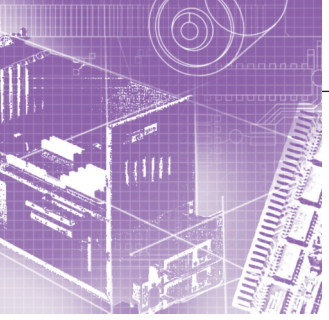

Fastwel I/O: развитие продуктовой линейки

Часть 2. Коммуникационные возможности

Александр Локотков

В статье рассматриваются особенности реализации и применения коммуникационных протоколов MODBUS и MODBUS TCP в контроллерах FASTWEL I/O.

Введение

Продуктовая линейка FASTWEL I/O в официальных документах имеет название «распределённая система ввода-вывода», которое ей было присвоено почти 12 лет назад во многом под влиянием роста популярности систем класса Distributed Control System (DCS, или распределённых систем управления). В связи с этим у некоторых специалистов в области АСУ ТП при первом знакомстве с продуктом может сложиться впечатление, что функциональное назначение FASTWEL I/O ограничивается выполнением функций устройства связи с объектом, обеспечивающего обмен данными между программируемыми контроллерами или системами класса SCADA/HMI и датчиками/исполнительными механизмами по одной из полевых шин.

Хотелось бы, однако, подчеркнуть, что FASTWEL I/O является семейством программируемых контроллеров с переменным составом модулей, способных функционировать совершенно автономно и имеющих коммуникационные интерфейсы для интеграции с верхним уровнем АСУ ТП, а также для создания распределённых систем сбора данных и управления. При этом любой из встроенных коммуникационных интерфейсов контроллеров может быть использован для взаимодействия с интегрированной средой разработки CoDeSys 2.3, включая загрузку и обновление приложений, удалённую отладку и мониторинг переменных, а также для загрузки/выгрузки файлов и обновления системного программного обеспечения контроллеров и модулей ввода-вывода.

В настоящее время в контроллерах FASTWEL I/O имеется встроенная поддержка нескольких промышленных сетевых протоколов, включая CAN/CANopen, PROFIBUS DP, MODBUS RTU/ASCII, MODBUS TCP и DNP3. Более того, коммуникационные возможности каждого контроллера могут быть расширены путём использования коммуникационных модулей, подключаемых к межмодульной шине FBUS контроллера наряду с другими модулями ввода-вывода, и системных библиотек, добавляемых в проекты CoDeSys 2.3.

При проектировании систем сбора данных и управления разработчики довольно часто задаются следующими вопросами, связанными с выбором сетевого протокола и построением промышленной сети.

1. Каким образом передать значения переменных прикладных алгоритмов между разными узлами сети?
2. Как реализовать логику команд аналогового и дискретного управления, выполняемых на разных узлах сети?
3. Возможно ли передать данные между приложениями на разных узлах сети в объёме, достаточном для решения поставленной задачи, и при этом удовлетворить требования к частоте обновления данных?
4. Как повлияют на работу сети кратковременные или постоянные отказы её отдельных узлов?

В данной статье делается попытка облегчить поиск ответов на эти вопросы применительно к промышленному протоколу MODBUS с учётом особенностей его реализации в контроллерах FASTWEL I/O.

MODBUS

Общие сведения

Протокол MODBUS[1], спецификация которого впервые была опубликована фирмой Modicon в 1979 г., до сих пор считается одним из наиболее распространённых сетевых решений для АСУ ТП. Основной причиной этого феномена, скорее всего, является доступность MODBUS для широкого круга производителей и пользователей, с которой вряд ли может соперничать какой-либо из современных промышленных сетевых протоколов. Доступность MODBUS складывается главным образом из низкой стоимости решения, а также простоты для освоения и реализации, даже если говорить о MODBUS TCP – его более современном воплощении для сетей TCP/IP.

В самом деле, для получения спецификаций протоколов семейства MODBUS не требуется вступать в какие-либо организации, платить взносы и отчисления, а достаточно зайти на веб-сервер http://www.modbus.org, согласиться с предложенными условиями предоставления информации и загрузить документы.

[1] Обратите внимание, что название протоколов данного семейства в спецификациях пишется прописными буквами: MODBUS. Весьма распространённое до недавнего времени написание Modbus формально используется только в логотипе и названии организации Modbus Organization.

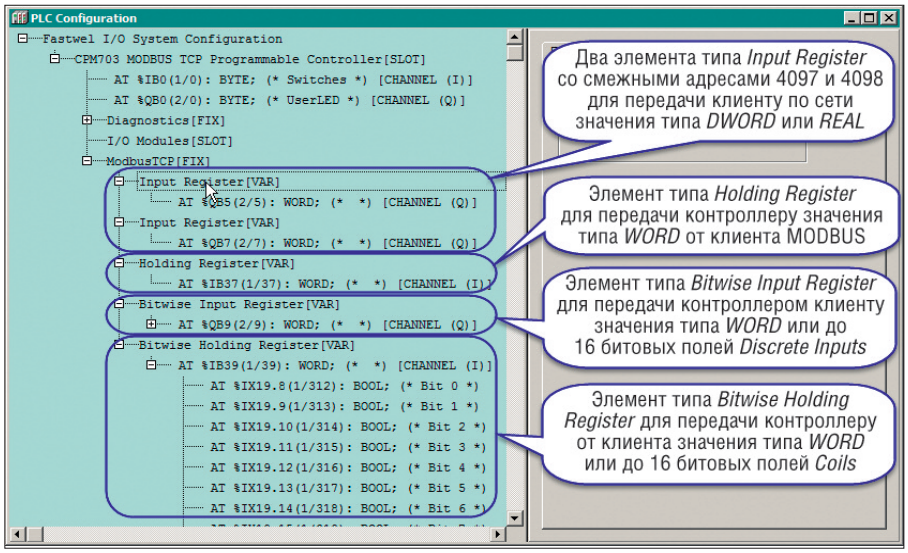

Рис. 1. Регистровая модель отображения переменных на регистры и битовые поля MODBUS

MODBUS базируется на обычных интерфейсах RS-232C/RS-485, а MODBUS TCP функционирует поверх стандартного сетевого транспорта TCP/IP, поэтому производителям оборудования и программного обеспечения для АСУ ТП не нужно приобретать специализированные микросхемы и дорогостоящие реализации стека протоколов. В этой связи на рынке АСУ ТП присутствует огромное количество контроллеров, сетевых шлюзов, датчиков и программного обеспечения, поддерживающих тот или иной вариант протокола MODBUS.

Наконец, в основе протоколов MODBUS лежит простой для понимания принцип работы: мастер сети, далее называемый клиентом, считывает и записывает данные подчинённых узлов (далее – серверов), представленные так называемыми регистрами типа *Input Register* и *Holding Register* размером 2 байта каждый и/или битовыми полями типа *Discrete Input* и *Coil*. В сети MODBUS RTU или ASCII может быть один клиент и до 246 серверов, а в сети MODBUS TCP максимальное количество клиентов и серверов по существу ограничено только возможностями реализации, поставляемой конкретным производителем. С точки зрения приложения, выполняющегося на узле-клиенте MODBUS, сеть выглядит как область памяти, доступ к ячейкам которой осуществляется по логическим адресам, состоящим из сетевого адреса узла-сервера, номера регистра или битового поля и количества регистров или битовых полей, к которым нужно получить доступ в текущем запросе. Узлы-серверы читают и обновляют свои участки этой сетевой памяти, и тем самым обеспечивается обмен данными между приложениями на разных узлах сети.

Разумеется, за простотой и доступностью протоколов семейства MODBUS скрыта масса особенностей и ограничений, которые необходимо учитывать при построении той или иной системы, причём особенности и ограничения присущи как самому протоколу, так и его конкретным реализациям в оборудовании и программном обеспечении разных производителей.

Особенности реализации MODBUS в контроллерах FASTWEL I/O

В контроллерах CPM702 и CPM703 первого поколения линейки FASTWEL I/O в качестве основных сервисов внешней сети реализованы сервер MODBUS RTU/ASCII и сервер MODBUS TCP соответственно. На сегодняшний день основным отличием сетевых средств контроллеров CPM70x по сравнению с версией 2008 г. является возможность реализации до четырёх серверов MODBUS в приложении CoDeSys 2.3 для любого из контроллеров средствами библиотеки FastwelModbusServer.lib. Для этого могут использоваться коммуникационные порты на базе модулей NIM741/NIM742 или сервисный порт, расположенный под пластиковой крышкой на передней панели контроллера.

Контроллеры семейства CPM71x и модульные компьютеры MK905-01\CDS и MK905-03\CDS с установленной системой исполнения приложений CoDeSys 2.3 имеют значительно более развитые сетевые сервисы MODBUS и MODBUS TCP по сравнению с младшими моделями CPM70x.

Основной интерфейс внешней сети контроллера CPM712 может быть сконфигурирован для работы либо в качестве сервера MODBUS RTU или ASCII, либо в качестве клиента (мастера). Сетевой интерфейс Ethernet контроллера CPM713 может применяться в качестве сервера и клиента MODBUS TCP одновременно, что позволяет создавать распределённые системы сбора данных и управления без главного вычислительного устройства, роль которого обычно играет более мощный ПЛК или промышленный компьютер. Более того, одновременно с клиентом и сервером MODBUS TCP на контроллере CPM713 может функционировать один или несколько пользовательских сетевых протоколов поверх TCP и/или UDP, реализуемых в приложении CoDeSys 2.3 средствами системной библиотеки FastwelSysLibSockets.lib. К перечисленным сетевым возможностям CPM71x можно добавить до четырёх дополнительных серверов MODBUS средствами уже упомянутой библиотеки FastwelModbusServer.lib через последовательные порты на базе NIM741/NIM742 или через сервисный порт.

Сервер MODBUS и MODBUS TCP контроллеров CPM712 и CPM713 имеет так называемую плоскую модель отображения переменных приложения на регистры и битовые поля MODBUS, которая, по мнению большинства пользователей, значительно более естественна и удобна, чем регистровая модель, используемая в контроллерах CPM702 и CPM703. Однако для облегчения модернизации ранее внедрённых систем контроллеры CPM712/CPM713 также поддерживают регистровую модель, что позволяет без изменений в конфигурации сети использовать проекты, ранее разработанные для CPM702 и CPM703, в контроллерах CPM712 и CPM713.

Регистровая модель отображения переменных предполагает, что пользователь в окне ресурса **PLC Configuration** проекта CoDeSys 2.3 добавляет в конфигурацию сервера MODBUS множество элементов типа *Input Register, Holding Register, Bitwise Input Register* и *Bitwise Holding Register*, каждый из которых описывает два байта в области входных и выходных данных приложения, как показано на рис. 1.

