Fastwel I/O изнутри

Александр Локотков

В статье рассматриваются внутреннее устройство и принципы функционирования основных составных частей аппаратно-программного комплекса Fastwel I/O, предназначенного для создания автоматизированных систем сбора данных и управления. Представлены подходы к проектированию и детально описаны межмодульная внутренняя шина FBUS, адаптированная среда исполнения прикладных программ CoDeSys, сервисы сетевых протоколов и особенности взаимодействия составных частей комплекса друг с другом.

Часть 4

Исполнение прикладной программы

Прикладная программа функционирует в контроллере под управлением среды исполнения CoDeSys следующим образом:

1. Сервис ввода-вывода выводит из области выходных данных данные, подготовленные на предыдущем цикле исполнения, в выходные каналы модулей ввода-вывода.
2. Данные входных каналов модулей ввода-вывода и входящих коммуникационных вводятся в область входных данных среды исполнения CoDeSys.
3. Вызывается на исполнение программа PLC_PRG.
4. Данные из области выходных данных среды исполнения CoDeSys подготавливаются к выводу во внешнюю сеть и в модули ввода-вывода.

Описанный цикл контроллера повторяется с периодом, который задаётся пользователем путём установки параметра CPM70x Coupler – SampleRate в диапазоне от 1 до 1000 мс в секции PLC Configuration.

Необходимо обратить внимание на следующие моменты:

1. Перед вызовом программы PLC_PRG происходит непосредственный вывод данных в выходные каналы модулей ввода-вывода, причём выводятся данные, буферизованные сервисом ввода-вывода в конце предыдущего цикла.
2. Вывод данных в выходные каналы модулей ввода-вывода и ввод данных входных каналов производятся за один обмен (за одну транзакцию) по внутренней шине контроллера.
3. После очередного исполнения программы данные из области выходных данных среды исполнения CoDeSys выводятся не в модули, а буферизуются сервисом ввода-вывода.
4. Запросы чтения и записи по внешней сети поступают в случайные моменты времени относительно цикла выполнения программы. При поступлении запроса чтения по внешней сети в сеть передаются буферизованные данные из области выходных данных среды исполнения CoDeSys, относящиеся к последнему завершившемуся циклу прикладной программы. При поступлении запроса записи данные, пришедшие в запросе по сети, передаются в область входных данных среды исполнения CoDeSys непосредственно перед следующим после запроса записи вызовом прикладной программы.

Для того чтобы сказанное стало более понятным, соберем небольшой стенд и оценим время реакции среды исполнения на изменение сигнала, поступающего на вход модуля дискретного ввода. В стенд войдут контроллер CPM703 (Modbus TCP) и следующие модули:

1) модули OM751 и DIM718. Первый служит для питания выходных цепей следующего, который содержит 8 каналов дискретного вывода с коммутируемым напряжением от 0 до 30 В. Третий канал модуля будет использоваться для генерации меандра в режиме формирования ШИМ-последовательности. Полученный меандр будет использоваться в качестве испытательного воздействия и подаваться на входы модулей аналогового и дискретного ввода. Седьмой канал модуля будет переключаться из прикладной программы, как только она обнаружит перепад сигнала на входе другого модуля, на который подан испытательный сигнал в виде меандра;
2) два модуля DIM716, каждый из которых имеет два канала дискретного ввода с диапазоном от 0 до 30 В. На первый канал второго модуля будет подаваться меандр, формируемый третьим каналом модуля DIM718;
3) четыре 2-канальных модуля аналогового ввода AIM726 с диапазоном входного напряжения от 0 до 40 В и четыре 4-канальных модуля аналогового ввода AIM728 с диапазоном входного напряжения от –20 до +20 В. Эти модули будут выступать в качестве «массовки», ничего не измеряя, но увеличивая время цикла межмодульной внутренней шины.

Определим потребное время опроса перечисленных модулей. Размеры областей входных и выходных данных выбранных модулей таковы:

- OM751: 2 байта,
- DIM718: 10 + 9 байт,
- DIM716: 7 байт,
- AIM726: 9 байт,
- AIM728: 17 байт.

Тогда потребное время опроса составит:

$$((1\times2+1\times19+2\times7+4\times9+4\times17)/168960)\times1000=0{,}82\text{ мс}$$

Тогда потребный период опроса, а значит, и период исполнения прикладной программы, составит: 0,82/0,70 = 1,18 ≈ 2 мс (округляем до «верхнего» целого, потому что оценка потребного времени менее 2 мс).

Создадим проект в среде CoDeSys для платформы Fastwel I/O System и добавим в конфигурацию контроллера CPM703 описания модулей с соблюдением указанного выше порядка: OM751, DIM718, DIM716, DIM716, AIM726, AIM726, AIM726, AIM726, AIM728, AIM728, AIM728, AIM728.

Установим значение параметра CPM703 ModbusTCP Coupler–SampleRate равным 2 мс.

В конфигурации модуля DIM718 для параметра Mode выберем значение Use PWM, в результате чего после загрузки конфигурации в контроллер и перезапуска 3-й и 4-й каналы модуля будут переведены в режим формирования ШИМ-последовательностей с характеристиками, определяемыми длительностями полуволн, которые задаются прикладной программой на его логических выходах firstHalfDutyState0: secondHalfDutyState0 и firstHalfDutyState1: secondHalfDutyState1.

Программа для оценки времени реакции среды исполнения на изменение сигнала на входном канале модуля дискретного ввода будет выглядеть так:

```
PROGRAM PLC_PRG
VAR
(* Длительность полуволн ШИМ-последовательности
   на третьем канале DIM718 в единицах, кратных 50 мкс *)
     pwmTime : WORD;
(* Признак обнаружения перепада на входе второго модуля DIM716 *)
     toggleSensed : BOOL := FALSE;
END_VAR
VAR_INPUT
(* Первый бит содержит логическое состояние на первом
   дискретном входе модуля DIM716 *)
     din_byte AT%IB57 : BYTE;
END_VAR
VAR_OUTPUT
(* Ссылка на 7-й, начиная с 1, выходной канал DIM718 *)
     dout1 AT%QX2.14 : BOOL := FALSE;
(* Ссылка на логический канал формирования первой полуволны
   на 3-м, начиная с 1, выходном канале DIM718 *)
     dout0_pwm_first AT %QW3 : WORD;
(* Ссылка на логический канал формирования второй полуволны
   на 3-м, начиная с 1, выходном канале DIM718 *)
     dout0_pwm_second AT %QW4 : WORD;
END_VAR
(* Программа начинается здесь *)

(* Задаваемая длительность полуволны испытательного сигнала:
  T = 200×0,05 = 10 мс *)
  pwmTime := 200;
(* Записываем значения длительности первой и второй полуволн *)
  dout0_pwm_first := pwmTime;
  dout0_pwm_second := pwmTime;
(* Определяем факт перепада и текущее логическое состояние
   на 1-м входе второго модуля DIM716 *)
  toggleSensed := ((din_byte AND 1) <> 0);
(* Выводим текущее логическое состояние в 7-й, начиная с 1,
   выходной канал модуля DIM718 *)
  dout1 := toggleSensed;
END_PROGRAM
```

