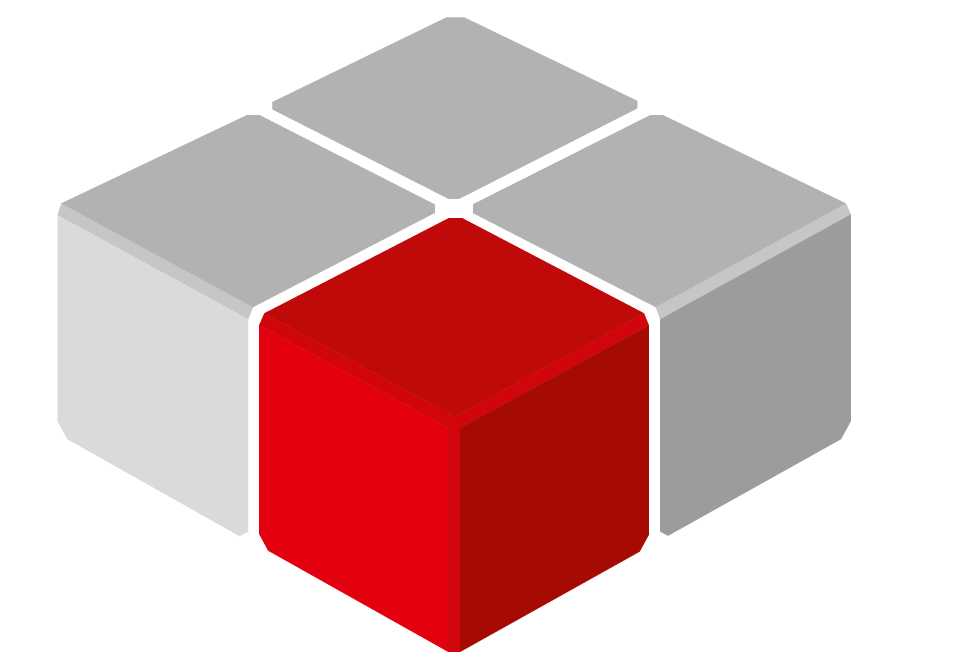


**CODESYS V3.5**

**Реализация нестандартных протоколов**



Руководство пользователя

25.02.2020

версия 2.1

Оглавление

[Глоссарий 2](#_Toc33519130)

[1 Цель и структура документа 2](#_Toc33519131)

[2 Основные сведения об обмене данными по последовательным интерфейсам 3](#_Toc33519132)

[2.1 Общие принципы организации обмена 3](#_Toc33519133)

[2.2 Работа с COM-портом 3](#_Toc33519134)

[2.3 Типы протоколов обмена 4](#_Toc33519135)

[2.4 Обработка ошибок обмена 5](#_Toc33519136)

[3 Библиотека CAA SerialCom 6](#_Toc33519137)

[3.1 Добавление библиотеки в проект CODESYS 6](#_Toc33519138)

[3.2 Структура COM.PARAMETER 7](#_Toc33519139)

[3.3 Cписок глобальных констант CAA\_Parameter\_Constants 7](#_Toc33519140)

[3.4 ФБ COM.Open 8](#_Toc33519141)

[3.5 ФБ COM.Write 9](#_Toc33519142)

[3.6 ФБ COM.Read 10](#_Toc33519143)

[3.7 ФБ COM.Close 11](#_Toc33519144)

[4 Пример опроса модуля МВ110-8А по протоколу DCON 12](#_Toc33519145)

[4.1 Формулировка задачи 12](#_Toc33519146)

[4.2 Описание протокола 12](#_Toc33519147)

[4.2 Алгоритмизация задачи 14](#_Toc33519148)

[4.3 ФБ управления портом (COM\_CONTROL) 14](#_Toc33519149)

[4.3.1 Инициализация порта (шаг INITIALIZE) 17](#_Toc33519150)

[4.3.2 Ожидание управляющего сигнала (шаг WAITING\_FOR\_SIGNAL) 17](#_Toc33519151)

[4.3.3 Открытие порта (шаг OPEN\_PORT) 18](#_Toc33519152)

[4.3.4 Закрытие порта (шаг CLOSE\_PORT) 21](#_Toc33519153)

[4.4 ФБ опроса модуля (MV110\_8A\_DCON) 23](#_Toc33519154)

[4.4.1 Подготовка запроса (шаг CREATE\_REQUEST) 25](#_Toc33519155)

[4.4.2 Отправка запроса (шаг SEND\_REQUEST) 29](#_Toc33519156)

[4.4.3 Получение ответа (шаг RECEIVE\_RESPONSE) 30](#_Toc33519157)

[4.4.4 Организация задержки (шаг RESPONSE\_DELAY) 34](#_Toc33519158)

[4.4.5 Завершение цикла опроса (шаг POLLING\_CYCLE\_ENDS) 35](#_Toc33519159)

[4.5 Программа опроса (PLC\_PRG) 37](#_Toc33519160)

[5 Пример опроса счетчика СЭТ-4ТМ.03М 39](#_Toc33519161)

[5.1 Формулировка задачи 39](#_Toc33519162)

[5.2 Описание протокола 40](#_Toc33519163)

[5.3 Алгоритмизация задачи 42](#_Toc33519164)

[5.4 ФБ управления портом (COM\_CONTROL) 42](#_Toc33519165)

[5.5 ФБ опроса счетчика (SET\_4TM) 43](#_Toc33519166)

[5.5.1 Подготовка запроса на открытие канала (шаг СREATE\_CHANNEL) 45](#_Toc33519167)

[5.5.2 Отправка запроса на открытие канала (шаг OPEN\_CHANNEL) 49](#_Toc33519168)

[5.5.3 Получение ответа на запрос открытия канала (шаг RECEIVE\_CHANNEL) 50](#_Toc33519169)

[5.5.4 Организация задержки (шаг RESPONSE\_DELAY\_CHANNEL) 52](#_Toc33519170)

[5.5.5 Подготовка запроса на чтение данных (шаг СREATE\_REQUEST) 52](#_Toc33519171)

[5.5.6 Отправка запроса на чтение данных (шаг SEND\_REQUEST) 53](#_Toc33519172)

[5.5.7 Получение ответа (шаг RECEIVE\_RESPONSE) 54](#_Toc33519173)

[5.5.8 Организация задержки (шаг RESPONSE\_DELAY) 57](#_Toc33519174)

[5.5.9 Завершение цикла опроса (шаг POLLING\_CYCLE\_ENDS) 57](#_Toc33519175)

[5.6 Программа опроса (PLC\_PRG) 59](#_Toc33519176)

[6 Рекомендации и замечания 61](#_Toc33519177)

[7 Библиотека OwenCommunication 62](#_Toc33519178)

[7.1 Общая информация 62](#_Toc33519179)

[7.2 ФБ COM\_Control 63](#_Toc33519180)

[7.3 ФБ UNM\_SerialRequest 64](#_Toc33519181)

[Приложение А. Листинг примера из п. 4 66](#_Toc33519182)

[А.1 ФБ COM\_CONTROL 67](#_Toc33519183)

[А.2 ФБ MV110\_8A\_DCON 69](#_Toc33519184)

[А.3 ФБ ANALYZE\_DATA 73](#_Toc33519185)

[А.4 Функция BYTE\_TO\_STRH 74](#_Toc33519186)

[А.5 Функция \_BUFFER\_CLEAR 74](#_Toc33519187)

[А.6 Перечисление COM\_STATE 75](#_Toc33519188)

[А.7 Перечисление DCON\_STATE 75](#_Toc33519189)

[Приложение Б. Листинг примера из п. 5 76](#_Toc33519190)

[Б.1 ФБ COM\_CONTROL 77](#_Toc33519191)

[Б.2 ФБ SET\_4TM 79](#_Toc33519192)

[Б.3 Функция CRC\_MG\_GEN 85](#_Toc33519193)

[Б.4 Функция \_BUFFER\_CLEAR 85](#_Toc33519194)

[Б.5 Перечисление COM\_STATE 86](#_Toc33519195)

[Б.6 Перечисление SET\_4TM\_STATE 86](#_Toc33519196)

# Глоссарий

**ПЛК** – программируемый логический контроллер.

**ФБ** – функциональный блок.

# Цель и структура документа

Одной из ключевых задач контроллеров является организация обмена данными с другими устройствами по последовательным интерфейсам RS-232/485.