Элементы *Input Register* и *Holding Register* описывают соответствующие регистры сервера, причём входными и выходными они являются по отношению к клиенту (мастеру) MODBUS. Чтение данных, отображённых в приложении контроллера на элементы *Input Register*, осуществляется клиентом MODBUS при помощи функции 04, а для доступа к

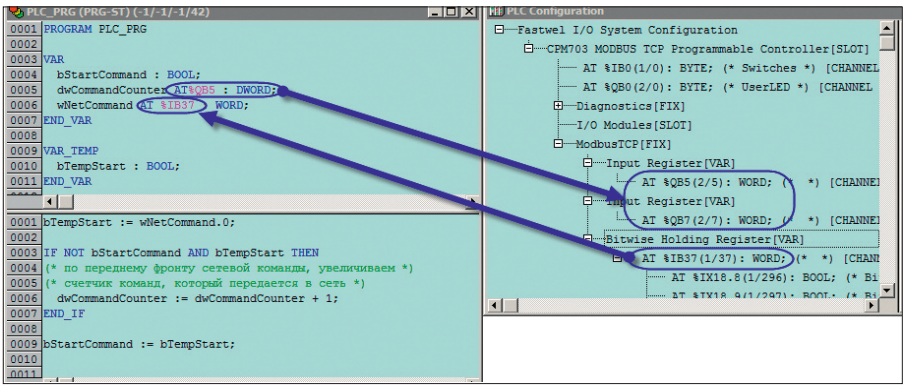

Рис. 2. Пример отображения переменных на регистры MODBUS для регистровой модели

данным, отображённым на элементы *Holding Register*, могут использоваться функции 03 и 06 (чтение), 16 (запись), 22 (запись с маской) и 23 (чтение и запись). В качестве начального адреса регистра в сетевом запросе должен использоваться адрес, на единицу меньший заданного в свойствах регистра в ресурсе **PLC Configuration.** Для приложения контроллера данные, отображаемые на адреса элементов *Input Register* и *Holding Register* в образе процесса, являются выходными и входными соответственно.

Элементы с префиксом *Bitwise* представляют так называемые делимые входные и выходные регистры. Под делимостью понимается возможность сетевого доступа к отдельным битовым полям внутри слова данных делимого регистра. Доступ к данным, отображённым на делимые регистры, со стороны клиентов MODBUS может осуществляться как с помощью регистровых функций чтения или записи (03, 04, 06, 16, 22, 23), так и посредством операций доступа к битовым полям (01, 02, 05, 15). Для доступа к любому из 16 битовых полей делимого регистра должен использоваться адрес, вычисленный по формуле:

Адрес битового поля = (Адрес регистра – 1)×16 + Номер битового поля в слове (начиная с 1)

Контроллеры CPM702 и CPM703 могут содержать в конфигурации приложения до 512 регистров разных типов и поддерживают групповые операции чтения (03, 04) до 125 регистров и записи (16) до 123 регистров за один запрос, а также чтения до 2000 битовых полей типа *Discrete Input* или *Coil* и записи до 1968 битовых полей типа *Coil*. Кроме того, поддерживается весьма эффективная с точки зрения утилизации сети операция чтения и записи 23, выполняемая за один сетевой запрос и позволяющая записать до 121 и прочитать до 125 регистров.

Для отображения некоторой переменной приложения на регистры и/или битовые поля сервера MODBUS пользователь должен снабдить декларацию переменной в коде приложения директивой ссылки на адрес в образе процесса *AT%*, после которой следует спецификатор области образа процесса: *I* (для *Holding*-регистров) или *Q* (для *Input*-регистров), затем спецификатор шага смещения: *B* (байт), *W* (слово), *D* (двойное слово) или *X* (бит), и, наконец, смещение участка образа процесса, занимаемого регистрами/битовыми полями. Пример отображения входной переменной приложения типа WORD и выходной переменной типа DWORD представлен на рис. 2. Обратите внимание, что MODBUS-адреса двух элементов типа *Input Register* в конфигурации приложения, на которые отображена переменная длиной более 2 байт, должны быть смежными, то есть MODBUS-адрес регистра для адреса *%QB5* в образе процесса должен быть на единицу меньше адреса регистра для адреса *%QB7* в образе процесса.

При отображении переменных типа массив (*ARRAY*), строка (*STRING*) и структура (*STRUCT*) и формировании запросов клиентов на чтение и запись этих переменных необходимо следить за тем, чтобы размер переменной, отображаемой на несколько входных (*Input*) регистров со смежными адресами, не превышал 250 байт, а для выходных (*Holding*) регистров – 246 байт. Это связано с тем, что групповой запрос чтения регистров может содержать не более 125 2-байтовых слов данных, а групповой запрос записи – не более 123 2-байтовых слов. Кроме того, чтение и запись переменных размером свыше двух байтов должны выполняться одним сетевым запросом по наименьшему MODBUS-адресу, на который отображена переменная. При нарушении любого из указанных правил практически неизбежно нарушение целостности данных, передаваемых серверу или получаемых от сервера, то есть клиент или сервер в какой-то момент времени может получить часть значения переменной, сформированную в один момент времени, а другую часть – в другой, что приведёт к неправильной работе или даже к аварийному завершению приложения.

Практика применения контроллеров CPM702 и CPM703 показала, что пользователям не всегда удобно формировать списки регистров, задавать их адреса и следить за тем, чтобы они оставались уникальными и отличными друг от друга на 1, даже несмотря на наличие специальных элементов пользовательского интерфейса (**Панель свойств Fastwel**) в пакете адаптации CoDeSys 2.3, позволяющих автоматически назначать смежные адреса регистрам. Кроме того, разработчики приложений CoDeSys довольно часто декларируют переменные непосредственно в ресурсе **PLC Configuration** в виде символических ссылок на адреса образа процесса. Очевидно, что при использовании регистровой модели непосредственно в окне ресурса **PLC Configuration** возможно объявить переменную длиной не более 2 байт.

В итоге для контроллеров CPM712, CPM713 и MK905-01,03\CDS в качестве основной была выбрана плоская модель отображения переменных на регистры MODBUS. В плоской модели сервер MODBUS связан с приложением CoDe-

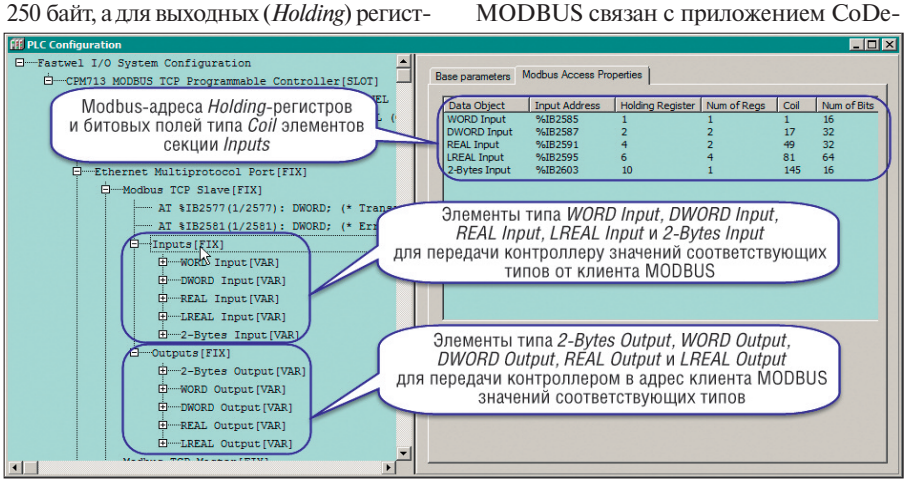

Рис. 3. Плоская модель отображения переменных на регистры и битовые поля MODBUS

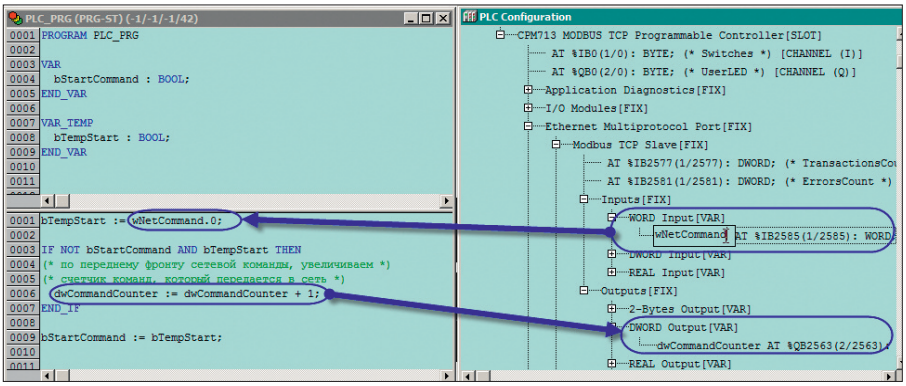

Рис. 4. Пример отображения переменных на регистры MODBUS для плоской модели

Sys 2.3 через две области памяти в образе процесса, входную и выходную, размером 8 кбайт каждая.

Для отображения переменных приложения на *Holding*-регистры и битовые поля типа *Coil* служит входная область памяти образа процесса, определяемая секцией *Inputs* в конфигурации сервера в ресурсе **PLC Configuration**, а для отображения переменных на *Input*-регистры и битовые поля типа *Discrete Input* используется выходная область памяти образа процесса в секции *Outputs*, как показано на рис. 3. То есть секция *Inputs* определяет входные данные для приложения со стороны сервера MODBUS, а секция *Outputs* – выходные данные приложения в сторону удалённых клиентов MODBUS.

Работа с секциями *Inputs* и *Outputs* основана на общем принципе, который будет далее объяснён на примере секции *Inputs*, поскольку отличие секции *Outputs* от *Inputs* состоит в том, что она служит для передачи данных приложения контроллера клиентам MODBUS, и декларации адресов в образе процесса, относящиеся к добавляемым в неё элементам, имеют спецификатор *%Q* в отличие от спецификатора *%I*, используемого в элементах секции *Inputs*.

Для того чтобы приложение контроллера могло получать данные по сети от клиентов MODBUS, в секцию *Inputs* должны быть добавлены элементы *WORD Input, DWORD Input, REAL Input* или *LREAL Input*, каждый из которых содержит входной канал соответствующего типа МЭК 61131-3: *WORD, DWORD, REAL u LREAL*. Входные переменные приложения затем могут быть отображены на адреса входных каналов добавленных элементов таким же образом, как это делается при использовании регистровой модели. Кроме того, можно объявить входные переменные типа *WORD, DWORD, REAL* или *LREAL* непосредственно в ресурсе **PLC Configuration**, дважды щёлкнув левой кнопкой мыши слева от директивы *AT%* каналов, и ввести имена переменных, как показано на рис. 4. В таком случае при вставке/удалении элементов секции *Inputs* не потребуется следить за сдвигом адресов и корректировать их значения в декларациях переменных в коде приложения, но может потребоваться скорректировать адреса регистров в конфигурации клиентов MODBUS.