Соберём описанную конфигурацию контроллера. Подключим выход блока питания с выходным напряжением 24 В к клеммам питания контроллера, а также к контактам с маркировкой «1» (+24 В) и «2» (Общий) модуля OM751. Далее соединим контакт с маркировкой «2» модуля DIM718 с контактом «1» второго модуля DIM716 и первым каналом осциллоскопа. Второй канал осциллоскопа подключим к контакту с маркировкой «4» модуля DIM718. Включим блок питания и загрузим предварительно скомпилированную программу в контроллер, после чего выполним команду **Online-Logout** в среде разработки CoDeSys, чтобы взаимодействие между средой разработки и контроллером не влияло на измерение.

На индикаторе осциллоскопа появится осциллограмма, аналогичная представленной на рис. 18. При этом запаздывание фронтов сигнала, формируемого программой в ответ на перепады испытательного сигнала, относительно фронтов испытательного сигнала будет периодически изменяться от 4 с небольшим до 5,5 мс.

Проанализируем полученный результат. Для этого зафиксируем осциллоскопом циклограммы исполнения прикладной программы и обмена с модулями по внутренней шине. Способ подключения щупов осциллоскопа к контроллеру иллюстрирует рис. 19. Общий провод одного из щупов может быть подключён ко второму или третьему сверху контакту под пластмассовой крышкой на передней панели контроллера. Самый верхний контакт под крышкой позволяет увидеть импульсы, передние фронты которых соответствуют началу ввода данных в программу, а задние – окончание вывода. Высокий уровень между фронтами удерживается в течение времени исполнения программы. Совмещённые циклограммы с дополнительными пояснениями представлены на рис. 20.

На рис. 20 рассмотрена наихудшая ситуация, когда изменение входного испытательного сигнала происходит в момент окончания получения модулями группового запроса. Как только модуль получил корректный пакет группового запроса, он меняет местами буфер, в котором находятся ранее зафиксированные входные данные, с коммуникационным буфером и приступает к формированию своей части цепочечного ответа. Если перепад входного сигнала произошёл в момент обслуживания модулем очередного группового запроса, то фактически данный перепад попадёт в коммуникационный буфер и будет передан мастеру FBUS только при обслуживании следующего группового запроса, что и обозначено на рисунке. А это означает, что программа «увидит» перепад испытательного сигнала во время выполнения второго цикла с момента его реального происхождения. Далее, перед началом следующего цикла программы произойдет очередной обмен с модулями ввода-вывода, и выходной сигнал, сформированный программой в ответ на обнаружение перепада испытательного сигнала, поступит модулю дискретного вывода, который после небольшой задержки актуализирует его на соответствующем дискретного выходе.

Таким образом, параметр «Время реакции прикладной программы на изменение состояния на канале дискретного ввода», определённый в документации, значение которого в предельном случае равно двум периодам исполнения программы, представляет максимальный интервал времени, по истечении которого программа обнаружит фронт сигнала на дискретном входе модуля. Если под временем реакции системы подразумевать интервал времени, выделенный на рис. 20 пунктирными линиями, то его предельное значение составляет два периода исполнения программы плюс время, которое понадобится потратить на один обмен со всеми модулями, подключёнными к внутренней шине контроллера, плюс время задержки выдачи, специфичное для каждого модуля.

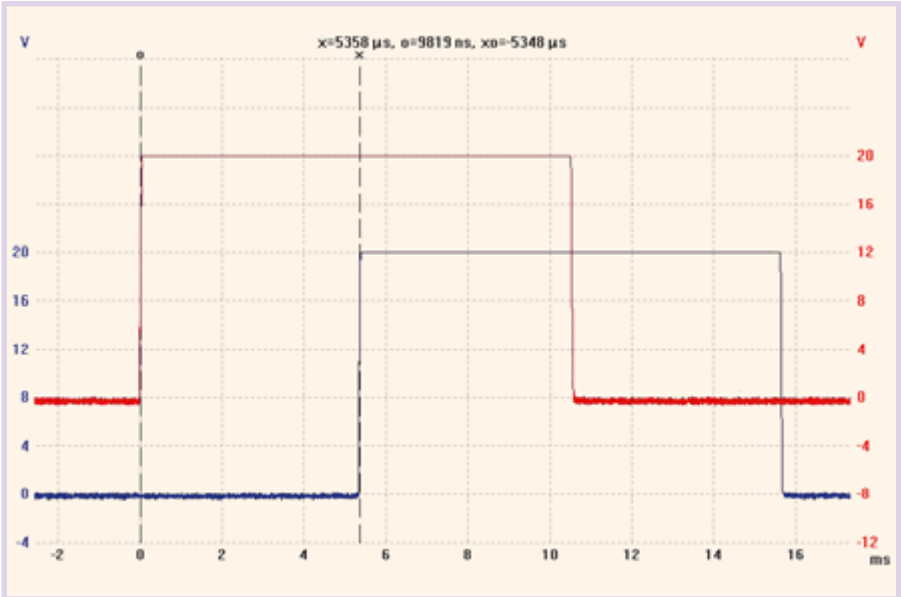

Рис. 18. Оценка времени реакции на перепад сигнала на дискретном входе. Трасса красного цвета – испытательное воздействие. Трасса синего цвета – подтверждение восприятия программой испытательного воздействия. Запаздывание составляет 5,35 мс

Рис. 19. Подключение осциллоскопа к контроллеру для получения циклограмм исполнения прикладной программы и обмена с модулями по внутренней шине