В значительном количестве случаев обмен осуществляется по стандартным промышленным протоколам (например, **Modbus RTU**) с помощью средств конфигурирования **CODESYS** или библиотек ОВЕН, но иногда возникает необходимость организовать обмен с устройством, которое поддерживает только свой собственный специфичный протокол (в качестве примеров таких устройств можно привести различные тепло- и электросчетчики, весовые индикаторы, модули ввода-вывода и т. д.). В данном случае пользователю следует реализовать поддержку этого протокола в контроллере с помощью системных библиотек, которые позволяют работать с последовательным портом напрямую. Настоящее руководство описывает решение этой задачи для контроллеров ОВЕН, программируемых в **CODESYS V3.5**

Структура руководства:

В [п. 2](#_2._Основные_сведения_1) приведена основная информация о работе с последовательным портом и реализации протоколов обмена.

В [п. 3](#_3._Библиотека_CAA) приведено описание библиотеки **CAA SerialCom**, которая используется для работы с последовательным портом.

В [п. 4](#_4._Пример_опроса) рассмотрен пример опроса модуля **МВ110-8А** по протоколу **DCON**.

В [п. 5](#_5._Пример_опроса) рассмотрен пример опроса счетчика **СЭТ-4ТМ.03М**.

В [п. 6](#_6._Рекомендации_и) приведены рекомендации по реализации нестандартных протоколов обмена.

В [п. 7](#_Библиотека_OwenCommunication) приведена информация о библиотеке **OwenCommunication**.

В [приложении](#_Приложение) приведены листинги программ из примеров, рассмотренных в документе.

|  |  |
| --- | --- |
|  | **ПРИМЕЧАНИЕ**  Работа с нестандартными протоколами подразумевает высокую квалификацию программиста, хорошее знание среды **CODESYS V3.5** и языка ST. Документ рекомендуется читать строго последовательно. |

# Основные сведения об обмене данными по последовательным интерфейсам

## Общие принципы организации обмена

В рамках настоящего руководства рассмотрены протоколы, работающие поверх последовательного интерфейса RS-232/485 и основанные на архитектуре **Master – Slave** (ведущий – ведомый).

Данная архитектура подразумевает наличие в сети единственного master-устройства (обычно таким устройством является контроллер), которое последовательно опрашивает slave-устройства (ими могут быть модули ввода-вывода, панели оператора, частотные преобразователи и т. д.). Slave-устройство не может являться инициатором обмена (т. е. оно может только отвечать на полученные запросы).

Реализация нестандартного протокола обмена подразумевает решение двух смежных задач:

1. Организация работы с COM-портом (открытие порта, запись в порт, чтение из порта, закрытие порта).
2. Организация работы с данными (формирование запросов и анализ ответов согласно спецификации протокола).

В простейшем случае обмен происходит непрерывно: сразу после старта master-устройство начинает циклически опрашивать slave-устройства. Тогда в предельно упрощенном виде процедура опроса с точки зрения master-устройства может быть представлена следующим образом:

1. Открыть COM-порт с заданными настройками.
2. Отправить запрос первому slave-устройству.
3. Получить ответ от первого slave-устройства.
4. …
5. Отправить запрос последнему slave-устройству.
6. Получить ответ от последнего slave-устройства.
7. Перейти к шагу 2.

## Работа с COM-портом

Работа с COM-портом организуется с помощью одной из системных библиотек:

1. **SysCom** (создана на базе библиотеки **SysLibCom** из **CoDeSys 2.3**);
2. **CAA Serial** (разработана под **CODESYS V3**).

В рамках данного документа рассматривается библиотека [CAA Serial](#_3._Библиотека_CAA). Описание библиотеки приведено в [п. 3](#_3._Библиотека_CAA).

При работе с COM-портом выполняются следующие операции:

1. Открытие порта с заданными настройками. Обычно порт однократно открывается при старте контроллера и в дальнейшем не закрывается. Результатом успешного открытия порта является получение дескриптора (handle). Все последующие операции с портом производятся с указанием его дескриптора.
2. Запись в порт. Производится в случае необходимости отправить запрос slave-устройству.
3. Чтение из порта. Производится в случае необходимости получить ответ от slave-устройства.
4. Закрытие порта. В некоторых случаях при возникновении ошибок обмена на стороне master-устройства может потребоваться закрыть порт и открыть его заново.

## Типы протоколов обмена

C точки зрения представления передаваемых данных можно выделить два типа протоколов обмена:

1. Бинарные (двоичные) протоколы.

В этом случае каждый передаваемый байт может принимать любые значения (от 00 до FF). Представление каждого типа данных должно соответствовать какому-либо формату. Благодаря этому для каждого запроса заранее известен размер ответа в байтах.

Одним из известных стандартных бинарных протоколов является **Modbus RTU**.

Число с плавающей точкой **111.222** при передаче с помощью бинарного протокола согласно стандарту [IEEE 754](https://ru.wikipedia.org/wiki/%D0%A7%D0%B8%D1%81%D0%BB%D0%BE_%D0%BE%D0%B4%D0%B8%D0%BD%D0%B0%D1%80%D0%BD%D0%BE%D0%B9_%D1%82%D0%BE%D1%87%D0%BD%D0%BE%D1%81%D1%82%D0%B8) будет выглядеть так:

|  |  |  |  |
| --- | --- | --- | --- |
| **Байт 3** | **Байт 2** | **Байт 1** | **Байт 0** |
| 0x43 | 0xDE | 0x71 | 0xAA |

1. Строковые (текстовые) протоколы.

В данном случае каждый передаваемый байт содержит код символа из [таблицы ASCII](https://ru.wikipedia.org/wiki/ASCII). Размер ответа в байтах на один и тот же запрос может отличаться в зависимости от передаваемых значений – например, значение «5» займет один байт, а «55» – два.

Одним из известных стандартных строковых протоколов является **Modbus ASCII**.

Число с плавающей точкой **111.222** при передаче с помощью строкового протокола будет выглядеть так:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Байт 6** | **Байт 5** | **Байт 4** | **Байт 3** | **Байт 2** | **Байт 1** | **Байт 0** |
| 0x32 | 0x32 | 0x32 | 0x2E | 0x31 | 0x31 | 0x31 |
| «2» | «2» | «2» | «.» | «1» | «1» | «1» |

## Обработка ошибок обмена

В простейшем случае обмен происходит непрерывно: сразу после старта master-устройство начинает циклически опрашивать slave-устройства. В предельно упрощенном виде процедура опроса с точки зрения master-устройства может быть представлена следующим образом:

1. Открыть COM-порт с заданными настройками.
2. Сформировать запрос для первого slave-устройства.
3. Отправить запрос первому slave-устройству.
4. Получить ответ от первого slave-устройства.
5. Разобрать ответ от первого slave-устройства.
6. …
7. Сформировать запрос для последнего slave-устройства.
8. Отправить запрос последнему slave-устройству.
9. Получить ответ от последнего slave-устройства.
10. Разобрать ответ от последнего slave-устройства.
11. Перейти к шагу 2.

В данном случае подразумевается, что все slave-устройства всегда корректно отвечают на запросы. На практике могут возникать следующие ситуации:

1. Ответ не пришел (например, slave-устройство выключено или перестало работать).

Если в программе не предусмотрена обработка такой ситуации, то она будет продолжать ждать ответа, в результате чего обмен с другими устройствами прекратится. Поэтому обычно имеет смысл ожидать ответ в течение некоторого промежутка времени (который зависит от скорости обмена, количества передаваемых данных и индивидуальных особенностей устройства) и по его истечению переходить к опросу следующего регистра данного slave-устройства или следующего slave-устройства.

1. Ответ пришел в виде нескольких фрагментов (в большинстве случаев это происходит на низких скоростях обмена и при значительном количестве получаемых данных).

Соответственно, следует контролировать целостность ответа (в бинарных протоколов – по размеру ответа, в строковых – по наличию в ответе символов начала и конца). Если условие целостности не выполняется, то разбирать ответ не имеет смысла – необходимо сделать еще один запрос (сразу или в следующем цикле опроса).

1. Ответ был поврежден во время передачи (например, из-за действия помех).

Большинство протоколов используют контрольную сумму для проверки корректности ответа.