Для удобства отображения входных булевых переменных приложения и переменных типа *BYTE* служит элемент типа *2-Bytes Input*, который содержит два входных канала типа *BYTE*, каждый из которых делится на восемь битовых полей типа *BOOL*. Размер 2 байта, установленный для данного элемента, позволяет автома-

тически выровнять на одно слово все переменные, отображаемые на регистры MODBUS, для обеспечения неразрывности (целостности) значений переменных длиной более двух байтов. Следует отметить, что каналы типа *BOOL* в CoDeSys 2.3 на самом деле никогда фактически не создаются, и отображение переменной типа *BOOL*, фактическая длина которой составляет 1 байт, на битовое поле в образе процесса в реальности приводит к генерации кода, извлекающего значение бита по адресу *%IX**, которое затем копируется в младший бит переменной типа *BOOL*, ссылающейся на адрес *%IX**. В результате код приложения становится менее эффективным как с точки зрения скорости выполнения, так и с точки зрения размера занимаемой памяти в сегменте кода.

Сетевой доступ к переменным приложения, отображённым на область памяти секции *Inputs* в конфигурации сервера MODBUS, может осуществляться посредством любых запросов чтения и записи *Holding*-регистров и битовых полей типа *Coil*, причём это относится к любому элементу, добавленному в секцию *Inputs*. Адрес MODBUS и количество объектов, которые должны использоваться в запросе чтения и/или записи, отображается на вкладке **Modbus Access Properties** в окне **PLC Configuration**, если щёлкнуть на секции *Inputs* или на любом добавленном в неё элементе в дереве **PLC Configuration**.

При этом для каждого элемента на вкладке отображаются как начальный адрес регистра (столбец *Holding Register*) и количество регистров (столбец *Num of Regs*), так и начальный адрес битового поля типа *Coil* и количество битовых полей в запросе. Однако обратите внимание, что отображаемые адреса регистров и битовых полей на 1 больше значений адресов, которые должны фактически передаваться в сетевых запросах клиентов. Так сделано по причине того, что в спецификации прикладного уровня протокола MODBUS нумерация регистров и битовых полей как объектов прикладного уровня протокола начинается с 1, а их адреса в сетевых запросах – с 0. Кроме того, во многих клиентских приложениях при конфигурировании традиционно принято нумеровать регистры и битовые поля с 1.

Всё, что было сказано по поводу секции *Inputs*, справедливо для секции *Outputs*, с той лишь разницей, что секция *Outputs* служит для передачи данных от контроллера (вернее, от приложения, исполняющегося на контроллере) в адрес клиентов MODBUS в ответ на запросы чтения *Input*-регистров и битовых полей типа *Discrete Input*.

Таким образом, при отображении переменных приложения на регистры и битовые поля MODBUS в контроллерах CPM712 и CPM713 не требуется явно задавать адреса регистров, следить за их уникальностью и смежностью, если предполагается читать и/или записывать множество значений переменных в одном сетевом запросе. Однако как для плоской, так и для регистровой модели отображения переменных на регистры характерна особенность, о которой уже говорилось ранее: максимальный размер переменной, которую предполагается читать по MODBUS запросами 03 (Read Holding Registers), 04 (Read Input Registers) или 23 (Read/Write Multiple Registers), не должен превышать 250 байт, а максимальный размер переменной, чьё значение требуется изменять по MODBUS запросом 16 (Write Multiple Registers), не должен превышать 246 байт (для запроса 23 – не более 242 байт), иначе будет невозможно обеспечить неразрывность значений переменных при транспортировке от клиентов серверу и от сервера клиентам.

Сервер MODBUS контроллеров CPM-712 и CPM713 обеспечивает доступ

Таблица 1

Максимальные значения задержки ответа на запрос MODBUS RTU (T_{SDR})

Скорость обмена, бит/с	Задержка ответа на запрос в режиме RTU, мс	
	CPM702	CPM712
19200	3,2	14,9
38400	2,0	13,9
57600	1,6	13,4
115200	1,2	13,0

Таблица 3

Измеренные значения времени обмена MODBUS RTU (функция 4, 125 регистров)

Скорость обмена, бит/с	T_S, мс	
	CPM702	CPM712
19200	153,5	166,3
38400	76,2	89,1
57600	51,6	64,1
115200	26,2	38,1

к входным и выходным переменным приложения суммарными размерами по 8 кбайт, чем значительно превосходит CPM702 и CPM703, а также оставляет далеко позади всех ближайших конкурентов. В связи с этим могут возникнуть вопросы, сколько времени может потребоваться на чтение и запись такого количества данных по сети и не влияет ли отрицательно столь большая информационная ёмкость контроллеров CPM712 и CPM713 на время задержки ответа контроллера на сетевой запрос.

Оценка времени обмена по сети

Таблица 1 содержит максимальные значения задержки ответа для контроллеров CPM702 и CPM712 при наиболее часто используемых скоростях обмена и формате символа, состоящего из одного стартового бита, 8 бит данных, одного стопового бита и одного бита контроля по чётности.

Время задержки ответа на запрос MODBUS для контроллера CPM702 составляет длительность, необходимую для передачи от 5,5 до 12,5 символов по линии передачи RS-485 при выбранной скорости обмена, из которых длительность в 3,5 символа составляет интервал тишины после приёма последнего байта входящего пакета, по истечении которого сервис протокола MODBUS CPM702 решает, что очередной пакет полностью принят, а оставшееся время, эквивалентное времени передачи до 9 символов, необходимо для полной обработки принятого пакета и начала передачи ответа.

Время задержки ответа на запрос MODBUS контроллера CPM712 на 11 с лишним миллисекунд больше, чем у CPM702. Это связано с тем, что изначальная длительность задержки ответа CPM712, равная всего лишь 560 мкс, оказалась слишком короткой для некоторых

Таблица 2

Оценки времени обмена (функция 4, 125 регистров)

Скорость обмена, бит/с	T_{REQ}, мс	T_{SDR}, мс		T_{RES}, мс	$T_{3.5}$, мс	T_S, мс	
		CPM702	CPM712			CPM702	CPM712
19200	4,58	3,2	14,9	146,09	2,01	155,88	167,58
38400	2,29	2,0	13,9	73,05	1,00	78,34	90,24
57600	1,53	1,6	13,4	48,70	0,67	52,49	64,29
115200	0,76	1,2	13,0	24,35	0,33	26,65	38,45

клиентов MODBUS RTU, которые просто не успевали переключить свой приёмо-передатчик интерфейса RS-485 с передачи на приём по завершении передачи пакета запроса в адрес CPM712. CPM712 начинал отвечать на запрос в момент, когда линия передачи ещё занята клиентом, и несколько символов в пакете ответа искажались. Именно поэтому порт интерфейса RS-485 контроллера CPM712 пришлось искусственно «замедлить».

Для оценки времени обмена мастера сети с контроллером T_S нужно для выбранной скорости обмена определить время передачи пакета запроса от клиента серверу T_{REQ}, прибавить к нему время задержки ответа на запрос T_{SDR} в соответствии с таблицей 1 и время передачи пакета ответа от сервера клиенту T_{RES}:

$$T_S = T_{REQ} + T_{SDR} + T_{RES} + T_{3.5}$$

В данном соотношении параметр $T_{3.5}$ является временем передачи 3,5 символов при выбранной скорости обмена, по истечении которого сервис клиента MODBUS RTU принимает решение о завершении приёма пакета от удалённого сервера.

Пакет запроса от клиента серверу в режиме RTU состоит из адреса сервера (1 байт), кода функции (1 байт), данных запроса, включая служебные поля для выбранной функции (от 0 до 252 байт) и поля циклической контрольной суммы (CRC) длиной 2 байта.

Пакет ответа сервера клиенту на правильный запрос состоит из адреса сервера (1 байт), кода функции (1 байт), размера данных (1 байт), самих данных и поля CRC длиной 2 байта.

Например, длина пакета запроса MODBUS RTU Read Input Registers (функция 04) составляет 8 байт, а длина ответа на запрос чтения 125 регистров имеет длину 255 байт. Оценки времени обмена с контроллерами CPM702 и CPM712 для данного типа запроса приведены в таблице 2.

Измеренные значения времени обмена с контроллерами CPM702 и CPM712 для этого же запроса представлены в таблице 3. На рис. 5 показан метод измерения времени обмена при помощи осциллографа, подключённого к линии RS-485. Обратите внимание, что результаты измерений не учитывают длительность 3,5 символов тишины $T_{3.5}$, по которому клиент MODBUS принимает решение о завершении приёма ответа на последний запрос.

Для контроллеров CPM703 и CPM713 с интерфейсом Ethernet и протоколом MODBUS TCP оценка времени обмена в общем довольно затруднительна, поскольку в расчёте практически невозможно учесть влияние множества факторов, таких как особенности реализации стека TCP/IP на клиенте, латентность коммутационного оборудования и наличие в сети другого трафика помимо MODBUS TCP. Однако измерить время обмена между компьютером и контроллером можно с помощью свободно распространяемой утилиты Wireshark (http://www.wireshark.org), которая, помимо прочих удобств, имеет встроенные средства анализа и трассировки протокола MODBUS TCP. Окно утилиты Wireshark при измерении времени обмена с контроллером CPM713 для функции 4 (Read Input Registers) показано на рис. 6. Столбец *Time* содержит временну́ю метку в секундах с точностью до микросекунд для каждого захваченного пакета Ethernet относительно момента начала захвата пакетов. В столбцах *Source* и *Destination* отображаются IP-адреса источника и получателя запроса соответственно. Столбец *Info* содержит расшифровку запросов и ответов с префиксами *Query* (запрос) и *Response* (ответ). Обратите внимание, что в столбце *Info* Wireshark отображает код функции MODBUS, передаваемый в пакетах запроса и ответа. Столбец *Length* при этом содержит значения длины пакетов запроса и ответа. Время обмена для каждой транзакции может быть вычислено путем вычитания времени запроса (*Query*) из времени ответа (*Response*).

Для контроллера CPM713 время обмена при чтении значений 120 регистров одним запросом составляет около 450 мкс, а для контроллера CPM703 около 2,3 мс.

При наличии в сети нескольких контроллеров суммарное время обмена по сети можно оценить как сумму времён обмена с каждым контроллером для

всех сетевых запросов. Однако с этого момента оценки становятся весьма оптимистическими, поскольку довольно часто промышленные сети функционируют далеко не в идеальных условиях, в результате чего пакеты запросов и ответов на запросы могут не доходить до адресатов. Причины этого могут быть самыми разнообразными: помехи в линии передачи, искажающие данные одного или нескольких пакетов, неработоспособность отдельных узлов сети ввиду вывода из работы части основного технологического оборудования и т.п., но общим местом для таких ситуаций является отсутствие немедленного ответа одного из подчинённых узлов (серверов) на очередной запрос мастера (клиента).

