу, повторное объявление выхода под другим именем либо может быть ошибочно указан тип “соседней” (в памяти) переменной, перекрывающий “чужую” память. Для контроля подобных фактов и предназначен флажок “Overlapping memory areas”. Для всех прочих переменных (локальных и глобальных) CoDeSys распределяет память автоматически, и подобные паразитные эффекты исключены.

Проверить, где и как используется в проекте интересующий нас прямой адрес, в CoDeSys можно с помощью вышеописанного инструмента контроля перекрестных ссылок. Например, по рис. 3 видно, что переменная *bOut_14* была объявлена по тому же адресу, что и *bOut_1*3.

В общем случае желательно использовать конфигуратор ПЛК для объявления всех прямоадресуемых переменных проекта и избегать вводить их вручную. Если вы умышленно хотите использовать в проекте прямые адреса, не прописанные изготовителем в конфигурации ПЛК (PLC Configuration), то включите в настройках вашей целевой платформы (Target settings) флажок “No address check”.

Конфигурационные переменные

Раз уж мы коснулись прямоадресуемых переменных, то рассмотрим попутно еще один полезный инструмент CoDeSys. Он позволяет свести к минимуму ручную работу и соответственно снизить вероятность ошибок. Допустим, мы имеем функциональный блок (назовем его BL), выполняющий некие действия с одним или несколькими входами/выходами ПЛК. Каждый экземпляр функционального блока (BL1, BL2 и т.д.) должен работать с разными адресами. В CoDeSys эту проблему лучше всего решить с помощью конфигурационных переменных. В разделе объявления переменных блока вместо полного прямого адреса мы зададим шаблон. Например:

```
bIn AT %I*: BOOL;
```

Далее на вкладке определения конфигурационных переменных менеджера проекта (“Resources”, “Variable Configuration”) определим полные адреса переменных для каждого экземпляра:

```
PLC_PRG.BL1.bIn AT %IX0.0 : BOOL;
PLC_PRG.BL2.bIn AT %IX0.1 : BOOL;
```

Определения не обязательно прописывать вручную. Дайте команду “Insert”, “All Instance Paths”. CoDeSys автоматически сформирует полный список объявлений. Вам останется только расставить прямые адреса.

Не забывайте о возможности использования конфигурационных переменных. Так можно объявлять все переменные с прямыми адресами. Компактное сосредоточение описаний прямоадресуемых переменных существенно упрощает сопровождение проекта.

Версия 4. Код не выполняется

Возможно, нужная нам команда вообще не выполняется, либо наш программный компонент (POU) по некоторым причинам завершает работу раньше, либо выполнение идет по другой ветви, либо не вызывается сам POU. Как это проверить? Проще всего поставить точку останова, запустить выполнение и подождать, выйдет ли программа в эту точку. В текстовом редакторе точка останова устанавливается (F9) на номер строки, в FBD и LD – на графический элемент и в SFC – на шаг. Элемент (номер строки) будет выделен голубым цветом. О достижении точки останова свидетельствует изменение цвета элемента на красный.

В точке останова мы можем просматривать значения переменных обычным образом. Часто полезно поставить точку останова раньше нужного места и “пошагать” по программе командами “Step over” (с перешагиванием вызываемых POU) или “Step in” (с заходом в вызываемые POU). В отличие от выполнения по циклам, данные команды обеспечивают выполнение по инструкциям. Дополнительно в точке останова можно просмотреть стек вызовов POU. Для этого служит команда “Online”, “Show Call Stack”. Это помогает понять, каким образом процесс выполнения пришел в данную точку.

Обзор взаимосвязей программных компонентов дает диаграмма вызовов компонентов проекта. Она вызывается командой “Project”, “Show Call Tree”.