1. Некоторые slave-устройства в течение определенного интервала времени после ответа не могут принять запрос.

В таких случаях необходимо делать в программе паузу между получением ответа на текущий запрос и отправки следующего запроса.

# Библиотека CAA SerialCom

## Добавление библиотеки в проект CODESYS

Библиотека **CAA SerialCom** используется для работы с последовательным портом.

Для добавления библиотеки в проект **CODESYS** в **Менеджере библиотек** следует нажать кнопку **Добавить** и выбрать библиотеку **CAA SerialCom**, расположенную в папке **Intern/CAA/System**.

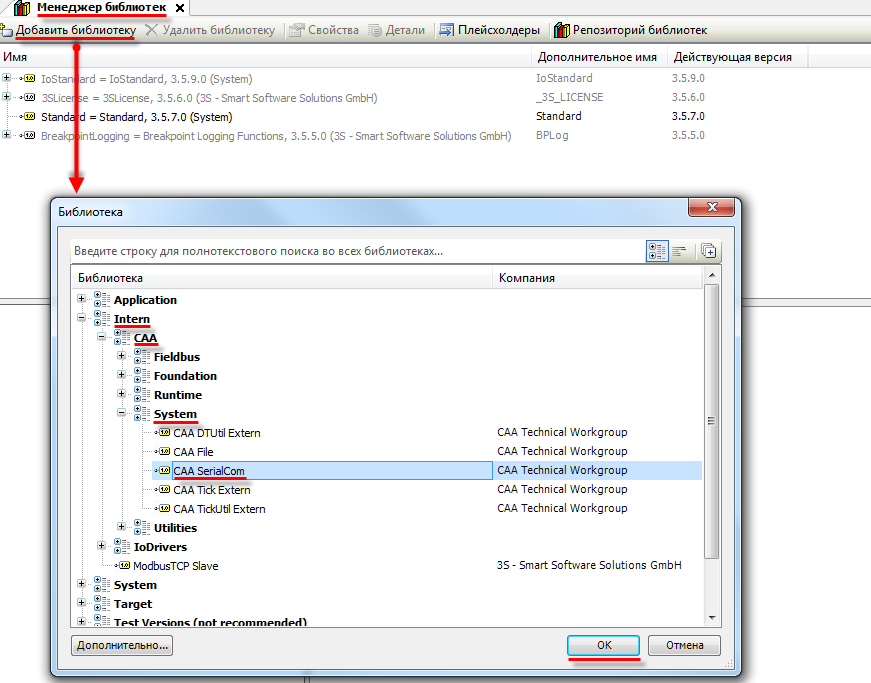


Рисунок 3.1 – Добавление библиотеки CAA SerialCom в проект CODESYS

При объявлении экземпляров ФБ библиотеки следует перед их названием указывать префикс «COM» (пример: **COM.**Open).

## Структура COM.PARAMETER

Структура **COM.PARAMETER** содержит имя параметра COM-порта и его значения. Для задания настроек COM-порта формируется массив структур типа **COM.PARAMETER**, указатель на который передается на вход блока [COM.Open](#_3.4._ФБ_COM.Open).

**Таблица 3.1 – Описание переменных структуры COM.PARAMETER**

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип данных** | **Описание** |
| udiParameterID | UDINT | ID или имя параметра COM-порта. Параметры определены в списке глобальных констант **CAA\_Parameter\_Constants** |
| udiValue | UDINT | Значение параметра |

## Cписок глобальных констант CAA\_Parameter\_Constants

Список глобальных констант **CAA\_Parameter\_Constant** содержит ID и имена параметров COM-порта, которые присваиваются переменным **udiParameterID** структуры **COM.PARAMETER**.

**Таблица 3.2 – Описание констант списка CAA\_Parameter\_Constant, поддерживаемых в контроллерах ОВЕН**

|  |  |  |  |
| --- | --- | --- | --- |
| **Название** | **ID** | **Тип данных** | **Описание** |
| ***Основные параметры*** | | | |
| udiPort | 16#1 | UDINT | Номер COM-порта. Номер COM-порта в CODESYS может не совпадать с номером на корпусе прибора. Подробнее см. в руководстве **CODESYS 3.5. FAQ** |
| udiStopBits | 16#2 | UDINT | Число стоп бит. Возможные значения определены в перечислении **COM.STOPBIT**:  **ONESTOPBIT** – один стоп бит;  **ONE5STOPBITS** – 1.5 стоп бита;  **TWOSTOPBITS** – 2 стоп бита |
| udiParity | 16#3 | UDINT | Режим контроля четности. Возможные значения определены в перечислении **COM.PARITY**:  **EVEN** – четный;  **ODD** – нечетный;  **NONE** – отсутствует |
| udiBaudrate | 16#4 | UDINT | Скорость обмена. Возможные значения: 1200/2400/4800/9600/19200/38400/57600/115200 |
| udiByteSize | 16#7 | UDINT | Количество информационных бит в передаваемых/принимаемых байтах |

## ФБ COM.Open

Функциональный блок **COM.Open** используется для открытия COM-порта с заданными настройками.

|  |  |
| --- | --- |
|  | **ПРИМЕЧАНИЕ**  Запрещается пытаться открыть уже открытый порт. |

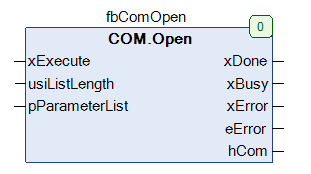


Рисунок 3.2 – Внешний вид ФБ COM.Open на языке CFC

**Таблица 3.3 – Описание входов и выходов ФБ COM.Open**

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип данных** | **Описание** |
| ***Входные переменные*** | | |
| **xExecute** | BOOL | Переменная активации блока. Запуск блока происходит по переднему фронту переменной |
| **usiListLength** | USINT | Количество используемых параметров порта |
| **pParameterList** | CAA.PVOID | Указатель на массив параметров порта ([COM.PARAMETER](#_3.2._Cтруктура_COM.PARAMETER)) |
| ***Выходные переменные*** | | |
| **xDone** | BOOL | Флаг успешного завершения работы блока |
| **xBusy** | BOOL | Флаг «ФБ в процессе работы» |
| **xError** | BOOL | Флаг ошибки. Принимает значение **TRUE** в случае возникновении ошибки |
| **eError** | COM.ERROR | Статус работы ФБ (или имя ошибки) |
| **hCom** | CAA.HANDLE | Дескриптор (handle) открытого порта |

## ФБ COM.Write

Функциональный блок **COM.Write** используется для записи данных в COM-порт.

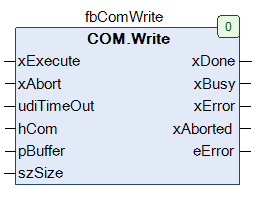


Рисунок 3.3 – Внешний вид ФБ COM.Write на языке CFC

**Таблица 3.4 – Описание входов и выходов ФБ COM.Write**

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип данных** | **Описание** |
| ***Входные переменные*** | | |
| **xExecute** | BOOL | Переменная активации блока. Запуск блока происходит по переднему фронту переменной |
| **xAbort** | BOOL | Переменная прекращения работы блока. Если она принимает значение **TRUE**, то блок немедленно прекращает работу, а выходные переменные сбрасываются к начальным значениям |
| **udiTimeOut** | UDINT | Время в мс, через которое блок принудительно завершает свою работу. При значении 0 эта функция отключается |
| **hCom** | CAA.HANDLE | Дескриптор (handle) порта, в который происходит запись |
| **pBuffer** | CAA.PVOID | Указатель на записываемые данные |
| **szSize** | CAA.SIZE | Размер записываемых данных (в байтах) |
| ***Выходные переменные*** | | |
| **xDone** | BOOL | Флаг успешного завершения работы блока |
| **xBusy** | BOOL | Флаг «ФБ в процессе работы» |
| **xError** | BOOL | Флаг ошибки. Принимает значение **TRUE** в случае возникновении ошибки |
| **xAborted** | BOOL | Флаг «работа ФБ была прервана» |
| **eError** | COM.ERROR | Статус работы ФБ (или имя ошибки) |

## ФБ COM.Read

Функциональный блок **COM.Read** используется для чтения данных из COM-порта.