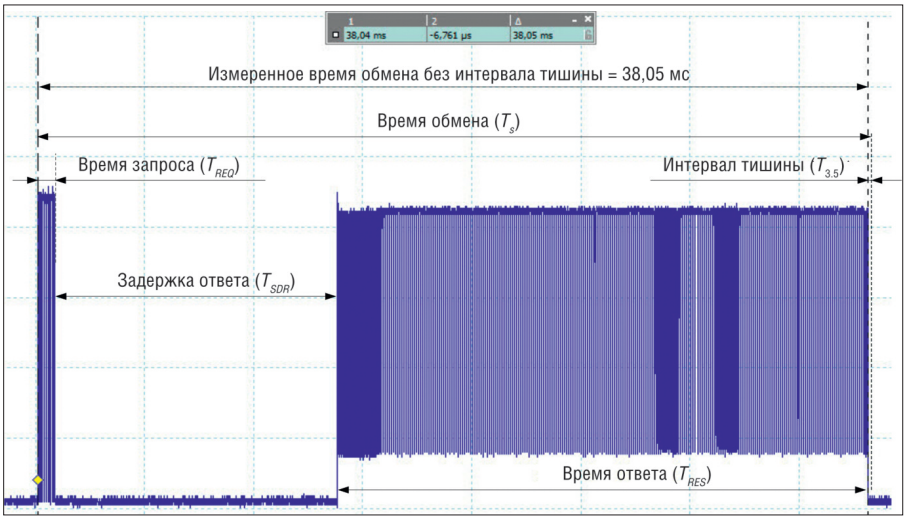

Рис. 5. Метод измерения времени обмена по MODBUS (CPM712, 115200 бит/с)

Клиент MODBUS после передачи запроса серверу, как правило, ожидает ответа в течение некоторого времени. Длительность ожидания обычно задаётся для каждого сервера при конфигурировании клиента в виде тайм-аута ответа, и нередки случаи, когда этому параметру не уделяется должного внимания при проектировании и наладке систем сбора данных и управления.

Оценка времени обмена по сети со всеми серверами для наихудшего случая, помимо суммы времён обмена с каждым из серверов, должна также учитывать сумму тайм-аутов ответа для всех сетевых запросов, которые могут быть переданы всем серверам, в предположении, что все они, кроме текущего опрашиваемого сервера, вдруг перестали отвечать. Даже если клиентское приложение MODBUS или устройство с функцией мастера MODBUS поддерживает режим исключения отдельных подчинённых узлов из расписания опроса при отсутствии от них ответов на запросы (так называемый режим *Demotion* или *Auto-Demotion*), до активизации данного режима для некоторого сервера MODBUS, который перестал отвечать, время обмена данными по сети увеличится на длительность тайм-аута ответа, заданного для переставшего отвечать сервера, умноженную на количество неудачных запросов к данному серверу, после выполнения которых сервер будет исключён из расписания обмена.

Приведённые рассуждения иллюстрируются простым примером. Пусть в

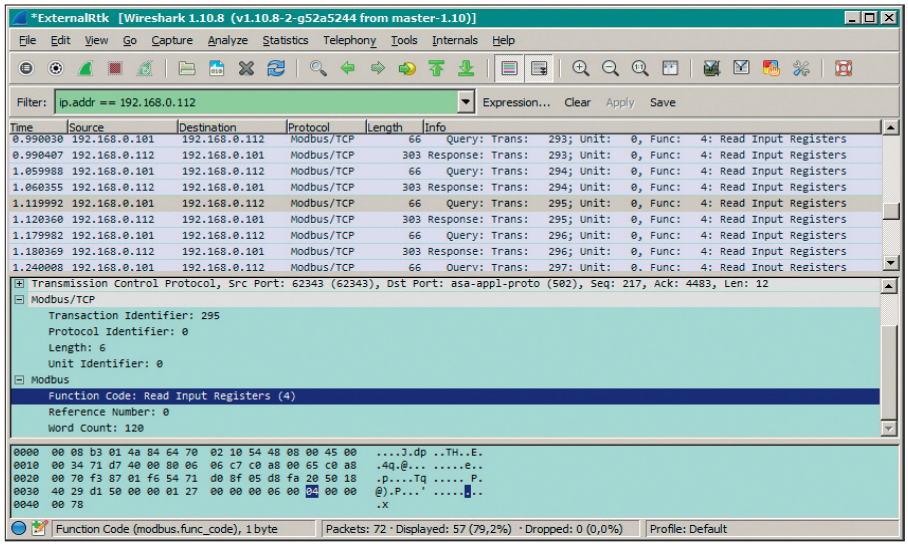

Рис. 6. Окно утилиты Wireshark при измерении времени обмена с CPM713

сети имеются два сервера, обмен данными с каждым из которых состоит из двух запросов чтения и одного запроса записи, причём запросы записи используются для формирования команд однопозиционного управления. Каждая команда состоит из записи клиентом MODBUS логической единицы в некоторое битовое поле типа *Coil* удалённого сервера, задержки длительностью 2 с и последующей записи логического нуля в это же битовое поле. Пусть в конфигурации клиента для каждого сервера задан тайм-аут ответа, равный 1 с. Предположим, что один из серверов по какой-то причине стал недоступным, скажем, из-за временного вывода управляемого им агрегата из эксплуатации, а клиент в это время по нажатию кнопки на экранной форме автоматизированного рабочего места оператора начинает выполнять команду включения агрегата, управляемого каналами дискретного вывода на другом сервере, записав логическую 1 в соответствующее поле типа *Coil*. Далее клиент пытается передать три запроса теперь уже недоступному серверу, и каждый из запросов через 1 с завершается неудачей по тайм-ауту, что в сумме составляет 3 с и превышает 2 с, по истечении которых требовалось передать логический 0 серверу, на котором начато выполнение команды. Последовательность дальнейших событий может, как минимум, закончиться выводом сообщения об ошибке выполнения команды для оператора.

Таким образом, при конфигурировании клиента MODBUS RTU необходимо устанавливать минимально возможные значения тайм-аута ответа для серверов, а также учитывать их значения при оценке времени обмена данными по сети. Если для клиента MODBUS неизвестен или слабо документирован механизм определения тайм-аута ответа, минимальное значение тайм-аута T_{TOUT} для каждого сервера можно оценить, воспользовавшись следующим приближённым соотношением:

$$T_{TOUT} = T_{SDR} + T_{RES} + T_{3.5}$$

Здесь T_{SDR} – задержка ответа сервера, T_{RES} – время самого длинного ответа сервера при выбранной скорости обмена, $T_{3.5}$ – длительность 3,5 символов тишины при выбранной скорости обмена.

Например, для сервера MODBUS RTU контроллера CPM712 можно определить ряд минимальных значений тайм-аута ответа, воспользовавшись данным приближённым соотношением и расчётными данными таблицы 2: для скорости 19200 бит/с – 170 мс; для 38400 бит/с – 90 мс; для 57600 бит/с – 70 мс; для 115200 бит/с – 40 мс. Итоговые оценки для верности округлены до ближайшего десятка миллисекунд сверху.

Приведённые рассуждения об оценке тайм-аута ответа относятся только к сетям MODBUS RTU, поскольку в режиме ASCII селекция пакетов запроса и ответа производится по специальным символам, обрамляющим пакет, а в протоколе MODBUS TCP между клиентом и каждым сервером устанавливается прямое TCP-соединение.

Обратите внимание, что тайм-аут ответа, который может быть задан пользователем в конфигурации клиента (мастера) MODBUS контроллера CPM712 в окне **PLC Configuration**, показанном на рис. 7, для каждого опрашиваемого сервера (подчинённого узла) представляет собой ожидаемое максимальное значение задержки ответа сервера T_{SDR}, то есть интервал времени между передачей клиентом последнего символа запроса до получения первого символа ответа от сервера.

Команды управления и параметризация приложений на подчинённых узлах

Упомянутые особенности протокола MODBUS ставят под сомнение правильность реализации сетевых команд управления путём обычной записи значений в *Holding*-регистры или битовые поля типа *Coil*. Такой способ реализации команд основан на предположении, что переменные удалённого сервера, отображённые на *Holding*-регистры, доступны для записи и чтения в любой момент времени. Однако даже кратковременные отказы сети, в том числе непосредственно не связанные с сервером, на котором требуется выполнить команду, могут привести к нарушению алгоритма её выполнения. Именно поэтому в более развитых промышленных сетевых протоколах, таких как DNP3, команды аналогового и дискретного управления являются объектами прикладного уровня протокола: Analog Output Block (блок аналогового управления, или AOB) и Control Relay Output Block (блок дискретного управления, или CROB). И по этой же причине пользователям рекомендуется реализовывать команды управления по сети MODBUS способом, чуть более сложным, чем просто запись клиентом значения в *Holding*-регистр или битовое поле типа *Coil* удалённого сервера.

Команда может рассматриваться как функция, вызываемая клиентом на удалённом сервере и имеющая тип (номер), чтобы можно было отличать одну команду от другой, набор входных аргументов, передаваемых функции при вызове, и возвращаемое значение. Набор входных аргументов и возвращаемое значение дополняются служебным полем, содержащим, как минимум, счётчик вызова. При запуске команды на удалённом сервере клиент записывает номер функции, значения входных аргументов и увеличенное на единицу значение счётчика вызова данной функции, получаемое у сервера, на котором выполняются команды. Команда считается завершённой, а результат готовым для чтения клиентом, как только значение счётчика вызова, получаемое клиентом от сервера, становится равным значению счётчика, переданному клиентом серверу при запуске команды.

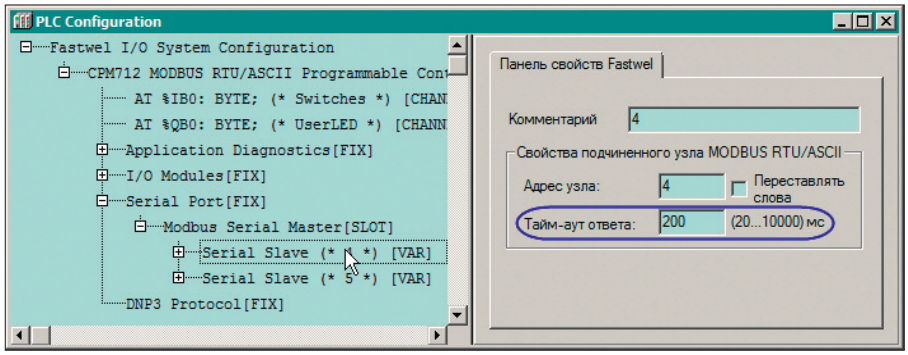

Рис. 7. Тайм-аут ответа подчинённого узла в конфигурации клиента MODBUS CPM712

При использовании протокола MODBUS номер функции и входные аргументы отображаются на один или несколько *Holding*-регистров удалённого сервера, а результат выполнения – на один или несколько *Input*-регистров. Помимо счётчика вызова, можно передавать между клиентом и сервером текущее состояние вызова, если предполагается имитировать объекты команд DNP3, подобные AOB или CROB. Следует отметить, что описанная идея применима к любым сетевым промышленным протоколам, поддерживающим циклический обмен данными по сети.