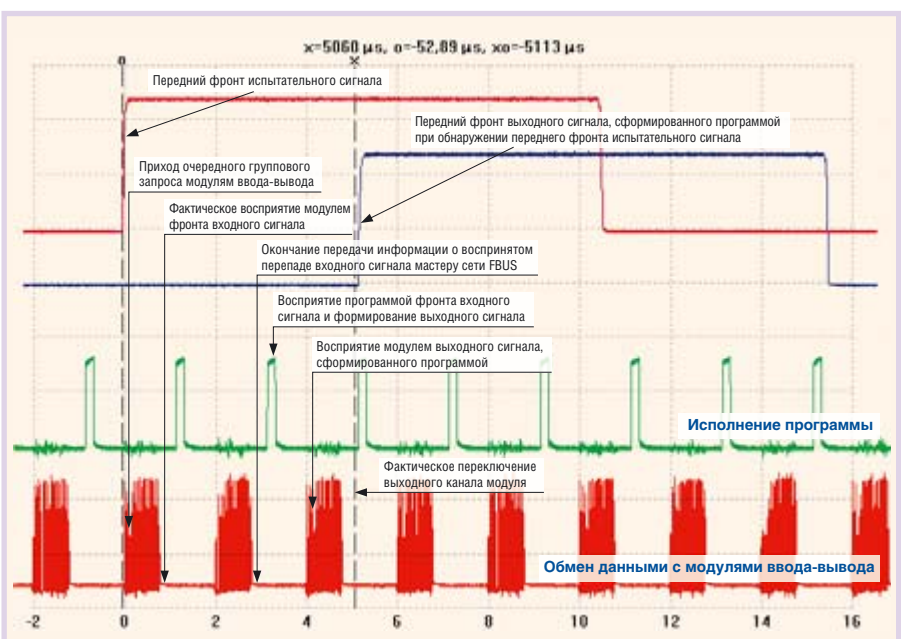

Рис. 20. Циклограммы для программы оценки времени реакции на изменение сигнала на входном канале модуля дискретного ввода

Режимы работы контроллера

Безопасный режим

При поставке контроллер не содержит прикладной программы и при включении питания запускается в так называемом безопасном режиме, о чем свидетельствует попеременное свечение индикатора RUN/ERR зеленым и красным цветами. В безопасном режиме системное программное обеспечение контроллера не обращается к модулям ввода-вывода и ожидает загрузки новой прикладной программы.

В случае когда неизвестны параметры обмена контроллера по внешней сети и требуется загрузить новую программу, контроллер может быть принудительно переведён в безопасный режим. Для этого следует включить переключатель «1» и перезапустить контроллер.

Контроллер может перейти в безопасный режим самостоятельно, если при запуске или при выполнении прикладной программы произошла одна из следующих ошибок:

1) размер бинарного кода программы или конфигурации при загрузке превысил максимальную величину (64000 байт – для программы, 64512 байт – для конфигурации);
2) в конфигурации внешней сети контроллера имеются описания по крайней мере двух коммуникационных объектов (например, регистров Modbus) с одинаковыми идентификаторами;
3) в прикладной программе произведено деление на 0;
4) в коде прикладной программы, загруженной в контроллер, произошла ошибка, повлекшая за собой немаскируемое прерывание микропроцессора или прерывание по неизвестному коду команды;
5) если в контроллер некоторого типа по ошибке загружена программа для контроллера другого типа.

Если контроллер запустился в безопасном режиме по ошибке, его коммуникационные параметры (например IP-адрес, адрес шлюза и маска подсети у контроллера CPM703) примут значения из последней удачно загруженной конфигурации.

Зачем мы сделали безопасный режим? Безопасный режим появился прежде всего потому, что программа, разработанная пользователем и загруженная в контроллер, может аварийно завершиться немедленно или по истечении некоторого времени. Возможные причины аварийного завершения:

1. Деление на ноль. Компилятор CoDeSys генерирует исполняемый код процессора целевой платформы, поэтому если в программе произошло деление на ноль, она будет аварийно завершена.

2. Пользователь добавит в проект на CoDeSys библиотеку, функции которой не поддерживаются адаптированной средой исполнения для Fastwel I/O, и вызовет какую-нибудь функцию у себя в про-

грамме. Либо сигнатура какой-нибудь библиотечной функции отличается от её текущей реализации в среде исполнения типами или количеством аргументов и/или возвращаемого значения.

3. Обращение по нулевому указателю или по указателю, ссылающемуся не туда, куда нужно, из программы пользователя.
4. Компилятор среды разработки CoDeSys сгенерирует неправильный код.

Рассмотрим перечисленные причины подробнее.

Деление на ноль может случиться сразу после загрузки программы или через некоторое время, и при этом не существует никакого корректного способа продолжить работу контроллера, поскольку компилятор CoDeSys генерирует исполняемый код целевого процессора.

О вызове библиотечных функций. Реализация любой функции внешней библиотеки, используемой в проектах CoDeSys для платформы на базе 80186, должна присутствовать в коде среды исполнения. Перед запуском прикладной программы среда исполнения выполняет динамическое связывание внешних функций, чьи имена перечислены в специальной секции бинарного файла загруженной программы. Если какая-либо функция отсутствует, то контроллеру остаётся только перезагрузиться либо «зависнуть» в момент вызова отсутствующей функции.

Кроме того, связывание выполняется только по именам функций без анализа списков аргументов и возвращаемого значения (информацию о сигнатурах компилятор в бинарный код не помещает), поэтому если вызов внешней функции, сгенерированный компилятором CoDeSys, отличается по типам и/или количеству аргументов и/или типу возвращаемого значения от текущей реализации в среде исполнения, произойдёт порча стека и крах системы.

Об указателях. CoDeSys имеет реализацию специального типа данных POINTER TO, означающего «указатель на». Несомненно, указатели являются мощным инструментом в умелых руках. Но также несомненно, что поддержка указателей отрицательно влияет на безопасность кода, ибо запись по неправильному и нулевому указателю с большой вероятностью приводит к краху систему.

Спрашивается, что делать, если произошла какая-нибудь из перечисленных неприятностей? Если просто перезагрузить контроллер, то крах произойдет опять и будет повторяться до тех пор, пока пользователь каким-то способом не прекратит мучения контроллера. А что если контроллер находится в труднодоступном месте? Кроме того, до следующей перезагрузки программа может успеть что-нибудь сделать, например, несколько раз переключить дискретные выходы, что не всегда желательно.

В общем, безопасный режим показался нам разумным решением, позволяющим пользователю увидеть, что что-то идет не так, и исправить положение, загрузив правильную программу. Мы установили собственные обработчики для немаскируемого прерывания, прерывания по неизвестному коду команды и для прерывания по делению на ноль. При возникновении любого из перечисленных прерываний мы стараемся сохранить в энергонезависимой памяти контроллера причину краха системы и принудительно перезагружаем контроллер аппаратным сбросом при помощи сторожевого таймера. Кроме того, во время работы системы регулярно проверяется целостность памяти (точнее, так называемой «кучи» системы исполнения языка Си, на котором написано системное ПО контроллера) с целью выяснить, не было ли деструктивных обращений по неправильному указателю из пользовательской программы. После перезапуска контроллера система считывает причину последней перезагрузки и выясняет, что произошла перезагрузка по ошибке, и контроллер начинает работать в безопасном режиме.