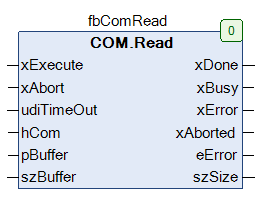


Рисунок 3.4 – Внешний вид ФБ COM.Read на языке CFC

**Таблица 3.5 – Описание входов и выходов ФБ COM.Read**

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип данных** | **Описание** |
| ***Входные переменные*** | | |
| **xExecute** | BOOL | Переменная активации блока. Запуск блока происходит по переднему фронту переменной |
| **xAbort** | BOOL | Переменная прекращения работы блока. Если она принимает значение **TRUE**, то блок немедленно прекращает работу и выходные переменные сбрасываются к начальным значениям |
| **udiTimeOut** | UDINT | Время в мс, через которое блок принудительно завершает свою работу. При значении 0 эта функция отключается |
| **hCom** | CAA.HANDLE | Дескриптор (handle) порта, из которого происходит чтение |
| **pBuffer** | CAA.PVOID | Указатель, по которому будут записаны считанные данные |
| **szBuffer** | CAA.SIZE | Размер считываемых данных (в байтах) |
| ***Выходные переменные*** | | |
| **xDone** | BOOL | Флаг успешного завершения работы блока |
| **xBusy** | BOOL | Флаг «ФБ в процессе работы» |
| **xError** | BOOL | Флаг ошибки. Принимает значение **TRUE** в случае возникновении ошибки |
| **xAborted** | BOOL | Флаг «работа ФБ была прервана» |
| **eError** | COM.ERROR | Статус работы ФБ (или имя ошибки) |
| **szSize** | CAA.SIZE | Размер считанных данных (в байтах) |

## ФБ COM.Close

Функциональный блок **COM.Close** используется для закрытия COM-порта.

|  |  |
| --- | --- |
|  | **ПРИМЕЧАНИЕ**  Запрещается пытаться закрыть уже закрытый порт. |

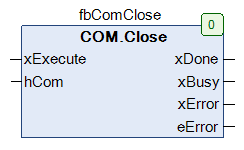


Рисунок 3.5 – Внешний вид ФБ COM.Close на языке CFC

**Таблица 3.6 – Описание входов и выходов ФБ COM.Close**

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип данных** | **Описание** |
| ***Входные переменные*** | | |
| **xExecute** | BOOL | Переменная активации блока. Запуск блока происходит по переднему фронту переменной |
| **hCom** | CAA.HANDLE | Дескриптор (handle) закрываемого порта |
| ***Выходные переменные*** | | |
| **xDone** | BOOL | Флаг успешного завершения работы блока |
| **xBusy** | BOOL | Флаг «ФБ в процессе работы» |
| **xError** | BOOL | Флаг ошибки. Принимает значение **TRUE** в случае возникновении ошибки |
| **eError** | COM.ERROR | Статус работы ФБ (или имя ошибки) |

# Пример опроса модуля МВ110-8А по протоколу DCON

## Формулировка задачи

В качестве примера будет рассмотрен опрос модуля МВ110-8А по протоколу **DCON** с помощью контроллера **СПК1хх [М01].** Модуль подключен к порту COM3, его сетевые настройки приведены в таблице ниже.

**Таблица 4.1 – Сетевые настройки модуля МВ110-8А**

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| COM-порт контроллера | СOM3 |
| Адрес модуля | 1 |
| Скорость обмена | 115200 |
| Количество бит данных | 8 |
| Контроль четности | Отсутствует |
| Количество стоп-бит | 1 |

Пример создан в среде **CODESYS 3.5 SP11 Patch 5** и подразумевает запуск на **СПК1хх [М01]** с таргет-файлом **3.5.11.x.** В случае необходимости запуска проекта на другом устройстве следует изменить таргет-файл в проекте (**ПКМ** на узел **Device** – **Обновить устройство).**

Пример доступен для скачивания: [Example\_MV110\_8A\_DCON.zip](https://owen.ua/uploads/101/example_mv110_8a_dcon_3511v1.zip)

Листинг POU примера приведен в [приложении А](#_А._Листинг_программы).

## 4.2 Описание протокола

Протокол [DCON](http://www.bookasutp.ru/Chapter2_10.aspx) является одним из простейших строковых протоколов обмена. Он основан на архитектуре «Master-Slave» и реализуется поверх физического интерфейса RS-485. На примере протокола DCON будут описаны принципы разработки, с помощью которых можно реализовать любой строковый протокол обмена.

Структура кадра в общем виде выглядит следующим образом:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Символ начала сообщения** | **Адрес** | **Команда** | **Данные** | **Контрольная сумма** | **Символ конца сообщения** |
| 1 байт | 2 байта | 1...5 байт | 1…256 байт | 2 байта | 1 байт |

В зависимости от конкретного устройства количество используемых полей кадра может быть различным.

В руководстве по эксплуатации на модуль МВ110-8А приведена информация по реализации протокола для данного устройства. Модуль поддерживает всего две команды: единичный и групповой запрос результатов измерений каналов модуля. В рамках данного примера будет реализован групповой запрос.

Соответствующая команда выглядит следующим образом:

**#AA[CHK](cr)**

где **#** – символ начала сообщения;

**AA** – адрес модуля (в HEX);

**[CHK]** – контрольная сумма (в HEX);

**(cr)** – символ конца сообщения ($R).

Ответ от модуля будет иметь следующий вид:

**>(data)[CHK](cr)**

где  **>** – символ начала сообщения;

**(data)** – записанные подряд без пробелов результаты всех 8 измерений в десятичном представлении. Длина каждой записи об одном измерении равна семи символам (знак, десятичная точка и 5 цифр). Положение десятичной точки модуль выбирает автоматически в зависимости от измеренного значения. В случае возникновения в измерительном канале исключительной ситуации возвращается значение **−99999** или **+99999**. Диагностики для определения типа исключительной ситуации не производится.

**[CHK]** – контрольная сумма (в HEX);

**(cr)** – символ конца сообщения ($R).

В случае получения синтаксически неверного запроса или несоответствии контрольной суммы модуль не отвечает.

Контрольная сумма представляет собой сумму значений кодов всех ASCII символов команды, исключающую символы самой контрольной суммы. Если ее значение превышает **16#FF**, то используется только младший байт.

Пример запроса к модулю с адресом **1**:

**#01[CHK]$R**

Адрес модуля записывается в виде двух HEX символов. Для модуля с адресом **74** запрос бы выглядел как **#4A[CHK]$R**.

Контрольная сумма для модуля с адресом 1 представляет собой сумму ASCII кодов символов запроса, расположенных до контрольной суммы: #, 0 и 1. Узнав коды символов (непосредственно из [таблицы ASCII](https://ru.wikipedia.org/wiki/ASCII) или, например, воспользовавшись одним из [онлайн-конвертеров](http://www.asciitohex.com/)), можно вычислить контрольную сумму. В данном случае, символ «#» имеет код 23h, символ «0» – 30h, символ «1» – 31h, и, соответственно, CHK = 23h + 30h + 31h = 84h.

Таким образом, запрос к модулю с адресом 1 с рассчитанной контрольной суммой будет иметь вид:

**#0184$R**

Пример ответа от модуля выглядит следующим образом:

**>+100.23+34.050+124.56+07.331-101.45+1038.9-50.501+05.880[CHK]$R**

## Алгоритмизация задачи

Процесс обмена данными через последовательный порт можно представить в виде циклически выполняемого алгоритма (см. [п. 2.1](#_2.1._Общие_принципы)). На языке ST для реализации подобных алгоритмов в большинстве случаев используется оператор условного выбора CASE.

Алгоритм решаемой в примере задачи можно представить следующим образом:

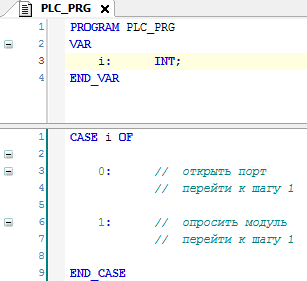


Рисунок 4.1 – Алгоритмизация задачи опроса модуля

Во время запуска программы будет однократно произведено открытие порта, после чего начнется циклический опрос модуля.

Соответственно, задача сводится к написанию кода, который будет выполняться в шагах 0 и 1. Рекомендуется упаковать его в два функциональных блока:

1. ФБ управления портом, который будет вызываться на шаге 0.
2. ФБ опроса модуля, который будет вызываться на шаге 1.