Ещё один вопрос, который время от времени возникает у пользователей контроллеров FASTWEL I/O с поддержкой протокола MODBUS: как передать в контроллер CPM712 или CPM713 от SCADA-системы значения уставок и других параметров алгоритма, которые в контроллере хранятся в энергонезависимых переменных, декларированных в секции VAR_GLOBAL RETAIN или VAR RETAIN?

Данный вопрос связан с тем, что на *Holding*-регистры сервера MODBUS контроллеров FASTWEL I/O могут быть отображены только переменные во входной части образа процесса, имеющие спецификаторы адреса *%I** и недоступные для записи из кода приложения.

Энергонезависимые переменные (далее *RETAIN*-переменные) контроллеров CPM71x в общем случае имеют три источника значений. Первым источником является область декларации переменных, в которой пользователь задаёт начальные значения переменных, как показано на рис. 8 .

Если начальные значения для переменных не заданы, им присваиваются исходные нулевые значения: 0 для числовых типов, FALSE – для типа *BOOL*, T#0ms – для типа *TIME* и т.д. Начальные значения используются при первом запуске загруженного приложения.

Вторым источником является энергонезависимая память, из которой при запуске контроллера считываются значения *RETAIN*-переменных, если до этого запуска они были сохранены в ней хотя бы один раз. Это связано с тем, что при включении питания контроллера перед запуском приложения нужно убедиться, что значения *RETAIN*-переменных не были повреждены, пока у контроллера было выключено питание.

Третьим источником значений *RETAIN*-переменных является код приложения во время исполнения, и при этом формировать значения энергонезависимых переменных могут прикладные алгоритмы, выполняющиеся под управлением разных циклических задач.

Если *RETAIN*-переменные сделать доступными через *Holding*-регистры, то поведение системы может стать слабо предсказуемым, поскольку одновременно с задачами приложения *RETAIN*-переменные могут быть изменены по сети, а в случае MODBUS TCP сделать это могут одновременно несколько разных клиентов MODBUS TCP.

В случае если все или часть *RETAIN*-переменных являются уставками и должны задаваться только по сети, то проблема становится менее острой, особенно в случае если известно, что записывать новые значения может только один MODBUS-клиент. Но при этом остаётся вопрос проверки корректности значений уставок, поступивших по сети.

1. А правильные ли по смыслу значения записаны?
2. А правилен ли момент записи, и нет ли каких-либо внутренних для приложения условий, ограничивающих запись именно сейчас?
3. А имел ли право этот сетевой клиент менять значения?

Для сети MODBUS на все три вопроса ответ можно дать только способом, специфическим для приложения, причём только в коде самого приложения. Но сделать это нужно до того, как полученные значения попали в энергонезависимую память. Поэтому даже если часть *RETAIN*-переменных всегда является уставками, значения которых могут быть изменены только по сети, делать любые *RETAIN*-переменные доступными через *Holding*-регистры в многозадачной/многоклиентской системе исполнения крайне опасно, так как некоторые логические ошибки не смогут быть обнаружены даже при длительной отладке и продумывании. Проблема в том, что при использовании протокола MODBUS и MODBUS TCP неизвестно, в каких регистрах передаются уставки, а в каких данные реального времени, то есть уставка является прикладным понятием, выходящим за рамки модели данных MODBUS. В то же время система исполнения контроллера не может самостоятельно определить, какие *RETAIN*-переменные являются уставками, а какие архивными данными, формируемыми самим алгоритмом, и, наконец, какой источник данных является доминирующим для тех или иных *RETAIN*-переменных: сетевые клиенты или сам алгоритм. При разработке максимально безопасной системы исполнения контроллера на этот счёт не могут делаться какие-либо неявные предположения, которые невозможно формально проверить при создании или исполнении пользовательского приложения. И, вообще говоря, сказанное можно отнести не только к *RETAIN*-переменным, но и к любым глобальным переменным с произвольным доступом по чтению и записи.

В контроллерах других производителей с однозадачной системой исполнения, в которой сетевой ввод-вывод всегда отделён от исполнения пользовательского кода по времени, вопрос синхронизации множества источников значений переменных с исполняемым кодом алгоритма решается автоматически, но за это приходится платить тем, что период цикла пользовательского алгоритма и время реакции на сетевые запросы невозможно предсказать заранее. Кроме того, при этом высока вероятность по-

```
TYPE AppSetPoints :
  STRUCT
    spRealValue : REAL;
    spDwordValue : DWORD;
  END_STRUCT
END_TYPE

VAR_GLOBAL RETAIN
(* начальные значения при первом запуске *)
  stSetpoints : AppSetPoints :=
  (
    spRealValue := 1.23,
    spDWordValue := 12345
  );
END_VAR
```

Рис. 8. Декларация *RETAIN*-переменных с начальными значениями

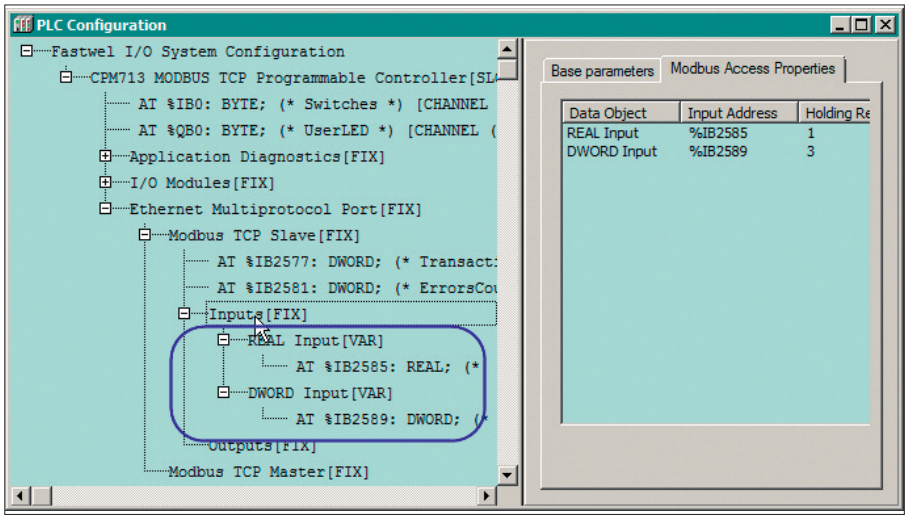

Рис. 9. Конфигурация MODBUS-сервера для приёма значений уставок, представленных типом *AppSetPoints*

```
VAR_GLOBAL
 (* Структурная переменная, используемая для получения *)
 (* уставок по MODBUS *)
   mbSetpoints AT %IB2585 : AppSetPoints;
END_VAR
```

Рис. 10. Структурная переменная для получения значений уставок по MODBUS

```
(* Аналог FwMemCopy из FastwelUtils.lib.                              *)
(* Главное отличие - позволяет копировать куда угодно откуда угодно,  *)
(* тогда как FwMemCopy копирует только в пределах сегмента глобальных *)
(* данных.                                                            *)
(* ВНИМАНИЕ! Перекрытие диапазонов адресов не проверяется             *)
FUNCTION MemCopyDirect : INT

VAR_INPUT
  pDestination : POINTER TO BYTE;
  pSource : POINTER TO BYTE;
  size : INT;
END_VAR

VAR_TEMP
  pDst : POINTER TO BYTE;
  pSrc : POINTER TO BYTE;
  idx : INT;
END_VAR

(* @END_DECLARATION := '0' *)
MemCopyDirect := 0;

IF 0 <> pDestination AND 0 <> pSource AND 0 <> size THEN
  pDst := pDestination;
  pSrc := pSource;
  FOR idx := 1 TO size DO
    pDst^ := pSrc^;
    pDst := pDst + SIZEOF(pDst^);
    pSrc := pSrc + SIZEOF(pSrc^);
  END_FOR
  MemCopyDirect := size;
END_IF

END_FUNCTION
```

Рис. 11. Функция копирования *size* байт из области памяти *pSource* в *pDestination*

явления трудноуловимых ошибок во время работы системы. Представим, к примеру, что значение булевой переменной *bOnOff* приложения, исполняемого в контроллере, определяется результатом записи в некоторое битовое поле типа *Coil* со стороны клиента MODBUS. Пусть приложение, обнаружив «передний фронт» значения *bOnOff*, формирует импульсную команду включения или выключения коммутационного аппарата, подключённого к каналу дискретного вывода контроллера, после чего самостоятельно сбрасывает в FALSE значение *bOnOff*. Это нормально работает до момента, когда клиент MODBUS, передав в *bOnOff* значение TRUE, пропускает ответ контроллера на данный запрос записи из-за помехи в линии связи. Контроллер же, обнаружив передний фронт *bOnOff*, выдаёт команду управления и самостоятельно сбрасывает значение *bOnOff*. Если клиент MODBUS, не получив ответа на последний запрос записи, повторно передаёт TRUE в *bOnOff*, это приводит к повторной выдаче команды на коммутационный аппарат. Либо приложение, взаимодействующее с контроллером по MODBUS, включает «красный транспарант», предупреждая оператора о неудачном выполнении последней команды.

В системе исполнения контроллеров FASTWEL I/O *Holding*-регистры для алгоритма являются строго входными (*%I**), что означает единственность источника значений отображённых на них переменных, а среда разработки CoDeSys 2.3 размещает переменные типа *%I** и *RETAIN*-переменные в разных непересекающихся сегментах памяти, что делает невозможным объявлять *%I**-переменные в качестве *RETAIN*. Чтение *Holding*-регистров клиентами MODBUS используется только для того, чтобы узнать, какие значения были ранее записаны в некоторую область *Holding*-регистров по сети. Такая модель повышает предсказуемость поведения системы, как с точки зрения выполнимости временны́х ограничений, так и с точки зрения предотвращения логических ошибок, которые могут возникнуть при наличии возможности менять значения любых переменных из разных источников данных.