Нормальный режим

Если в контроллер была успешно загружена прикладная программа, конфигурация которой соответствует типу контроллера и не содержит описаний коммуникационных объектов (регистров Modbus или коммуникационных объектов CANopen) с одинаковыми адресами (идентификаторами), контроллер при включении питания или перезапуске будет запускаться в нормальном режиме.

В нормальном режиме исполняются основные функции контроллера, включая периодическое исполнение прикладной программы, обмен данными с модулями ввода-вывода, обслуживание запросов, поступающих по внешней сети, и т.д.

При этом

1) индикатор RUN/ERR светится зелёным цветом, если среда исполнения CoDeSys успевает выполнять прикладную программу с заданным периодом. В противном случае RUN/ERR светится красным цветом;
2) индикатор APP светится зелёным цветом;
3) индикатор I/O светится зелёным цветом при успешном обмене с модулями ввода-вывода. Если текущая конфигурация модулей ввода-вывода, подключённых к контроллеру, не совпадает с конфигурацией модулей ввода-вывода, заданной для прикладной программы, либо в случае наличия большого количества ошибок обмена по внутренней шине контроллера индикатор I/O светится красным цветом.

Как указывалось ранее, программа, разработанная в среде CoDeSys, транслируется в исполняемый код микропроцессора 80186. При выполнении команды **Online–Download** в среде разработки CoDeSys сначала загружается бинарный код программы, который сохраняется в отдельном файле в энергонезависимой памяти контроллера, после чего выполняется загрузка конфигурации в бинарном формате с сохранением в отдельном файле. Подробности о взаимодействии контроллера со средой разработки приведены далее.

При запуске контроллера происходит загрузка конфигурации из энергонезависимой памяти в оперативную и инициализация сервисов адаптированной среды исполнения CoDeSys.

Если предполагается разрабатывать большие программы для контроллеров с большим количеством модулей ввода-вывода и коммуникационных объектов внешней сети, необходимо до загрузки программы в контроллер определить, поместится ли в контроллер конфигурация, созданная пользователем в секции PLC Configuration. Как определить размер некоторой конфигурации программы? В документе «Система ввода-вывода Fastwel I/O. Модули ввода-вывода. Руководство программиста» приведены размеры элементов конфигурационной информации, которые могут быть добавлены в конфигурацию контроллера. Например, если требуется иметь 64 модуля DIM714 (394 байта на 1 модуль) в составе контроллера CPM702 (861 байт), то размер конфигурации составит $64 \times 394 + 861 = 26077$ байт. Тогда в этой конфигурации может быть до 463 Modbus-регистров (83 байта на 1 регистр).

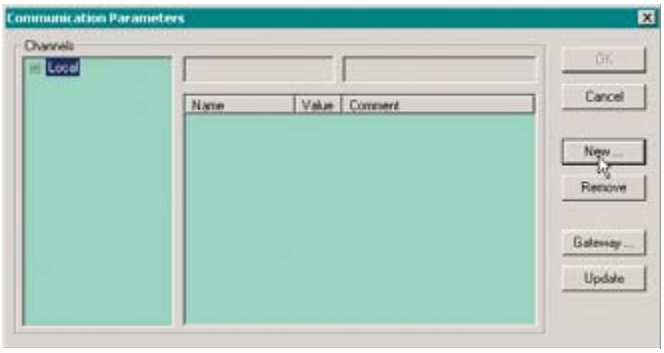

Рис. 21. Диалоговая панель параметров коммуникационного сервера связи с удаленным контроллером

Использование библиотек

Адаптированная среда исполнения CoDeSys для Fastwel I/O содержит реализации функций следующих внешних системных библиотек:

1) Standard.lib;
2) SysLibFile.lib. Содержит реализации функций для работы с файлами;
3) SysLibCom.lib. Здесь реализованы все функции для работы с последовательным портом, расположенным под пластиковой крышкой на передней панели контроллера. Для получения доступа к данному порту из прикладной программы через функции библиотеки SysLibCom.lib необходимо включить переключатель «4» на передней панели контроллера.

Кроме того, при установке комплекта адаптации CoDeSys для Fastwel I/O в каталог \Support корневого каталога установки помещаются библиотеки, содержащие функциональные блоки для преобразования данных от модулей аналогового ввода Fastwel I/O.

Все необходимые для повторного использования дополнительные функции и блоки рекомендуется реализовывать на Structured Text и оформлять в виде внутренних библиотек (тип проекта CoDeSys – Internal Library), поскольку реализация Structured Text в CoDeSys вполне приемлема для разработки довольно сложных программ.

Взаимодействие между средой разработки CoDeSys и средой исполнения контроллера

Адаптированная среда исполнения CoDeSys для Fastwel I/O поддерживает следующие виды взаимодействия со средой разработки:

1) загрузку прикладной программы и конфигурации в контроллер;
2) чтение и запись значений переменных программы, выполняющейся в контроллере;
3) пошаговую отладку программы в контроллере;
4) загрузку файлов в контроллер;
5) выгрузку файлов из контроллера.

Любое из перечисленных взаимодействий выполняется либо через интерфейс внешней сети контроллера, либо через специальный интерфейс «точка-точка» по последовательному каналу связи через порт COM1, соединитель которого расположен под пластиковой крышкой на передней панели контроллера. Типовой сценарий загрузки программы в контроллер выглядит следующим образом:

1. Пользователь создаёт проект в среде CoDeSys для платформы Fastwel I/O System, разрабатывает программу и компилирует командой **Project-Rebuild all**.
2. Выбирается команда меню **Online–Communication Parameters...** На экране появляется диалоговая панель **Communication Parameters**, показанная на рис. 21.
3. Создаётся логический информационный канал удалённого взаимодействия с контроллером, для чего нажимается кнопка **New** и в появившейся диалоговой панели вводится имя создаваемого канала, а в списке **Device** выбирается тип драйвера протокола (например, *ModbusTCP: Fastwel Modbus TCP driver,* как показано на рис. 18), и диалог **Communication Parameters: New Channel** закрывается нажатием кнопки **OK**. В древовидном списке **Channels** диалога **Communication Parameters** появится элемент, соответствующий созданному каналу, а в таблице параметров канала справа – параметры созданного канала, как показано на рис. 22.
4. Выполняется настройка параметров канала (рис. 23). В данном примере по двойному щелчку мыши на значении параметра *Address* становится доступным для редактирования значение IP-адреса контроллера. По окончании настройки диалоговая панель **Communication Parameters** закрывается нажатием кнопки **OK**. В настоящий момент среда CoDeSys подготовлена к загрузке прикладной программы в контроллер и выполнению других операций с контроллером по сети.