Предварительно следует добавить в проект библиотеку [CAA SerialCom](#_3.1._Добавление_библиотеки), так как ее функции и ФБ потребуются при реализации протокола.

## ФБ управления портом (COM\_CONTROL)

Функциональный блок управления COM-портом с названием **COM\_CONTROL** используется для открытия порта (для начала обмена) и его закрытия в случае возникновения ошибок или необходимости остановить обмен.

В качестве примера рассмотрим типичный функционал блока, чтобы определиться с набором входных и выходных переменных:

1. ФБ должен уметь открывать порт с заданными настройками по логическому сигналу.
2. ФБ должен уметь закрывать порт по логическому сигналу.
3. ФБ должен содержать выход, который сигнализирует о том, что порт успешно открыт.
4. ФБ должен содержать выход, который получает номер дескриптора (handle) открытого порта.
5. ФБ должен содержать выходы, сигнализирующие о возникновении ошибок при открытии и закрытии порта.

Получается следующий список переменных ФБ:

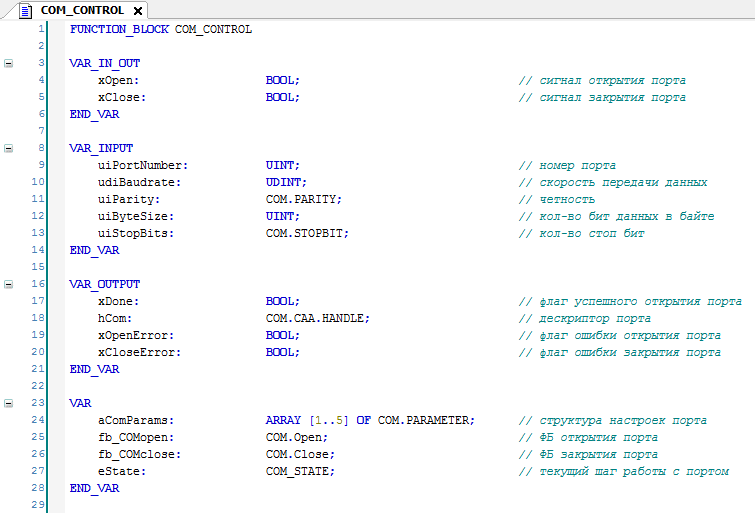


Рисунок 4.2 – Объявление переменных ФБ COM\_CONTROL

|  |  |
| --- | --- |
|  | **ПРИМЕЧАНИЕ**  Переменные **xOpen** и **xClose** объявлены как **VAR\_IN\_OUT** – это позволяет менять их значения из ФБ. Таким образом, можно автоматически сбрасывать входные сигналы открытия и закрытия порта, чтобы предотвратить циклическое выполнение данных операций. |

К локальным переменным ФБ относится структура параметров COM-порта и экземпляры ФБ [COM.Open](#_3.4._ФБ_COM.Open) и [COM.Close](#_3.7._ФБ_COM.Close), входящие в состав библиотеки [CAA SerialCom](#_3._Библиотека_CAA).

Как и сам алгоритм опроса, работу с COM-портом рекомендуется реализовывать через оператор CASE:



Рисунок 4.3 – Алгоритм ФБ управления COM-портом

На шаге 0 будет происходить инициализация ФБ (завершение операций открытия/закрытия порта, завершенных при предыдущем вызове), на шаге 1 – ожидание управляющего сигнала. В зависимости от типа сигнала (открытие или закрытие порта), будет осуществлен переход к шагу 2 или 3. После завершения операции должно произойти возвращение к шагу 0.

На рисунке 4.3 для обозначения шагов используются порядковые номера. Данный подход лишен наглядности и затрудняет понимание алгоритма. Поэтому следует создать **перечисление** **COM\_STATE**, которое позволит использовать имена для обозначения шагов (**Application** – **Добавление объекта** – **DUT** – **Перечисление**):

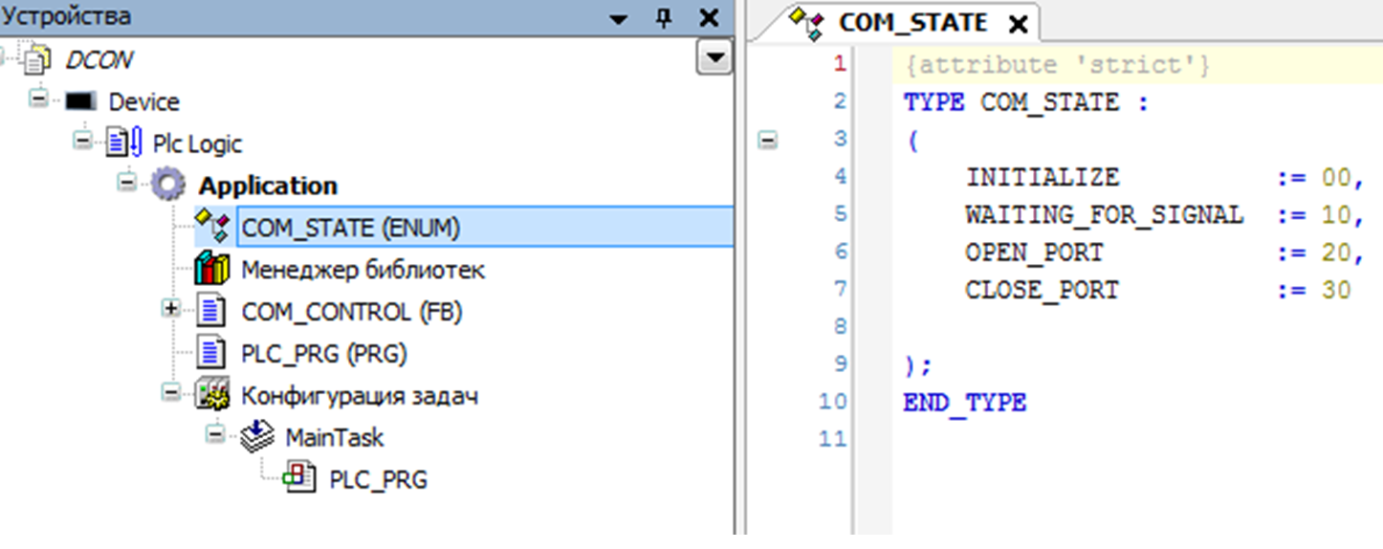


Рисунок 4.4 – Объявление перечисления COM\_STATE

Для работы с перечислением следует в переменных программы объявить его экземпляр. В примере экземпляр уже был объявлен (см. переменную **eState** на рисунке 4.2). Разрывы в номерах шагов позволяют в случае необходимости добавлять промежуточные шаги.

Теперь в операторе CASE можно использовать имена шагов, что явно повышает читабельность кода. Подобный прием будет использоваться и в процессе разработки ФБ опроса модуля.

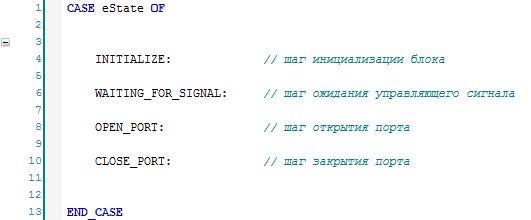


Рисунок 4.5 – Алгоритм ФБ управления COM-портом с использованием перечисления

### Инициализация порта (шаг INITIALIZE)

На шаге **INITIALIZE** происходит сброс ФБ открытия и закрытия порта, после чего осуществляется переход к шагу ожидания управляющего сигнала ([WAITING\_FOR\_SIGNAL](#_4.4.2._Шаг_WAITING_FOR_SIGNAL)):



Рисунок 4.6 – Код шага INITIALIZE

### Ожидание управляющего сигнала (шаг WAITING\_FOR\_SIGNAL)

На шаге **WAITING\_FOR\_SIGNAL** будет происходить ожидание управляющего сигнала. В случае детектирования сигнала происходит переход к соответствующему шагу ([OPEN\_PORT](#_4.4.3._Шаг_OPEN_PORT) или [CLOSE\_PORT](#_4.4.4._Шаг_CLOSE_PORT)):