Теперь о том, как решить обозначенную проблему сетевой параметризации контроллеров FASTWEL I/O. Идея состоит в том, что перед запуском приложения нужно вручную, используя арифметику указателей, скопировать значения требуемых *RETAIN*-переменных в соответствующие входные переменные, отображённые на *Holding*-регистры MODBUS. Системное событие OnInit является единственным местом, где это удастся сделать, поскольку после него пользовательский код имеет дело с адресами *%I**-переменных, размещённых в персональных сегментах каждой циклической задачи, о чём было подробно рассказано в первой части статьи, опубликованной в «СТА» 3/2014.

Пусть уставки декларированы в виде энергонезависимой переменной, как показано на рис. 8. Тогда в конфигурации MODBUS-сервера следует создать необходимое число входных переменных для приёма значений уставок по сети, используя те же типы данных и в том же порядке, как они перечислены в структурном типе AppSetPoints, как показано на рис. 9.

Затем нужно декларировать переменную, отображённую на начальный адрес группы переменных в конфигурации MODBUS-сервера, как показано на рис. 10.

Далее следует создать функцию копирования данных по произвольному адресу, код которой показан на рис. 11 .

Теперь необходимо установить обработчик системного события *OnInit*, которое происходит всякий раз при запуске контроллера при включении пита-

ния или после загрузки приложения в контроллер, но до запуска самого приложения. Рекомендуемый в документации способ установки обработчика системного события иллюстрирует рис. 12. В рассматриваемом примере можно выполнить все необходимые действия непосредственно в функции *SysEventDispatcher*, но если потребуется сделать что-то большее, чем просто копирование, могут понадобиться временные переменные внутри *SysEventDispatcher*, а их в ней декларировать нельзя из-за особого соглашения о вызовах функций обработки системных событий. Поэтому из функции *SysEventDispatcher* лучше вызвать специфический обработчик события *OnInitHandler*, как показано на рис. 12. Функция *OnInitHandler* копирует содержимое уставок из *RETAIN*-переменной *stSetpoints*, расположенной в сегменте энергонезависимых переменных, в переменную *mbSetpoints*, отображенную на *Holding*-регистры. Более изящный способ обработки события состоит в использовании вместо отдельной функции *OnInitHandler* специального *действия* (Action), которым дополняется программа, содержащая уставки и/или управляющая значениями уставок. В CoDeSys 2.3 *действия* во многом аналогичны методам классов в объектно-ориентированных языках программирования и позволяют оперировать внутренними переменными программных единиц типа *программа* (PROGRAM) или *функциональный блок* (FUNCTIONAL BLOCK) из других программных единиц.

Наконец, в приложение нужно добавить код обновления уставок в *RETAIN*-переменных из *Holding*-регистров во время исполнения, как показано на рис. 13. В данном случае программа *PLC_PRG* первой же инструкцией копирует значения уставок, полученных по MODBUS, в *RETAIN*-переменную *stSetpoints*. В приложении для реальной «боевой» системы копирование можно делать только после проверки корректности значений, полученных по MODBUS, а результат проверки может быть передан SCADA-системе через отдельный *Input*-регистр.

Реализация сервера MODBUS в коде приложения

Описание особенностей применения встроенного сервера протокола MODBUS контроллеров FASTWEL I/O будет неполным, если не упомянуть о системной библиотеке FastwelModbusServer.lib. Данная библиотека предназначена для конфигурирования и запуска сервера MODBUS, входящего в состав системного программного обеспечения всех контроллеров FASTWEL I/O, из кода приложения CoDeSys 2.3, загруженного в контроллер. По существу FastwelModbusServer.lib позволяет превратить любой контроллер FASTWEL I/O в подчинённый узел сети MODBUS RTU или ASCII, даже если у контроллера нет лишних коммуникационных портов. Это стало возможным благодаря существенному улучшению интеграции интерфейсных модулей NIM741 и NIM742 в систему исполнения контроллеров FASTWEL I/O: начиная с лета 2013 г., данные модули могут использоваться как полноправные коммуникационные порты.

Изначально FastwelModbusServer.lib была предназначена только для работы с сервисным портом контроллеров FASTWEL I/O, расположенным на передней панели под пластиковой крышкой, а для контроллера CPM712 – и со штатным портом интерфейса RS-485, если в конфигурации приложения для него установлена опция *Not Used* (не используется). Однако, как только порты на базе модулей NIM741/ NIM742 стали равноправными со встроенными коммуникационными портами, функции библиотеки FastwelModbusServer.lib автоматически распространились и на данные модули, что открыло для пользователей новые возможности.

Во-первых, если в системе в качестве основной магистральной сети, объединяющей множество территориально распределённых объектов, должен использоваться MODBUS, необязательно приобретать контроллер со встроенным интерфейсом подчинённого узла MODBUS, такой как CPM702 или CPM712. В таком случае можно применить контроллер любого типа, в том числе с интерфейсом Ethernet или PROFIBUS DP, а сервер MODBUS реализовать на базе библиотеки FastwelModbusServer.lib. Основной же сетевой интерфейс контроллера может служить для организации «быстрой» локальной сети на самом объекте, как в случае контроллера CPM713, который одновременно является и клиентом (мастером) и сервером (подчинённым) MODBUS TCP. Некоторые из возможных ролей конт-

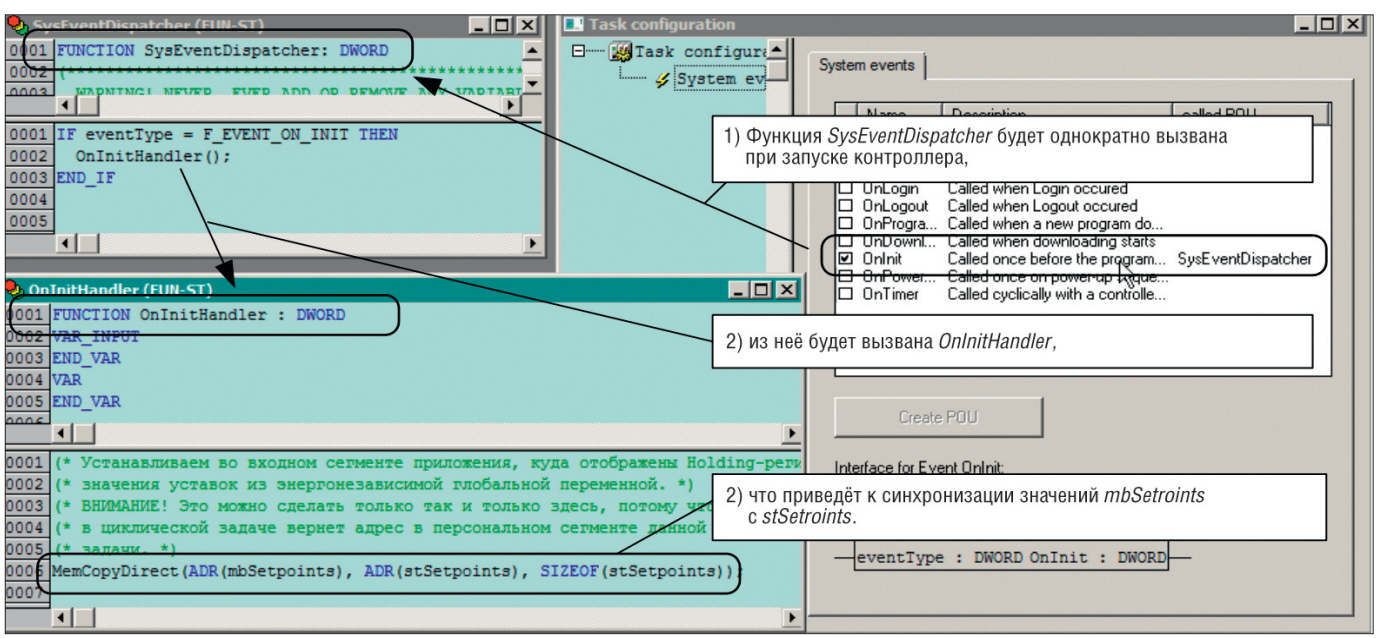

Рис. 12. Установка обработчика системного события *OnInit* для записи начального или ранее сохранённого значения *RETAIN*-переменной в переменную, отображённую на *Holding*-регистры MODBUS

роллера CPM713 в сетевом взаимодействии показаны на рис. 14.

Во-вторых, дополнительный сервер MODBUS можно использовать для обмена данными между контроллером и панелями локальной визуализации, такими как поставляемые фирмой Weintek, которые обычно размещаются вблизи контроллера.

Таким образом, библиотека FastwelModbusServer.lib позволяет создавать весьма экономичные решения, причём не только за счёт разницы в стоимости аппаратных средств и программного обеспечения по сравнению с продукцией конкурентов, имеющей аналогичные функциональные возможности, но и благодаря унификации используемого оборудования и программного обеспечения.

Для реализации сервера MODBUS с помощью библиотеки FastwelModbusServer.lib нужно включить данную библиотеку в ресурс **Library Manager** (Менеджер библиотек) проекта CoDeSys 2.3.

Если обмен данными с сервером должен осуществляться через сервисный порт на передней панели контроллера, необходимо включить переключатель

```
PROGRAM PLC_PRG

VAR
(* просто счетчик циклов *)
  dwCounter : DWORD;
END_VAR

(* @END_DECLARATION := '0' *)

(* Копируем значения уставок, полученных по MODBUS, в энергонезависимую *)
(* глобальную переменную. *)
(* В боевом проекте необходимо делать валидацию значений! *)
  stSetpoints := mbSetpoints;

  dwCounter := dwCounter + 1;

END_PROGRAM
```

Рис. 13. Обновление уставок в *RETAIN*-переменной *stSetpoints* во время исполнения приложения

4-го блока переключателей контроллера. Если же требуется использовать коммуникационный порт на базе модуля NIM741 или NIM742, следует подключить данный модуль к внутренней шине контроллера, а затем в ресурсе **PLC Configuration** проекта CoDeSys 2.3 добавить элемент *NIM741 RS-485 1xUART Stream Module* или *NIM742 RS-232 1xUART Stream Module* в конфигурацию межмодульной шины. Добавлять или вставлять данный элемент нужно в позицию списка модулей ввода-вывода, соответствующую порядковому номеру реального модуля на шине, исключая все пассивные модули. Для удобства идентификации описаний модулей можно щёлкнуть на корневом элементе списка модулей (*I/O Modules*) в дереве конфигурации и нажать кнопку *Обновить номера*, что приведёт к появлению номеров модулей справа от их названий, а для элементов *NIM741 RS-485 1xUART Stream Module* и *NIM742 RS-232 1xUART Stream Module* справа от названий появятся полные идентификаторы COM101...COM164, числовую составляющую которых нужно использовать при открытии соответствующих коммуникационных портов.