Версия 5. Привнесенные ошибки

Нередко в объемных проектах ошибка может появиться вследствие недавно сделанных изменений. На первый взгляд изменения не касались соответствующих фрагментов. Однако загрузив из архива предшествующую версию нашего проекта, мы обнаруживаем, что она работает верно. Для анализа таких ситуаций в CoDeSys имеется очень

удобная функция сравнения проектов. Она вызывается командой “Project”, “Compare”. Текущий проект сравнивается с любым записанным на диске. Мы можем проводить сравнение без учета изменений форматирования, комментариев и свойств объектов (задается флажками). CoDeSys обладает достаточным “интеллектом” для понимания таких несущественных деталей. Результаты сравнения отображаются в разделенном окне. Красным цветом выделяются измененные компоненты, зеленым – новые и синим – удаленные. Например, на рис. 4 мы видим выделенную темным цветом (на экране красный) функцию *AxFunc6*. Раскрыв ее в режиме сравнения, обнаруживаем, что операция **XOR** была заменена на операцию **OR**, что подозрительно. С помощью специального набора команд, содержащегося в меню “Extras”, мы можем автоматически отменить определенные исправления.

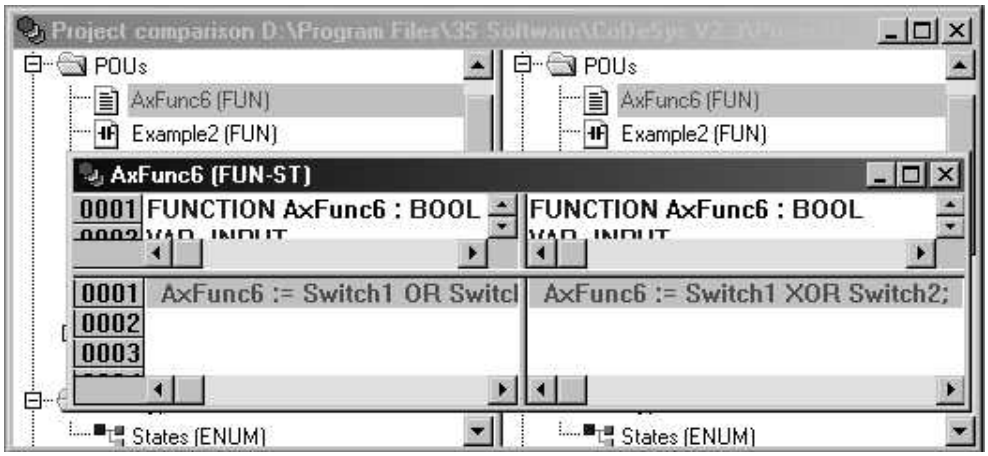

Рис. 4. Сравнение проектов.

Гораздо более широкие возможности дает применение инжинирингового сервера 3S (ENI). Это дополнительный опциональный компонент комплекса CoDeSys. ENI представляет собой многопользовательскую систему управления версиями. Он позволяет хранить компоненты проекта в базе данных в открытом XML-формате. Любое изменение программного компонента сохраняется в базе. Мы можем проводить откатку каждого компонента на любую дату, сравнивать версии компонентов и разделять их между несколькими проектами нашей компании. Кроме того, несколько человек могут одновременно удаленно работать над проектом. Если кто-либо открыл компонент для изменения, то это явно видно всем программистам проекта. Мы можем точно проконтролировать, что делалось с проектом, кем, когда и с какой целью.

Диагностика путем моделирования состояний

Выше мы рассмотрели способы поиска причины явного дефекта с устойчивой симптоматикой. К сожалению, часто дефект начинает проявляться только в процессе работы после выполнения ряда условий или определенной последовательности событий. Качественно проанализировать такую ситуацию сложно. Многократное практическое повторение ситуации может быть слишком сложным или недопустимым. Желательно иметь возможность “сфотографировать” мгновенные значения всех или специально выбранных значений переменных проекта. Считывать и устанавливать значения большого числа переменных вручную нереально. Для этого в CoDeSys служит специальный инструмент – “Менеджер рецептов”.

Откройте на вкладке ресурсов проекта окно “Watch and Receipt Manager”. В режиме off-line команда “Insert”, “New Watch List” позволяет создать новый поименованный список переменных. Используйте “Ассистент ввода” (F2) для определения списка. Мы можем выбрать переменные, которые должны войти в список. Здесь же в списке можно сразу вручную редактировать значения констант для переменных, которые мы хотим записывать в контроллер. Например: PLC_PRG.Point_1 := 15. Чтобы считать значения всех переменных в список, в режиме on-line используйте команду “Extras”, “Read Receipt”. Она заменит в списке все определения констант текущими значениями переменных. Команда “Extras”, “Write Receipt” выполняет обратное действие, то есть значения из списка синхронно записываются в память контроллера. Команда “Save Watch List” позволяет записать список на диск компьютера (файл с расширением wtc). Считывается список командой “Read Receipt”.