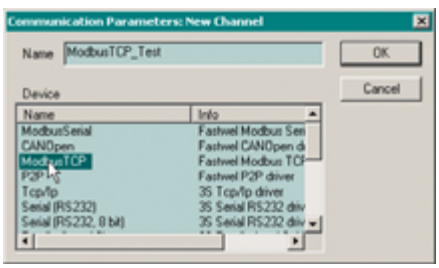

Рис. 22. Создание канала с использованием драйвера Fastwel Modbus TCP driver

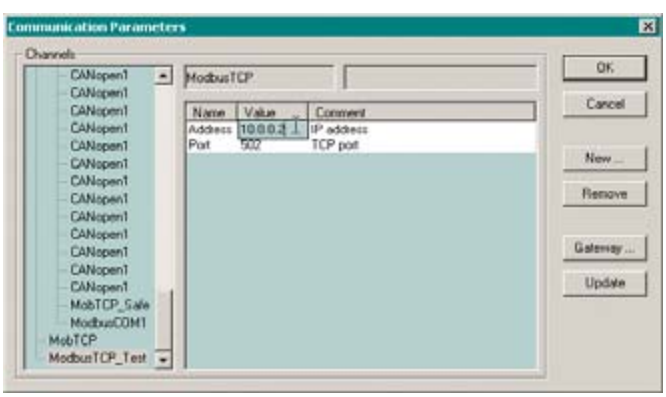

Рис. 23. Настройка параметров канала

5. В среде разработки выбирается команда **Online–Login**. При успешном соединении на экран будет выведена диалоговая панель, показанная на рис. 24.
6. После нажатия кнопки **Yes** на экран будет выведено окно, отображающее ход загрузки программы, показанное на рис. 25. Следует обратить внимание на тот факт, что в данном окне отображается ход загрузки только исполняемого кода программы. Когда исполняемый код программы загружен, начинается загрузка конфигурации контроллера,

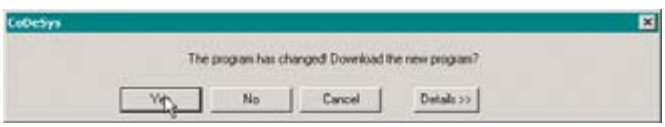

Рис. 24. Предложение загрузить программу

однако счётчик в окне не изменяется, будто бы загрузка уже завершена. Указанная ситуация особенно заметна в больших проектах, когда конфигурация контроллера содержит большое количество модулей ввода-вывода и/или регистров.

По завершении загрузки программы и конфигурации контроллер автоматически перезапускается и пытается запустить загруженную программу, если не включен переключатель «1» (выключите его заранее до загрузки программы!). Если контроллер во время загрузки программы был в безопасном режиме по отсутствию программы или по переключателю «1», после перезапуска контроллера произойдет разрыв связи между средой CoDeSys и контроллером, поскольку параметры интерфейса внешней сети узла в конфигурации контроллера, созданной в загруженной прикладной программе, скорее всего, отличаются от параметров внешней сети для безопасного режима (они должны отличаться!).

Если загрузка программы выполнялась в нормальном режиме контроллера, после перезапуска контроллера по завершении загрузки среда CoDeSys с очень высокой вероятностью сохранит соединение с контроллером и начнёт отображать значения переменных текущей прикладной программы.

При загрузке новой программы в контроллер исполнение текущей программы и обмен данными с модулями ввода-вывода прекращаются. Дело в том, что однозадачная среда исполнения CoDeSys для контроллеров на базе процессоров, совместимых с 80186, обслуживает сетевые запросы, поступающие от среды разработки, при каждом вызове прикладной программы в том же потоке исполнения, на контексте которого вызывается прикладная программа. Если при этом осуществляется интенсивный обмен по внутренней шине с большим количеством модулей ввода-вывода и выполняется большой объём вычислений в прикладной программе, то несколько пакетов от среды разработки могут быть потеряны, в результате чего среда разработки может решить, что пора разрывать связь с контроллером.

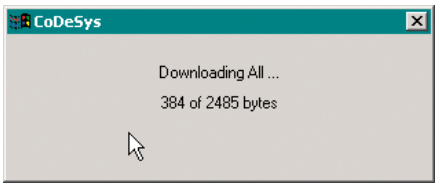

Рис. 25. Отображение хода загрузки программы

Рассмотрим процесс загрузки программы в контроллер более подробно.

Любая операция взаимодействия между средой разработки и средой исполнения CoDeSys в контроллере представляется в виде обращения клиента (среды разработки) к некоторой удалённой процедуре на сервере (контроллере). Каждый сетевой запрос клиента содержит код удаленной процедуры, собственно данные запроса и номер фрагмента транспортируемых данных запроса. Ответ сервера выглядит практически так же. Мы передаём запросы клиента и ответы сервера по сети, упаковывая их в пакеты стандартных сервисов сетевого протокола, реализуемого контроллером. Например, при взаимодействии через Modbus (и Modbus TCP) используется стандартный сервис инкапсулированного транспорта (2Bh), а при взаимодействии через CANopen – чтение/запись мультиплексоров объектного словаря контроллера через нулевой серверный SDO (передача запросов – через 5F25h:0h, приём ответов на запросы – через 5F26h:0h). При взаимо-

действии «точка-точка» (P2P) никаких дополнительных протоколов не используется, поэтому при таком способе взаимодействия достигается наивысшая производительность обмена. В контроллере взаимодействие сервиса внешней сети (любой из поддерживаемых) с коммуникационным драйвером среды исполнения CoDeSys осуществляет специальный сервис вызова удаленных процедур (RPC), который получает управление каждый раз, когда по сети приходит пакет запроса от среды разработки CoDeSys, либо когда среда исполнения CoDeSys в контроллере отвечает на запрос среды разработки. При этом сервис RPC в качестве источника и назначения данных, передаваемых между средой разработки и средой исполнения CoDeSys, использует коммуникационный буфер небольшого размера, который, в свою очередь, используется коммуникационным драйвером CoDeSys для приёма запросов и передачи ответов на запросы. Таким образом, взаимодействие можно представить следующим несколько упрощённым сценарием.