Библиотека FastwelModbusServer.lib имеет единственную функцию *FwModbusServerInit*, которая должна быть вызвана один раз из обработчика систем-

ного события *OnInit*. Функция принимает четыре аргумента. Первый аргумент должен содержать указатель на переменную типа *F_MODBUS_SERVER_SETTINGS*, которая определяет номер коммуникационного порта, адрес узла в сети MODBUS и параметры обмена. Следующими тремя аргументами функции передаются три указателя на переменные типа *F_VAR_DESCRIPTOR*, описывающие области чтения, записи и диагностики сервера MODBUS. Декларация структуры *F_VAR_DESCRIPTOR* показана на рис. 15.

Поля *Address* и *Size* должны содержать адрес и размер переменной, предпочтительно структуры или массива, которую предполагается читать или изменять по MODBUS. В поле *PouIndex* должен быть передан индекс программной единицы (программы или экземпляра функционального блока), в которой объявлена отображаемая переменная, чтобы обеспечить координированную работу сервера MODBUS и задачи, из которой осуществляется доступ к отображаемой переменной. Индекс программной единицы может быть получен оператором *INDEXOF*, которому в качестве параметра передаётся имя программной единицы, а адрес и размер переменной, отображаемой на MODBUS, получаются операторами *ADR* и *SIZEOF* соответственно. На рис. 16 приведён пример *действия* (Action), вызываемого при обработке системного события *OnInit* для инициализации сервера MODBUS через библиотеку FastwelModbusServer.lib. В данном примере переменные *MbSrvInput*, *MbSrvOutput* и *MbSrvDiag*, отображаемые на MODBUS, принадлежат программе *MBSRV1*, а переменная *mbNodeSettings*, посредством которой устанавливаются параметры обмена, может быть глобальной константой.

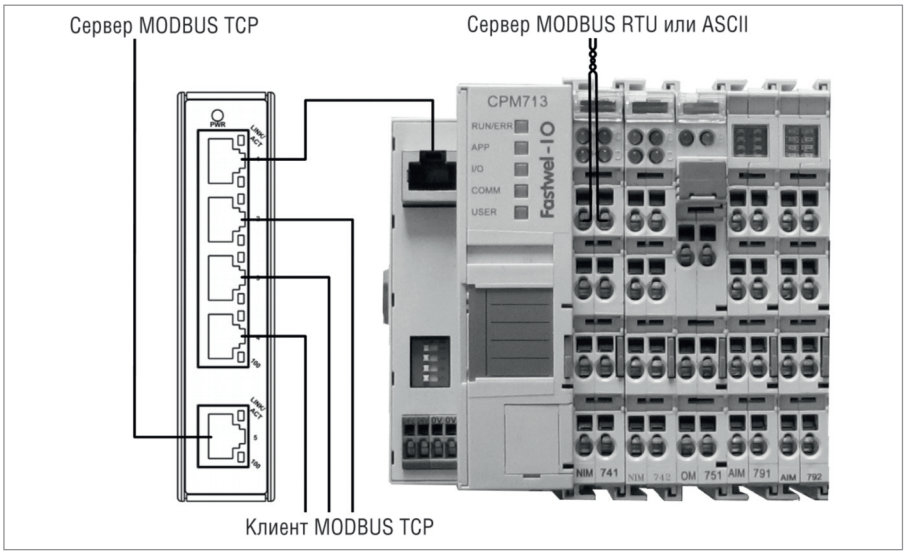

Рис. 14. Возможные роли контроллера CPM713 в сетевом взаимодействии по протоколу MODBUS

Если сервер инициализирован успешно, функция *FwModbusServerInit* возвращает нулевое значение. В дальнейшем от приложения не требуется какого-либо участия в обмене данными с клиентом по сети. Переменная *MbSrvDiag*, адрес и размер которой

```
TYPE F_VAR_DESCRIPTOR :
  STRUCT
   (* Указатель на область памяти, отображаемой на MODBUS. *)
    Address : DWORD;
   (* Размер отображаемой области памяти в байтах. *)
    Size : INT;
   (* Индекс программной единицы (POU), содержащей описываемую переменную. *)
    PouIndex : INT;
  END_STRUCT
END_TYPE
```

Рис. 15. Структурный тип *F_VAR_DESCRIPTOR*

переданы в последнем параметре при вызове *FwModbusServerInit* в рассматриваемом примере, имеет тип *F_MODBUS_SERVER_DIAGNOSTICS*, который определён в библиотеке FastwelModbusServer.lib и позволяет приложению следить за текущим состоянием сервера, количеством успешных операций обмена по сети, количеством низкоуровневых и высокоуровневых ошибок (исключений), которые могут возникнуть при поступлении запросов к несуществующим объектам со стороны клиента MODBUS.

Область памяти, определяемая вторым аргументом *FwModbusServerInit*, отображается на набор регистров и битовых полей, доступных для чтения и записи по сети (*Holding Register* и *Coil*), а область, описываемая третьим аргументом, – на регистры и битовые поля, доступные только для чтения по сети (*Input Register* и *Discrete Input*). Регистры и битовые поля нумеруются и адресуются по тем же правилам, описание которых было дано для плоской модели отображения. При использовании библиотеки FastwelModbusServer.lib пользователь имеет дело с той же реализацией сервиса MODBUS, которая функционирует в контроллерах CPM712, CPM713 и MK905-01,03\CDS в качестве основного сервиса внешней сети. Разница состоит только в способе конфигурирования сервиса MODBUS. Конфигурация сервиса, создаваемая пользователем в окне ресурса **PLC Configuration**, передаётся сервису MODBUS без участия приложения, разрабатываемого пользователем, средствами системного программного обеспечения контроллера. Библиотека FastwelModbusServer.lib позволяет сконфигурировать сервис MODBUS из приложения.

Во время работы приложения перед каждым циклом задачи, из которой вызывается программная единица с индексом, переданным во втором параметре функции *FwModbusServerInit*, происходит копирование данных из внутреннего буфера *Holding*-регистров сервера MODBUS в переменную, адрес и размер которой были переданы во втором параметре вместе с номером программной единицы. Таким образом, изменения, сделанные удалённым клиентом MODBUS по сети, никогда не могут быть восприняты приложением посреди цикла задачи, оперирующей переменной, которая отображена на *Holding*-регистры и битовые поля типа *Coil*. В конце цикла задачи, из которой вызывается программная единица с индексом, переданным функции *FwModbusServerInit* в третьем параметре, происходит копирование данных из переменной, отображённой на *Input*-регистры и битовые поля типа *Discrete Input*, во внутренний буфер *Input*-регистров сервера MODBUS. Таким образом, клиент MODBUS никогда не сможет прочитать частично сформированное значение переменной посреди цикла задачи, под управлением которой формируется её значение. Если запрос чтения поступает в такой неподходящий момент, ответ на него формируется без лишних задержек из внутреннего буфера *Input*-регистров сервера MODBUS.

Подробная информация о применении библиотеки FastwelModbusServer.lib приведена в документации, а также в примерах программирования, поставляемых в пакете адаптации CoDeSys 2.3 для FASTWEL I/O.

Мастер MODBUS и MODBUS TCP в контроллерах CPM712 и CPM713

В завершение рассказа о функциональных возможностях контроллеров FASTWEL I/O, связанных с протоколом MODBUS, стоит упомянуть о реализации клиента (мастера) MODBUS и MODBUS TCP в контроллерах CPM712, CPM713 и MK905-01,03\CDS.

Для включения клиента MODBUS RTU или ASCII на контроллере CPM712 нужно в окне ресурса **PLC Configuration** для элемента дерева *Serial Port* выбрать опцию *Modbus Serial Master*, как показано на рис. 17. На контроллере CPM713 клиент MODBUS может функционировать одновременно с сервером, поэтому в дереве конфигурации всегда присутствует элемент *Ethernet Multiprotocol Port – Modbus TCP Master*.

Далее для CPM712 необходимо настроить параметры обмена, включая режим протокола (RTU или ASCII), скорость обмена, режим контроля по чётности и количество стоповых битов. Общим параметром для клиентов MODBUS и MODBUS TCP на контроллерах CPM712 и CPM713 является параметр *Гранулярность опроса* (мс), показанный на рис. 18.

Данный параметр определяет минимальный интервал времени между очередным и следующим запросами чтения и/или записи регистров и битовых полей, передаваемых в адрес удалённых серверов MODBUS. Конфигурация кли-