Создав несколько списков, мы можем менять отдельные значения переменных и достаточно просто моделировать различные ситуации. Это позволяет детально тестировать сложные в диагностике ситуации.

Версия 6. Динамические ошибки

Иногда мы можем обнаружить удивительную ситуацию. При выполнении программы по циклам значение выходной переменной устанавливается верно. Но стоит только запустить программу в реальном времени, выход моментально сбрасывается. Вполне возможно, он включается в нужном месте и в нужное время, но далее сбрасывается. Это может происходить настолько быстро, что мы не успеваем “поймать” значение выхода ни при мониторинге,

ни наблюдая светодиодный индикатор состояния выхода ПЛК. Исследовать подобные проблемы помогает графическая трассировка CoDeSys.

Трассировка

Данный инструмент подобен трендам SCADA-систем, но имеет существенное отличие. Данные трассировки CoDeSys аккумулируются в памяти контроллера. Это делается системой исполнения в реальном времени. Для отображения на компьютере данные передаются по его запросу асинхронно от записи. Это означает, что мы можем проводить детальнейшее отслеживание значений переменных по рабочим циклам, даже если контроллер очень быстрый, а канал связи медленный. То есть трассировка CoDeSys приемлема не только для анализа медленных технологических процессов, но и для анализа хода выполнения программы. Мы не будем описывать все опции настройки трассировки. Они достаточно подробно изложены в руководстве по программированию. Рассмотрим только несколько идей, полезных для наших текущих целей.

Итак, перейдите на вкладку “Resources” менеджера проектов и откройте окно “Sampling Trace”. Дайте команду “Extras”, “Trace Configuration”. Окно настройки трассировки требует некоторых пояснений (рис. 5). Минимум, что нужно сделать, – это задать переменные, значения которых нас интересуют. Мы можем ввести имя переменной в поле “Input of trace variable” вручную либо выбрать из списка (кнопка “Help Manager”). Нажав кнопку “Insert”, мы добавим переменную в список отслеживаемых переменных “Variables”. Параллельно можно контролировать до 20 переменных. Точнее, под работу трассировки в памяти системы исполнения отведен кольцевой буфер фиксированного размера (разный в разных моделях ПЛК). Это означает, что чем больше переменных мы добавим, тем меньшее число замеров (Number of samples) сможет аккумулировать буфер. Допустимое число замеров, количество и тип переменных взаимосвязаны.