1. Среда разработки CoDeSys инициирует очередную операцию взаимодействия с контроллером, передавая код операции, которую он бы хотел выполнить на сервере (в контроллере), и параметры (данные) операции своему коммуникационному сервису Gateway Server, установленному на том же компьютере, что и среда разработки.
2. Gateway Server передаёт код и параметры операции в текущий выбранный коммуникационный канал, обслуживание которого осуществляет наш драйвер протокола, выбранный пользователем при создании канала.
3. Драйвер протокола начинает транзакцию коммуникационного протокола (например, запись потока байтов через SDO в CANopen или через сервис инкапсулированного транспорта Modbus), поместив в поле данных первого исходящего пакета транзакции код типа взаимодействия (этот код означает, что в данном сетевом пакете передаются данные, относящиеся к обмену между IDE CoDeSys и контроллером), код операции (например Login), часть данных операции (или все данные, если они помещаются в пакет) и номер фрагмента данных.
4. Сервис протокола внешней сети в контроллере, закончив входящую транзакцию сетевого протокола, по коду типа взаимодействия находит сервис RPC, обслуживающий взаимодействия данного типа, и передаёт ему данные транзакции для обработки, исключив код типа взаимодействия. Сервис RPC, обслуживающий взаимодействие между средой разработки и исполнения CoDeSys в контроллере, попросту копирует пришедшие данные в буфер коммуникационного драйвера среды исполнения CoDeSys.
5. Перед началом очередного цикла контроллера среда исполнения CoDeSys вызывает функцию чтения блока данных своего коммуникационного драйвера, которая обнаруживает, что пришли данные запроса от среды разработки или их часть. Как только данные запроса приняты полностью (на это может потребоваться несколько сетевых транзакций протокола внешней сети), по коду операции, пришедшему с запросом, вызывается соответствующий обработчик в среде исполнения CoDeSys, который делает, что ему положено сделать в зависимости от кода операции, после чего записывает в коммуникационный буфер данные ответа на запрос путём вызова соответствующей функции коммуникационного драйвера среды исполнения CoDeSys. Передача ответа на запрос из контроллера в сеть осуществляется в порядке, обратном описанному выше.

Следует отметить, что данные, пересылаемые между средой разработки и контроллером в процессе взаимодействия, могут не поместиться в коммуникационный буфер среды исполнения CoDeSys, что непременно происходит при загрузке больших программ и конфигураций, а также при загрузке и выгрузке файлов (команды меню **Online–Write file to PLC** и **Online–Read file from PLC**) и при чтении/записи большого количества переменных. Поэтому среда исполнения CoDeSys обязана предоставлять своему коммуникационному драйверу некий буфер, в который могли бы поместиться все данные запроса. Программа, загружаемая в контроллер из среды разработки, состоит из собственно исполняемого кода программы, снабжённого некоторой служебной информацией (размеры кода и данных, таблицы перераспределения адресов POU и символьных ссылок на функции внешних библиотек и т.п.), и бинарного дерева конфигурации контроллера, сгенерированного средой разработки на основании информации из секции PLC Configuration. В предельном случае среда исполнения должна уметь принять 64000 + 64512 = 128512 байт, что и делает наша адаптация среды исполнения CoDeSys. Однако при этом мы несколько ограничили возможности системы.

1. Во время загрузки новой программы исполнение имеющейся программы и опрос модулей ввода-вывода приостанавливаются.
2. Каждый раз выполняется загрузка как кода программы, так и бинарных данных конфигурации контроллера.
3. По окончании загрузки контроллер перезапускается и после успешного запуска новой программы в подавляющем большинстве случаев успешно возобновляет связь со средой разработки.

Как было указано ранее, интенсивный обмен по внутренней шине с большим количеством модулей ввода-вывода и большой объём вычислений в прикладной программе могут привести к потере пакетов от среды разработки, в результате чего практически неизбежен разрыв связи. При экстремальном тестировании системы нам легко удавалось создавать тестовые программы, при исполнении которых было невозможно сделать даже Login на контроллере, исполняющем прикладную программу и опрашивающем большое количество модулей ввода-вывода. А значит, для загрузки новой программы приходилось включать переключатель «1», перезапускать контроллер, менять сетевые параметры коммуникационного канала в среде разработки и только тогда делать Login и загружать новую программу.

Почему мы всегда заставляем среду разработки вместе с обновлённой программой загружать и всю конфигурацию, даже если пользователь ничего в ней не менял, несмотря на то что CoDeSys вроде бы позволяет этого не делать? Потому что мы категорически не согласны с тем, что идентичность конфигураций (особенно большого размера) допустимо устанавливать путем сравнения контрольных сумм, вычисленных по всем байтам конфигурации, имеющейся у среды разработки и присутствующей в контроллере. Мы полагаем, что различие (или идентичность) конфигураций должно выясняться более надёжным способом. Более того, вероятно, следовало бы выяснять и загружать только отличия между текущей и загружаемой конфигурациями.

Третье ограничение, по всей видимости, является наиболее спорным, особенно если учесть заявления большинства производителей ПЛК о том, что они поддерживают обновление программ в контроллерах без остановки контролируе-

мого процесса, а также тот факт, что среда исполнения CoDeSys обеспечивает возможность переключения на вновь загруженную программу на ходу (правда, для этого требуется лишних 64 кбайт оперативной памяти).

Мы считаем, что загрузка новой программы и переключение на неё в начале очередного цикла среды исполнения контроллера после окончания загрузки с сохранением значений переменных заменяемой программы в целом представляют собой небезопасную системную операцию. Описанная программная модель контроллера предполагает, что выходные данные, формируемые программой в некотором цикле, зависят от значений её входных данных в текущем цикле и переменных состояния, которые были вычислены данной программой в текущем и, быть может, предыдущих циклах выполнения. Оценка корректности реализации алгоритма обычно основывается на данном предположении, однако при использовании замены программы на лету выходные данные вновь загруженной программы могут зависеть также и от переменных состояния предыдущей заменённой программы, что существенно затрудняет оценку, ибо по сути в анализе должны присутствовать две версии программы. В ряде случаев совершенно корректная с алгоритмической точки зрения программа, будучи загруженной в контроллер, может начать работать неправильно по причине влияния на неё переменных состояния предыдущей заменённой программы.

Рассмотрим следующий пример.