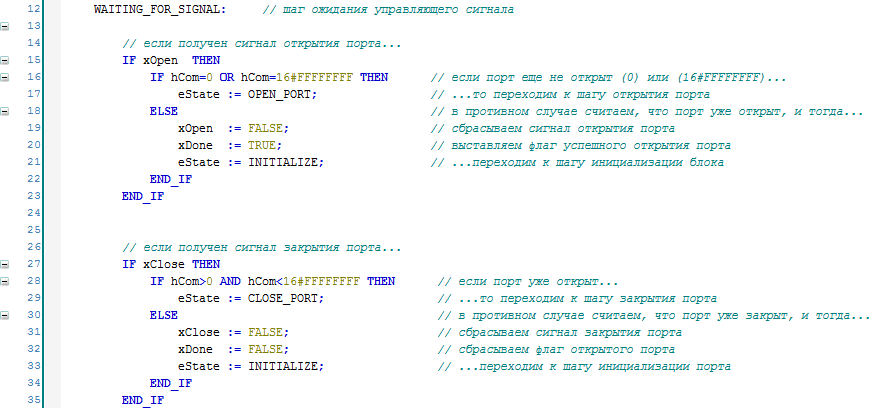


Рисунок 4.7 – Код шага WAITING\_FOR\_SIGNAL

При получении управляющего сигнала производится проверка дескриптора (handle) **hCom**. Значение **0** соответствует закрытому порту, значение **16#FFFFFFFF** – ошибке открытия порта. Значения между **0** и **16#FFFFFFFF** соответствуют открытому порту. Соответственно, не имеет смысла открывать уже открытый порт и закрывать закрытый – в данном случае следует сбросить управляющий сигнал, обновить значение выхода **xDone** и перейти на шаг инициализации ФБ.

Если порт закрыт и получен сигнал открытия порта, то происходит переход к шагу [OPEN\_PORT](#_4.4.3._Шаг_OPEN_PORT). Если порт открыт и получен сигнал закрытия порта – к шагу [CLOSE\_PORT](#_4.4.4._Шаг_CLOSE_PORT).

### Открытие порта (шаг OPEN\_PORT)

На шаге **OPEN\_PORT** следует открыть порт с заданными параметрами (параметры являются входным переменными ФБ **COM\_CONTROL**).

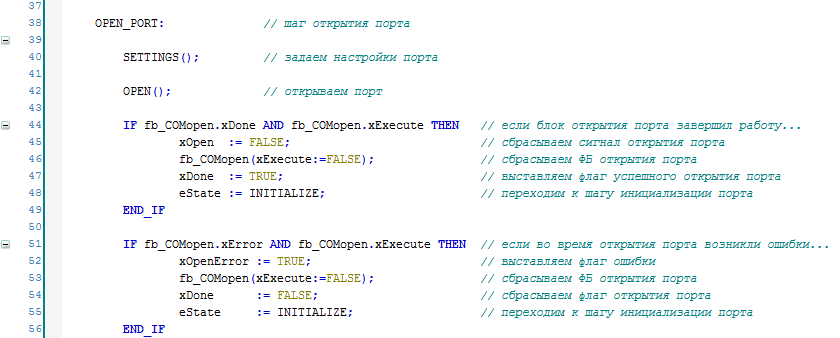


Рисунок 4.8 – Код шага OPEN\_PORT

На этом шаге используется еще один прием структурирования программы – выделение законченных фрагментов кода в **действия**. Действия являются вложенными **POU**, для создания которых необходимо нажать **ПКМ** на основной **POU** и выбрать команду **Добавление объекта** – **Действие**. В примере для ФБ **COM\_CONTROL** будут созданы три действия – **SETTINGS** (задание настроек порта), **OPEN** (открытие порта) и **CLOSE** (закрытие порта).



Рисунок 4.9 – Действия ФБ COM\_CONTROL

Код действия **SETTINGS** выглядит следующим образом:

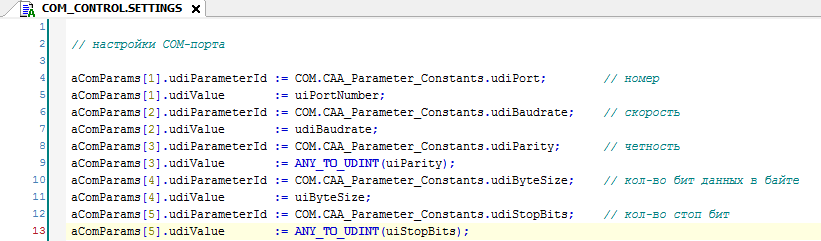


Рисунок 4.10 – Код действия SETTINGS

Массив **aComParams** содержит настройки COM-порта. Каждый его элемент представляет собой структуру типа [COM.PARAMETER](#_3.2._Cтруктура_COM.PARAMETER), содержащую две переменных – имя параметра (**udiParameterId**) и его значение (**udiValue**). Имена параметров определены в списке глобальных констант **CAA\_Parameter\_Constant**. Их значения задаются на входе ФБ **COM\_CONTROL**.

Параметры **uiParity** и **uiStopBits** представляют собой перечисления (**COM.Parity** и **COM.STOPBIT** соответственно). Чтобы избежать предупреждений компилятора, их следует преобоазовать к нужному типу (т. е. к типу переменной **udiValue** – **UDINT**). Узнать фактический тип перечислений можно, посмотрев их описание в библиотеке, но проще будет воспользоваться функцией **ANY\_TO\_UDINT**, которая подходит для конверсии любого типа.

Код действия **OPEN** выглядит следующим образом:

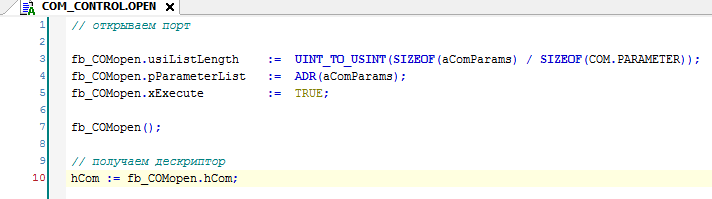


Рисунок 4.11 – Код действия OPEN

В данном действии производится работа с экземпляром ФБ [COM.Open](#_3.4._ФБ_COM.Open).

Сначала задаются входные параметры блока.

Вход **usiListLength** содержит число используемых параметров COM-порта. Разумеется, их можно задать и в явном виде (в рамках нашего примера usiListLength := 5), но тогда в случае необходимости изменить число параметров придется менять и данное значение. Этого можно избежать с помощью оператора **SIZEOF**, который возвращает размер переменной в байтах. В результате деления размера массива параметров на размер одного параметра будет получено текущее число параметров. При изменении размерности массива новое значение будет рассчитано автоматически.

Вход **pParamterList** содержит адрес структуры параметров COM-порта.

Вход **xExecute** используется для управления блоком.

После установки входных параметров необходимо вызвать ФБ, что приведет к открытию COM-порта с заданными настройками.

Затем следует получить значение дескриптора (handle), присвоив его выходной переменной **hCom** ФБ **COM\_CONTROL**.

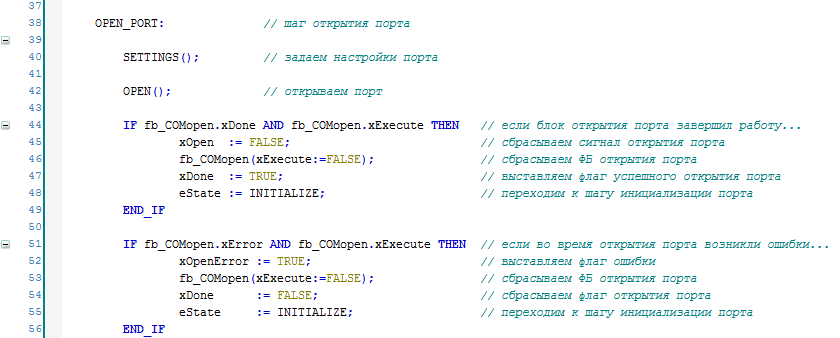


Рисунок 4.12 – Код шага OPEN\_PORT

После выполнения действия **OPEN** следует проанализировать его результат. Если порт успешно открыт (у **fb\_COMopen** выход **xDone = TRUE**), то происходит сброс управляющего сигнала, остановка работы **fb\_COMopen**, взведение флага успешного открытия порта (выход **xDone** ФБ **COM\_CONTROL**) и переход на шаг инициализации порта ([INITIALIZE](#_4.4.1._Шаг_INITIALIZE)).