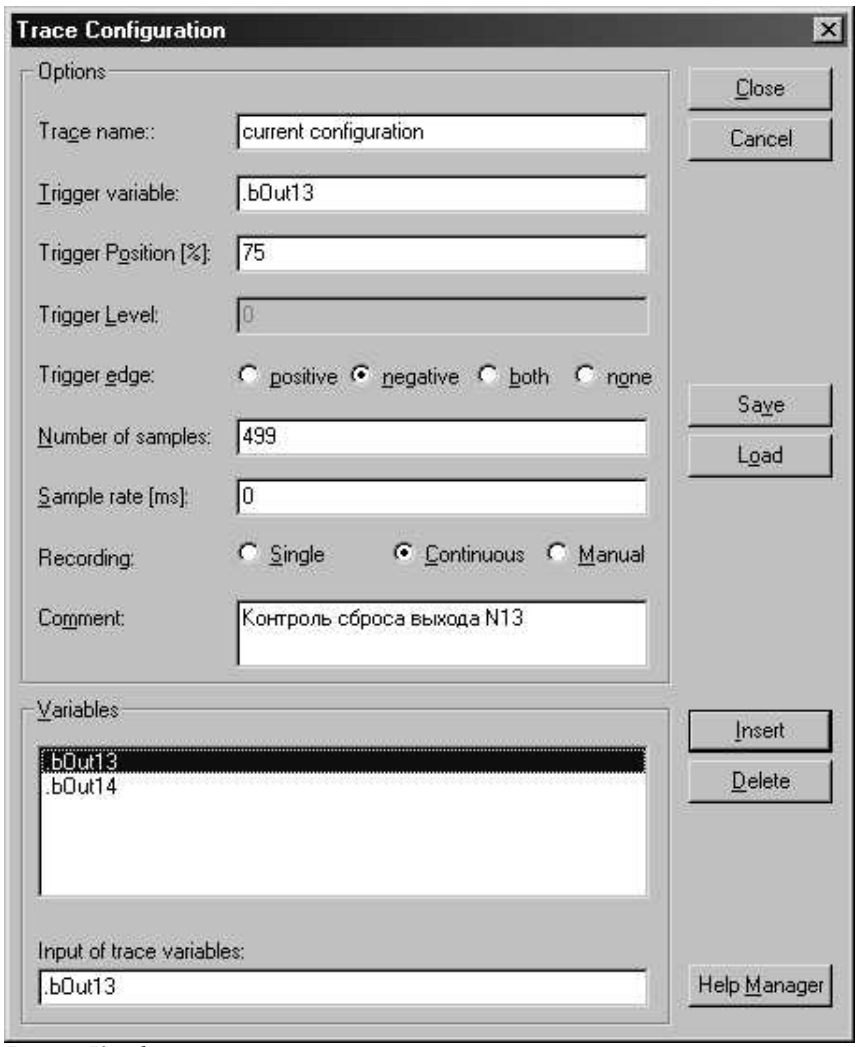

Рис. 5. Конфигурация трассировки.

Поле “Sample Rate” используется для установки периода между записями значений переменных в миллисекундах. Значение по умолчанию 0 означает синхронную запись значений в каждом рабочем цикле.

По умолчанию, трассировка работает непрерывно в режиме самописца до команды остановки. Но как уже было сказано, с ее помощью можно фиксировать определенные события. Таким событием может служить изменение любой переменной, именуемой триггерной. Ее необходимо указать в поле “Trigger Variable”. Критический порог значения триггерной переменной задается “Trigger Level”. Флажки “positive”, “negative”, “both” задают останов трассировки соответственно при росте, уменьшении или равенстве значения триггерной переменной уровню порога. Чтобы мы смогли проанализировать ситуацию не только до события, но и непосредственно после него, есть смысл останавливать трассировку с некоторой задержкой. Величину этой задержки в процентах от общего числа замеров мы можем задать в поле “Trigger position”. Поля “Trace Name” (название конфигурации трассировки) и “Comment” (комментарий) пояснений не требуют.

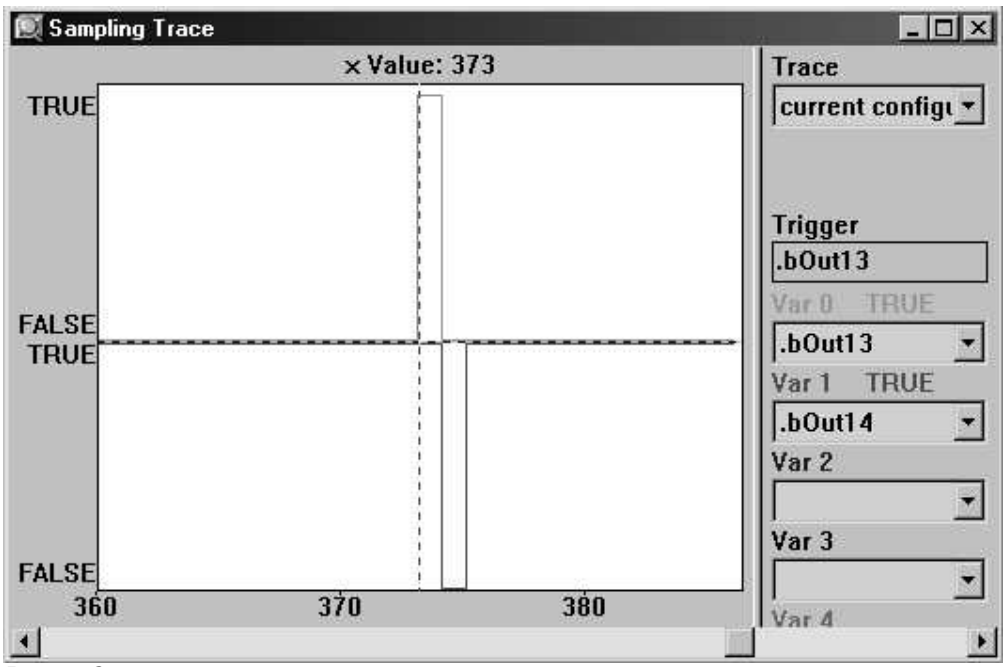

Рис. 6. Окно трассировки.