Пусть, к примеру, в контроллере выполняется следующая программа:

```
TYPE STATE : (UNCERTAIN, START, RUN, MANUAL);
END_TYPE
PROGRAM SOME_DUMMY_REGULATOR
VAR_INPUT
  T : INT;
END_VAR
VAR_OUTPUT
  actuator1 : REAL;
  actuator2 : REAL;
END_VAR
VAR CONSTANT
  MANUAL_VALUE : REAL := 1.0;
  START_VALUE : REAL := 4.0;
  A : REAL := 0.2;
  B : REAL := -4.0;
  Tmin : INT := 40;
  Tmax : INT := 120;
  Tedge1 : INT := 60;
  Tedge2 : INT := 61;
END_VAR
VAR
  progState : STATE := UNCERTAIN;
END_VAR
(* Пользовательский код *)
CASE progState OF
  UNCERTAIN:
    IF T <= Tmin AND T > Tmin-2 THEN progState := START;
    END_IF
  START:
    actuator1 := START_VALUE;
    actuator2 := START_VALUE;
    progState := RUN;
  RUN:
    CASE T OF
      Tmin..Tedge1:
        actuator1 := A * INT_TO_REAL(T) + B;
      Tedge2..Tmax:
        actuator2 := A * INT_TO_REAL(T) + B;
      ELSE
        progState := MANUAL;
    END_CASE;
  MANUAL:
    IF T > Tmax OR T < Tmin THEN
      actuator1 := MANUAL_VALUE;
      actuator2 := MANUAL_VALUE;
    ELSE
      progState := START;
    END_IF
END_CASE;
END_PROGRAM
```

Некое подобие автомата здесь реализовано для того, чтобы намекнуть на возможно гораздо более сложные действия внутри пользовательского кода, чем те, что для краткости показаны в данном примере. Будучи реалистами, считаем, что система исполнения контроллера ничего не знает о смысле загруженной программы, а делает только то, что написано в коде, в зависимости от значений входных переменных и текущих значений переменных состояния.

В соответствии с описанным алгоритмом переключение на вновь загруженный обновлённый вариант данной программы осуществляется путём замены блока пользовательского кода, который начинается после комментария (* Пользовательский код *). Кроме того, необходимо заменить значения констант в секции VAR CONSTANT. Попробуем перечислить возможные изменения, которые могут быть внесены в программу перед online-загрузкой.

1. Изменение значений констант — типичный случай настройки параметров регулятора. Хотя может оказаться проще объявить эти параметры энергонезависимыми и менять их значения, не меняя и не загружая новую программу. Но и такой способ конфигурирования может привести к проблемам.
2. Удаление существующих переменных и/или констант и соответствующая коррекция кода алгоритма.
3. Добавление переменных и/или констант и соответствующая коррекция кода алгоритма.
4. Изменение типов существующих переменных и/или констант и соответствующая коррекция алгоритма.
5. Коррекция кода алгоритма, не затрагивающая состав и типы переменных. Вероятно изменение значений констант.
6. Изменение кода и состава переменных некоторых функциональных блоков (в смысле МЭК 61131-3). Функциональный блок сам по себе является типом, а, значит, может представляться в программе множеством экземпляров (объектов).

Смотрим, что будет в самом, на первый взгляд, простом случае, когда требуется изменить значения пары констант.

Делаем в изменённой программе Tedge1 := 50 и Tedge2 := 51. То есть по смыслу, когда новая программа находится в состоянии RUN, до тех пор, пока T пребывает в диапазоне от 40 до 50 включительно, на выходе actuator1 должно формироваться значение, пропорциональное T, в диапазоне от 4,0 до 6,0, а на выходе actuator2 должно удерживаться значение, равное START_VALUE, если T растёт, и 6,2 — если T уменьшается. Далее, когда T переходит в диапазон от 51 до 120 включительно, на выходе actuator1 удерживается значение 6,0, а на выходе actuator2 должно формироваться значение, пропорциональное T, в диапазоне от 6,2 до 20,0.

Пусть к моменту окончания загрузки значение T на входе старой программы было равным 57, то есть значение actuator1 было равным 7,4, а actuator2 — START_VALUE (4.0). После переключения на новую программу она остаётся в состоянии RUN, однако actuator1 имеет старое значение (7,4) вместо положенных по новому варианту алгоритма 6.0 и более не будет вычисляться, а значение actuator2 изменяется скачком с 4,0 на 7,4. Казалось бы, большое дело – изменить пару констант. Однако среда исполнения контроллера не в состоянии парировать возникшую логическую ошибку. Поэтому нужно либо загружать новую программу, подождав, когда старая окажется в состоянии MANUAL, либо когда T превысит 60, либо в коде программы учитывать, что она может быть заменена во время выполнения. В приведённом примере коррекция алгоритма, учитывающая возможность обновления на лету, неочевидна (необходимо устанавливать фиксированное значение на выходе, для которого не производится вычислений в текущем диапазоне T).

Рассмотрим еще один вариант «незначительных» изменений функционирующей программы. Речь пойдет о случае, когда не затрагиваются переменные и константы программы, а только «незначительно» изменяется ее код. Пусть в контроллере функционирует программа, исходный текст которой приведён ранее, и требуется поменять местами логику формирования значений выходных переменных actuator1 и actuator2. То есть фрагмент кода:

```
...
RUN:
  CASE T OF
    Tmin..Tedge1:
      actuator1 := A * INT_TO_REAL(T) + B;
    Tedge2..Tmax:
      actuator2 := A * INT_TO_REAL(T) + B;
    ELSE
      progState := MANUAL;
  END_CASE;
...
```

модифицируется следующим образом:

```
...
RUN:
  CASE T OF
    Tmin..Tedge1:
      actuator2 := A * INT_TO_REAL(T) + B;
    Tedge2..Tmax:
      actuator1 := A * INT_TO_REAL(T) + B;
    ELSE
      progState := MANUAL;
  END_CASE;
...
```

С точки зрения системы исполнения контроллера и пользователя, это мизерное изменение, не так ли? Но что будет, если переключение на новую программу произойдёт в момент, когда T равно, скажем, 66?

До обновления переменные состояния имеют следующие значения:

T = 66, progState = RUN, actuator1 = 8,0, actuator2 = 9,2.

После загрузки нового варианта, переключения на него и первого цикла

T = 66, progState = RUN, actuator1 = 9,2, actuator2 = 9,2.

Но по логике обновлённого алгоритма здесь должно быть

T = 66, progState = RUN, actuator1 = 9,2, actuator2 = 8,0.

Таким образом, опять получаем логическую ошибку прикладной программы, которая является побочным эффектом переключения на загруженную программу в начале нового цикла. Избежать этой ошибки можно либо дождавшись безопасного для изменений состояния старой программы, либо учитывая в прикладном алгоритме возможность его замены на лету.