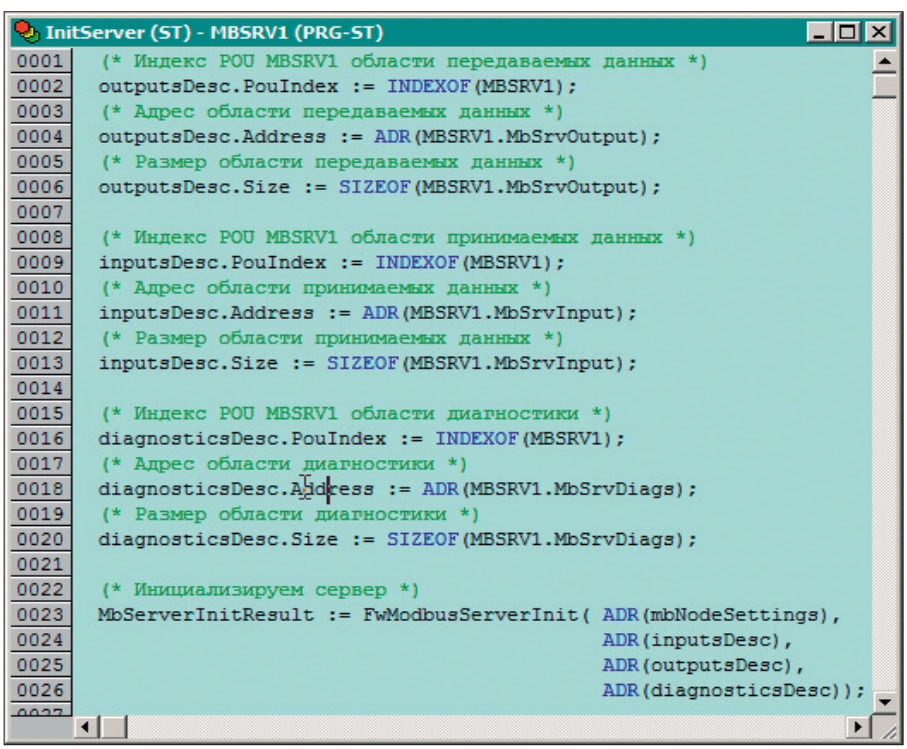

Рис. 16. Инициализация сервера MODBUS в приложении

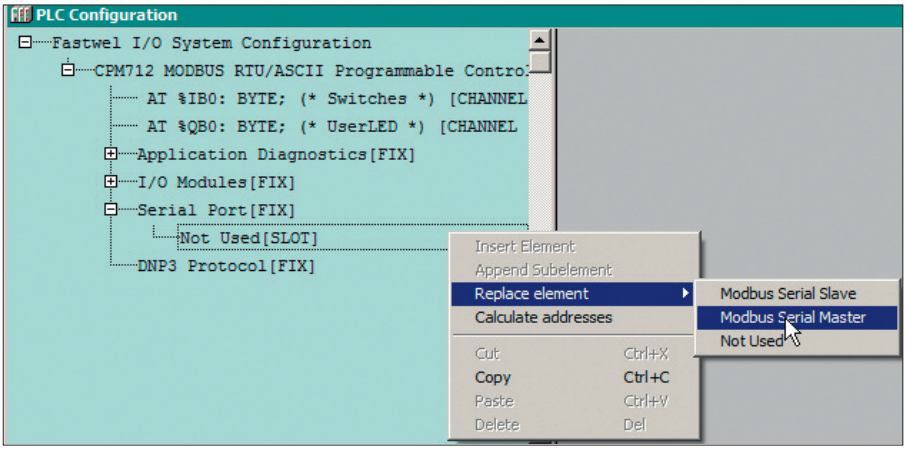

Рис. 17. Активизация клиента MODBUS в приложении для контроллера CPM712

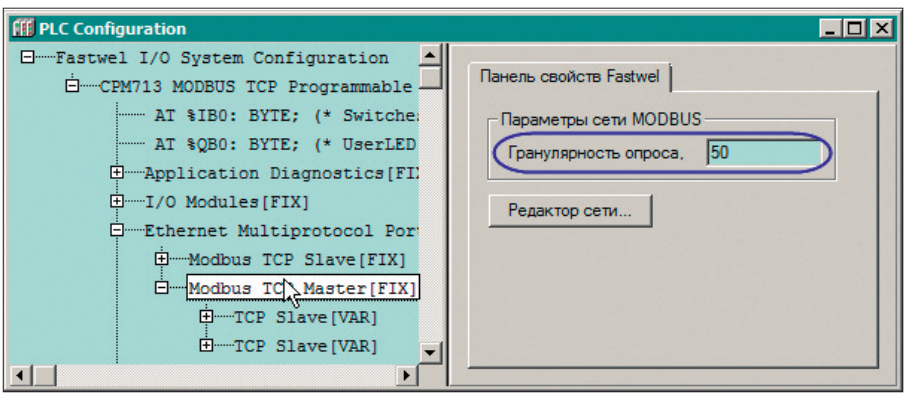

Рис. 18. Гранулярность опроса подчинённых узлов MODBUS

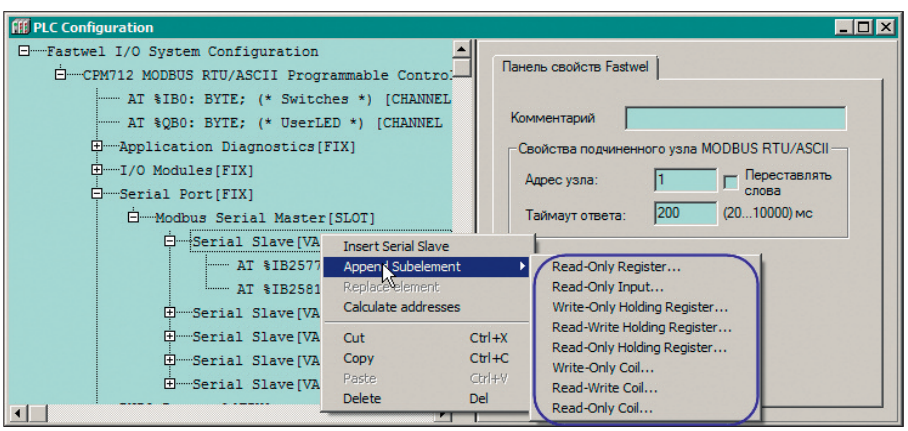

Рис. 19. Добавление описания коммуникационного объекта в конфигурацию сервера MODBUS

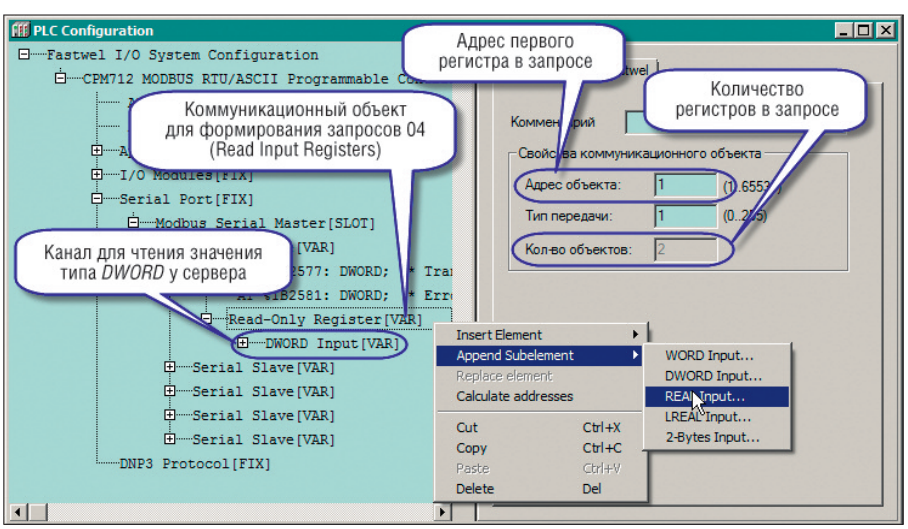

Рис. 20. Добавление каналов в описание коммуникационного объекта

ента MODBUS и MODBUS TCP по существу определяет расписание обмена по сети с удалёнными серверами, то есть множество запросов чтения и записи, которые клиент будет передавать серверам, и частоту их передачи.

Для сбалансированной работы всех подсистем контроллеров, поддерживающих функциональность клиента MODBUS, минимальный интервал времени между двумя запросами удалённым серверам физически ограничен значением, чуть меньшим 20 мс, поэтому не имеет смысла задавать значение гранулярности менее 20 мс.

Затем в конфигурацию клиента MODBUS или MODBUS TCP нужно добавить элементы, описывающие подчинённые узлы, с которыми предстоит обмениваться данными по сети. Для протокола MODBUS TCP свойства каждого сервера содержат IP-адрес, номер порта (по умолчанию 502) и номер опрашиваемого узла на сервере ввиду того, что протокол MODBUS TCP поддерживает маршрутизацию запросов в подсеть MODBUS, подключённую к удалённому серверу. В конфигурации сервера MODBUS RTU или ASCII собственным параметром является только сетевой адрес подчинённого узла. Общими в конфигурации серверов MODBUS TCP и MODBUS RTU/ASCII являются два параметра: тайм-аут ответа и признак необходимости менять местами байты в словах запросов и ответов к серверу, если на сервере используется кодировка Big-Endian, когда старшие байты значений располагаются по младшим адресам памяти. Обратите внимание, что задаваемый тайм-аут ответа определяет максимальное значение задержки ответа удалённого сервера T_{SDR}, о которой говорилось ранее.

Далее в конфигурацию каждого сервера должны быть добавлены элементы, описывающие коммуникационные объекты MODBUS, на основании которых клиент MODBUS будет передавать запросы удалённым серверам. Добавление элемента, описывающего коммуникационный объект, иллюстрирует рис. 19.

Название каждого коммуникационного объекта состоит из префикса сетевого доступа: *Read-Only* (только чтение), *Write-Only* (только запись) и *Read-Write* (чтение и запись) и типа объекта: *Register* – регистр типа *Input Register*; *Input* – битовое поле типа *Discrete Input*, *Holding Register* – регистр типа *Holding Register*, и *Coil* – битовое поле типа *Coil*.

Для объектов *Read-Only Register* и *Read-Only Input* клиент MODBUS во время работы контроллера будет формировать запросы чтения 02 (Read Discrete Inputs) и 04 (Read Input Registers). Количество регистров и битовых полей, запрашиваемых при чтении, определяется суммарным размером входных каналов (в словах и битах), добавленных пользователем в конфигурацию коммуникационных объектов, как показано на рис. 20. Период передачи запросов чтения в миллисекундах определяется значением параметра *Тип передачи*, умноженным на значение параметра *Гранулярность опроса*, установленного в свойствах сети для элемента *Modbus Serial Master* (CPM712) или *Modbus TCP Master* (CPM713). Если установлено нулевое значение параметра *Тип передачи*, запросы к данному объекту исключаются из расписания.

Для объектов *Write-Only Coil* и *Write-Only Holding Register* клиент MODBUS-контроллера будет формировать запросы записи 15 (Write Multiple Coils) и 16 (Write Multiple Registers). Адрес первого регистра или битового поля в запросе определяется значением параметра *Адрес объекта*, а количество битовых полей или регистров в запросе – суммарным размером (в битах или словах) выходных каналов, добавленных в конфигурацию коммуникационного объекта. Если параметр *Тип передачи* имеет значение от 1 до 250, то запросы записи будут передаваться удалённому серверу циклически с периодом, равным произведению значения данного параметра на длительность гранулярности опроса сети. Если *Тип передачи* равен 255, запросы записи будут передаваться однократно при каждом изменении значения переменной приложения, отображённой на каналы данного коммуникационного объекта.

Для объектов с префиксом *Read-Write* будут формироваться запросы записи и чтения к соответствующим *Holding*-регистрам и битовым полям типа *Coil* удалённого сервера по тем же правилам, что описаны ранее. Основное отличие коммуникационных объектов с префиксом *Read-Write* состоит в том, что в их конфигурацию добавлены парные входные (*%I**) и выходные (*%Q**) каналы, позволяющие контролировать успешность записи значений переменных в *Holding*-регистры или битовые поля типа *Coil* удалённого сервера MODBUS.

Переменные, значения которых должны передаваться удалённым серверам MODBUS или приниматься от них по сети, отображаются на каналы коммуникационных объектов в конфигурации клиента MODBUS по общим правилам, принятым в среде разработки CoDeSys 2.3: либо путём декларации непосредственно в дереве ресурса **PLC Configuration**, либо путём отображения в секциях декларации переменных приложения при помощи директивы *AT%*.

В заключение хотелось бы отметить, что коммуникационные возможности контроллеров FASTWEL I/O, связанные с протоколом MODBUS, позволяют решать разнообразные задачи сбора данных и управления в системах промышленной автоматизации. При этом пользователям не приходится нести дополнительных затрат на приобретение оборудования и лицензий на программное обеспечение. Все функциональные возможности, описанные в данной статье, доступны для применения непосредственно «из коробки» после приобретения соответствующих контроллеров и интерфейсных модулей. ●

Автор – сотрудник
ЗАО «НПФ «ДОЛОМАНТ»
Телефон: (495) 234-0639
E-mail:
alexander.lokotkov@dolomant.ru