После настройки конфигурации нужно выбрать цвета для отражения переменных в диалоговом окне трассировки. Это можно делать и “на ходу”. Далее запускаем трассировку командой “Start Trace”, ждем заданного события (сообщения в статусной строке) и считываем результат командой “Read Trace”. В нашем примере, растянув изображение (Stretch), можно заметить (рис. 6), что выход *bOut_13* сбрасывается в следующем рабочем цикле после включения, причем синхронно с *bOut_14*.

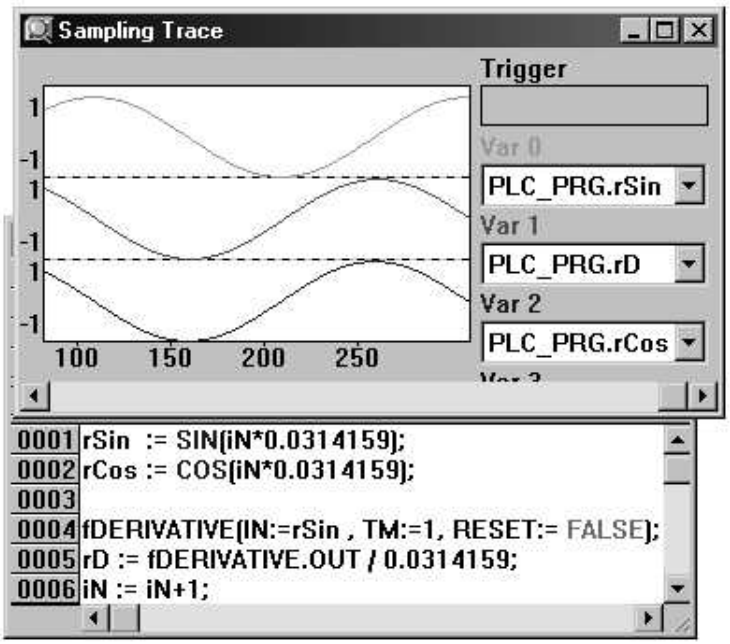

Рис. 7. Тестирование блока дифференцирования.

Трассировка – это исключительно удобный и востребованный инструмент в CoDeSys. Он применяется на этапе проектирования для изучения объектов и исследования функциональных зависимостей (рис. 7), при эксплуатации для технологических целей и исследования проблемных ситуаций. Обратите внимание, что после запуска трассировки связь с компьютером иметь необязательно. Трассировка может годами работать параллельно с программой, не влияя на нее. После “срабатывания” триггера данные хранятся в памяти контроллера до тех пор, пока не будут считаны. Так мы можем снабдить наш проект своего рода гибко конфигурируемым “черным ящиком”. Без трассировки нам пришлось бы организовывать в программе запись в log-файл или в очередь событий, что в ПЛК не всегда возможно.

Обратите внимание на возможность сохранения на диске не только наборов конфигураций трассировки, но и данных. Причем команда “Extras”, “Save Values”, “Trace in ASCII-File” сохраняет данные в текстовом формате, пригодном для экспорта в электронные таблицы и базы данных.

Заключение

Что должна делать правильная офисная программа, если столкнется с невозможностью выполнить предписанные действия? Очевидно, пора выдать соответствующее сообщение на экран пользователю. Но что делать в подобных случаях в ПЛК? Хотелось бы зависнуть, но не положено. Экрана нет, оператор далеко. “Однако, будучи в руках заказчиков, программное обеспечение должно функционировать в ситуациях, которые никогда не проверялись и не ожидались” [2].

Продолжение следует.

Список литературы

1. *Тэллес М., Хсих Ю.* Наука отладки / Пер. с англ. М.: КУДИЦ-ОБРАЗ, 2003.
2. *Ослэндер Д.М., Риджли Дж.Р., Рингенберг Дж.Д.* Управляющие программы для механических систем: объектно-ориентированное проектирование систем реального времени / Пер. с англ. М.: БИНОМ. Лаборатория знаний, 2004.