Следующий вариант возможных изменений – добавление переменных и коррекция кода. Добавим коэффициент коррекции значений actuator1 и actuator2, вычисляемый в зависимости от значения и знака производной температуры. Новая версия программы, которую нужно загрузить на лету, может выглядеть так:

```
PROGRAM SOME_DUMMY_REGULATOR
VAR_INPUT
  T : INT;
END_VAR
VAR_OUTPUT
  actuator1 : REAL;
  actuator2 : REAL;
END_VAR
VAR CONSTANT
  MANUAL_VALUE : REAL := 4.0;
  START_VALUE : REAL := 5.0;
  A : REAL := 0.2;
  B : REAL := -4.0;
  Tmin : INT := 40;
  Tmax : INT := 120;
  Tedge1 : INT := 60;
  Tedge2 : INT := 61;
  T_UNKNOWN : INT := -32768;
END_VAR
VAR
  progState : STATE := UNCERTAIN;
  deltaT : INT := T_UNKNOWN;
  Tprev : INT := T_UNKNOWN;
  Kcorr : REAL := 1.0;
END_VAR
(* Пользовательский код начинается отсюда *)
CASE progState OF
  UNCERTAIN:
    IF T < Tmax AND T > Tmin AND deltaT <> T_UNKNOWN THEN
      progState := START;
    END_IF
  START:
    actuator1 := START_VALUE;
    actuator2 := START_VALUE;
    progState := RUN;
  RUN:
    IF deltaT > 0 THEN
      Kcorr := 1.0 + INT_TO_REAL(deltaT)/ INT_TO_REAL(T);
    ELSIF deltaT < 0 THEN
      Kcorr := 1.0 - INT_TO_REAL(deltaT)/ INT_TO_REAL(T);
    ELSE
      Kcorr := 1.0;
    END_IF
    CASE T OF
      Tmin..Tedge1:
        actuator1 := Kcorr * (A * T + B);
      Tedge2..Tmax:
        actuator2 := Kcorr * (A * T + B);
    ELSE
      progState := MANUAL;
    END_CASE;
  MANUAL:
    IF T > Tmax OR T < Tmin THEN
```

```
      actuator1 := MANUAL_VALUE;
      actuator2 := MANUAL_VALUE;
    ELSE
      progState := START;
    END_IF
END_CASE;

IF Tprev <> T_UNKNOWN THEN deltaT := T - Tprev;
END_IF
Tprev := T;
END_PROGRAM
```

Обращаем внимание, что при изменении состава переменных программы при переключении на вновь загруженную программу невозможно просто дать новой программе использовать значения переменных старого варианта программы, – нужно сделать что-то ещё. Будем считать, что в системе исполнения контроллера имеется некий механизм разрешения такой ситуации. Пусть загрузка приведённого нового варианта программы успешно завершилась в момент, когда старая программа находилась в состоянии RUN, а значение T было равным 81. Тогда после переключения на новую программу после первого цикла значения её переменных состояния и выходных переменных будут следующими:
actuator1 = 8,0, Kcorr = 405,543, actuator2 = 4947,6246,
Tprev = 81, deltaT = –32768.

Таким образом, алгоритмически корректная программа, загруженная в контроллер, по крайней мере два цикла будет формировать на выходе actuator2 значение, равное температуре теплового излучения при ядерном взрыве. Для того чтобы избежать проявления данной ошибки, нужно, чтобы разработчиком в коде была учтена возможность замены на лету.

Последний вариант касается случая, когда меняется тип какой-нибудь переменной и соответствующим образом корректируется код. Пусть, к примеру, принято решение в переменной state хранить не только текущее состояние программы, но и значение кода команды, поступающей по сети:

```
TYPE COMPOUND_STATE :
  STRUCT
    command : BYTE;
    state : STATE;
  END_STRUCT
END_TYPE
PROGRAM SOME_DUMMY_REGULATOR
...
VAR
  progState : COMPOUND_STATE;
...
END_VAR
(* Пользовательский код начинается отсюда *)
CASE progState.state OF
...
END_CASE;
...
END_PROGRAM
```

Очевидно, в таком случае не существует ни одного приемлемого способа возобновить работу программы после загрузки без её полного перезапуска.

Для приведённого примера характерно наличие в заменяемой программе информации о её состоянии на предыдущих циклах исполнения. Если выходные данные, формируемые программой, зависят от входных данных и переменных состояния только на текущем цикле, то переключение на новую версию программы при незначительных изменениях скорее всего не приведет к проблемам. Проблема, однако, в том, что среда исполнения контроллера не знает, зависит ли правильность алгоритма, реализуемого загруженной программой, от состояния на предыдущих циклах исполнения. По идее, среда исполнения могла бы предоставлять прикладной программе функцию, вызов которой информировал бы среду исполнения о допустимости переключения на новую программу. Тогда пользователю не пришлось бы ждать, когда заменяемая программа окажется в допустимом для замены состоянии. Кроме того, вновь загруженная программа, по всей видимости, также могла бы иметь возможность указать среде исполнения момент, когда допустимо произвести переключение, например, при выполнении некоторого условия. А до достижения такого подходящего момента она могла бы выполняться параллельно со старой программой, как бы обучаясь, и при этом не передавая данные ни одному из потребителей данных этой программы.

Пример:

```
PROGRAM SOME_DUMMY_REGULATOR
...
VAR
        progState : COMPOUND_STATE;
...
END_VAR
(* Пользовательский код начинается отсюда *)
CASE progState.state OF
        UNCERTAIN:
         ...
        START:
         ...
        RUN:
         ...
        MANUAL:
         (* Переключение на эту программу допустимо *)
         SwitchToThisProgramAllowed(TRUE);
         (* Переключение этой программы на новую допустимо *)
         SwitchFromThisProgramAllowed(TRUE);
END_CASE;
...
END_PROGRAM
```

В качестве аргументов SwitchToThisProgram()/SwitchFromThisProgram() могут использоваться любые выражения с результатом типа BOOL.

Конечно же, описанный возможный механизм не прост в реализации и имеет определённые ограничения, однако ничего подобного нет ни в стандарте МЭК 61131-3, ни в среде исполнения CoDeSys.

Таковы причины, по которым мы ограничили возможности взаимодействия между средой разработки и средой исполнения CoDeSys в контроллерах Fastwel I/O в части обновления программ на лету, а также не поддержали переменные типа VAR PERSISTENT.

Автор благодарит И. Петрова, технического директора ПК «Пролог», за весьма ценные замечания и предложения по содержанию данной части статьи, а также за оперативную помощь в решении технических проблем, возникавших у нас в процессе работы над проектом.

Автор — сотрудник фирмы Fastwel
Телефон: +7 (495) 234-0639
E-mail: info@fastwel.ru
Web: www.fastwel.ru