Если при открытии порта произошла ошибка (у ФБ **fb\_COMopen** выход **xError = TRUE**), то происходит взведение флага ошибки открытия порта, остановка работы **fb\_COMopen**, сброс флага успешного открытия порта (выход **xDone** ФБ **COM\_CONTROL**) и переход на шаг инициализации порта ([INITIALIZE](#_4.4.1._Шаг_INITIALIZE)).

### Закрытие порта (шаг CLOSE\_PORT)

В случае перехода на шаг **CLOSE\_PORT** следует закрыть порт. Закрытие порта может потребоваться в случае необходимости прекращения обмена или обработке ошибок обмена.

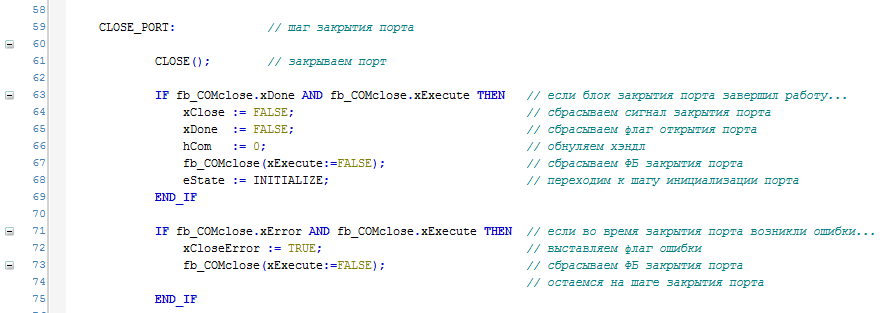


Рисунок 4.13 – Код шага CLOSE\_PORT

Как и при открытии порта, код закрытия следует поместить в действие.

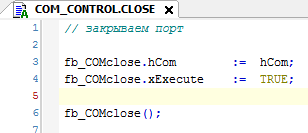


Рисунок 4.14 – Код действия CLOSE

Для закрытия порта достаточно указать его дескриптор (handle) **hCom** и вызвать **fb\_COMclose** с **xExecute = TRUE**.

После выполнения действия **СLOSE** следует проанализировать его результат. Если порт успешно открыт (у **fb\_COMclose** выход **xDone = TRUE**), то происходит сброс управляющего сигнала, сброс флага успешного открытия порта, обнуление значение дескриптора (handle) на выходе ФБ **COM\_CONTROL**, остановка работы **fb\_COMclose** и переход на шаг инициализации порта.

Если при закрытии порта произошла ошибка (у ФБ **fb\_COMclose** выход **xError = TRUE**), то происходит взведение флага ошибки закрытия порта и остановка работы **fb\_COMclose**. Программа остается на том же шаге выполнения, т. е. в следующем цикле опять будет произведена попытка закрытия порта и так до тех пор, пока не произойдет успешное закрытие порта.

Теперь следует вернуться к основной программе [PLC\_PRG](#_4.3._Алгоритмизация_задачи), объявить экземпляр ФБ **COM\_CONTROL** и написать код для шага 0:

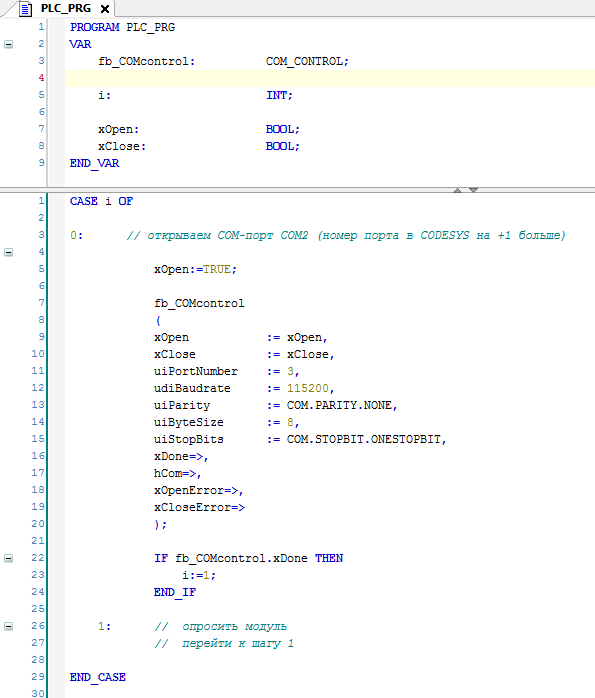


Рисунок 4.15 – Алгоритмизация задачи опроса модуля с кодом шага 0

В рамках примера не будет рассматриваться закрытие порта, поэтому следует в явном виде присвоить переменной **xOpen** значение **TRUE**. Во время запуска программы шаг 0 будет выполнен однократно, поскольку возвращение на него не происходит.

Порт с заданными настройками открыт – теперь можно отправлять в него данные и считывать ответы. Теперь следует приступить к реализации протокола DCON.

Листинг ФБ **COM\_CONTROL** приведен в [приложении А.1](#_A.1._ФБ_COM_CONTROL).

## ФБ опроса модуля (MV110\_8A\_DCON)

Для опроса модуля **МВ110-8А** по протоколу **DCON** следует создать ФБс названием **MV110\_8A\_DCON**. В рамках примера его входными переменными будут являться:

1. Вход управления блоком (старт/прекращение работы).
2. Адрес опрашиваемого устройства.
3. Дескриптор (handle) COM-порта.

К выходным переменным будут относиться:

1. Измеренные значения каналов модуля.
2. Флаг окончания цикла опроса.
3. Флаги ошибок обмена.

Ниже также приводится список локальных переменных ФБ. Необходимость их объявления будет поясняться по мере описания принципов работы блока.

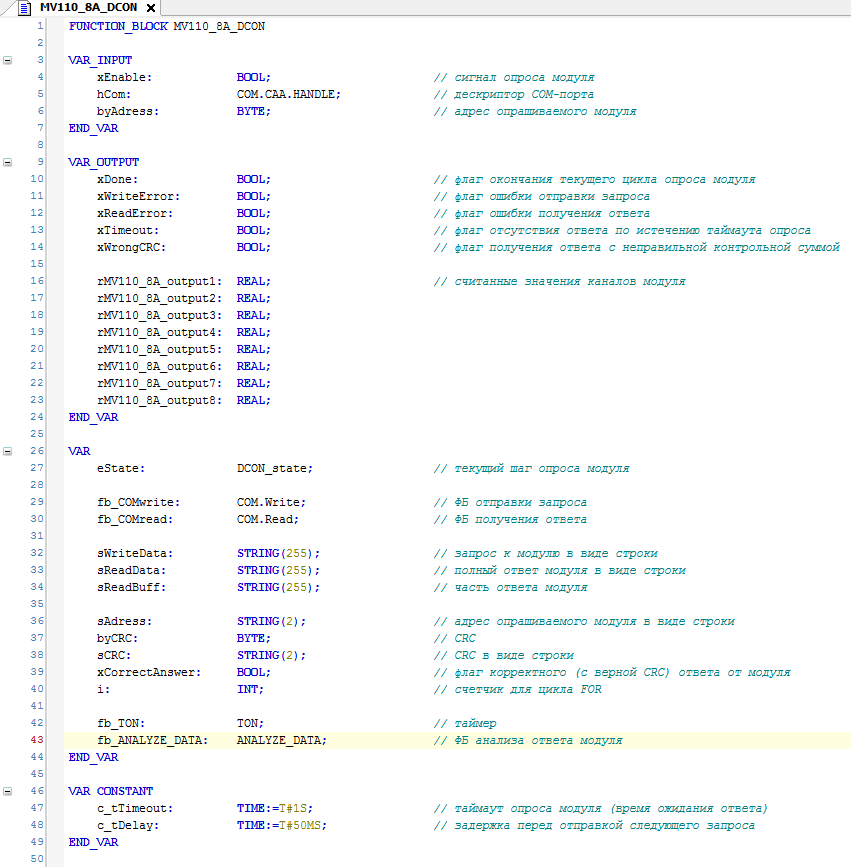


Рисунок 4.16 – Объявление переменных ФБ MV110\_8A\_DCON

Для реализации блока будет использоваться пошаговый алгоритм с оператором CASE и **перечислением**, которое позволит использовать имена в качестве названий шагов. Экземпляр перечисления имеет название **eState**.

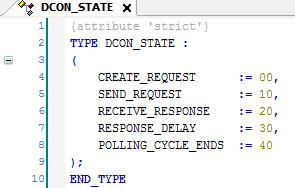


Рисунок 4.17 – Объявление перечисления DCON\_STATE

Алгоритм опроса модуля в общем виде:

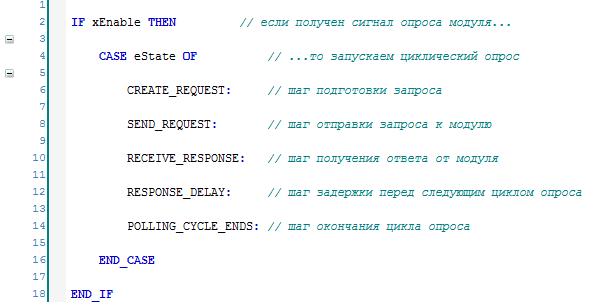


Рисунок 4.18 – Алгоритм опроса модуля

На шаге **CREATE\_REQUEST** будет происходить формирование запроса к модулю.

На шаге **SEND\_REQUEST** запрос будет отправлен в COM-порт.

На шаге **RECEIVE\_RESPONSE** будет происходить получение и анализ ответа от модуля.

Шаг **RESPONSE\_DELAY** будет использоваться для создания задержки между получением ответа от модуля и отправки следующего запроса.

На шаге **POLLING\_CYCLE\_ENDS** будет происходить формирование значений выходов ФБ, после чего следует переход к начальному шагу.

|  |  |
| --- | --- |
|  | **ПРИМЕЧАНИЕ**  Необходимость наличия шага **RESPONSE\_DELAY** зависит от особенностей конкретного опрашиваемого устройства. Некоторые устройства удерживают линию связи фиксированное время после ответа, в связи с чем следует организовывать задержку перед отправкой следующего запроса. В случае опроса модуля МВ110-8А такая задержка не обязательна, но рассматривается в настоящем руководстве в качестве примера. |

Законченные фрагменты кода ФБ будут оформлены в виде **действий**:

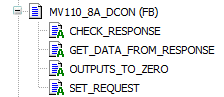


Рисунок 4.19 – Действия ФБ MV110\_8A\_DCON

### Подготовка запроса (шаг CREATE\_REQUEST)

На шаге **CREATE\_REQUEST** происходит очистка буферов данных и подготовка запроса.

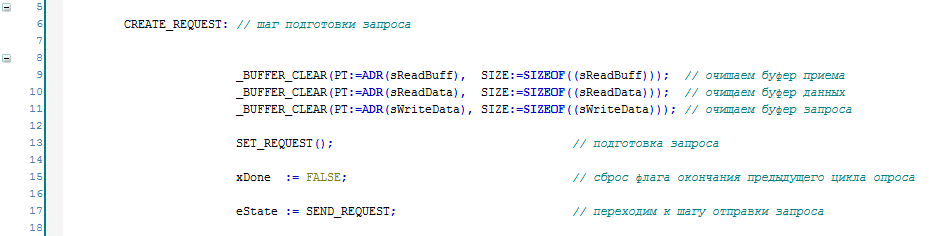


Рисунок 4.20 – Код шага CREATE\_REQUEST

Под буфером понимается переменная, используемая для временного хранения данных. В данном случае такими переменными являются **sReadBuff** (буфер фрагмента ответа), **sReadData** (буфер полного ответа) и **sWriteData** (буфер запроса). В рамках примера модулю каждый раз отправляется один и тот же запрос, поэтому, в целом, нет реальной необходимости каждый раз очищать буфер запроса и формировать запрос заново – но в других ситуациях это может понадобиться (например, когда необходимо с помощью одного ФБ считывать несколько параметров устройства с помощью разных запросов).

Для очистки буфера используется функция **\_BUFFER\_CLEAR** из библиотеки **OSCAT**. Библиотека доступна для скачивания на сайте [oscat.de](http://www.oscat.de/) и [сайте компании Овен](http://www.owen.ua/) в разделе **CODESYS V3/Библиотеки**. Библиотека OSCAT имеет открытые исходные коды, поэтому во многих случаях удобнее копировать ее функции и ФБ в пользовательский проект (вместо добавления через **Менеджер библиотек**).

Функция **\_BUFFER\_CLEAR** имеет два входных параметра: адрес буфера и его размер. Вместо задания размера буфера с помощью числа удобнее использовать оператор **SIZEOF**, возвращающий размер переменной в байтах. Возвращаемое значение имеет тип **BOOL** и принимает значение **TRUE** в случае успешного завершения очистки буфера.

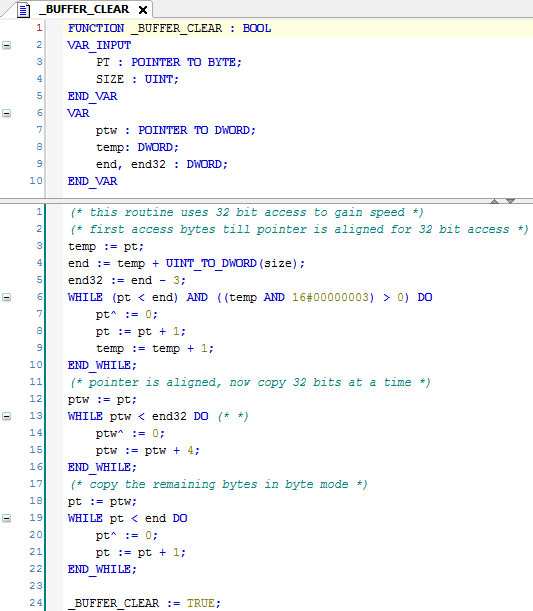


Рисунок 4.21 – Код функции \_BUFFER\_CLEAR из библиотеки OSCAT

Код функции скопирован из библиотеки **OSCAT** без каких-либо изменений.

Действие **SET\_REQUEST** используется для формирования запроса к модулю. Структура запроса известна из описания протокола **DCON**.

|  |  |
| --- | --- |
|  | **ПРИМЕЧАНИЕ**  DCON является строковым протоколом – т. е. все данные в нем передаются в виде ASCII-кодов соответствующих символов. |

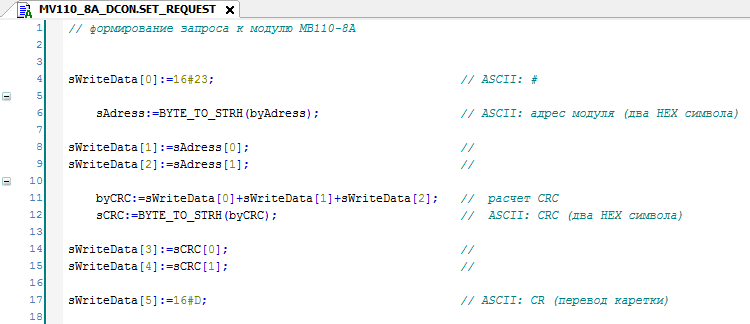


Рисунок 4.22 – Код действия SET\_REQUEST

В **CODESYS V3.5** со строковыми переменными можно работать как с массивом байт, что является очень удобным при наполнении буфера.

Структура запроса к модулю выглядит следующим образом:

**Таблица 4.2 – Структура запроса по протоколу DCON**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Старт символ** | **Адрес модуля** | **Контрольная сумма** | **Стоп символ** |
| **Размер** | 1 байт | 2 байта | 2 байта | 1 байт |
| **Значение** | # | Переменная | Переменная | CR |
| **Код ASCII** | 16#23 | Переменная | Переменная | 16#D |

Входная переменная **byAdress** имеет тип **BYTE** – поэтому ее следует конвертировать в два HEX символа с помощью **BYTE\_TO\_STRH** из библиотеки OSCAT (см. рисунок 4.23). Код функции скопирован из библиотеки **OSCAT** без каких-либо изменений.

Методика расчета контрольной суммы известна из описания протокола – она, действительно, представляет собой сумму всех элементов буфера, расположенных до контрольной суммы (т. е. старт-символ и адрес модуля). В данном случае контрольная сумма вычисляется в явном виде, но в случае необходимости (например, в случае опроса нескольких различных модулей) ее расчет можно вынести в отдельную функцию.

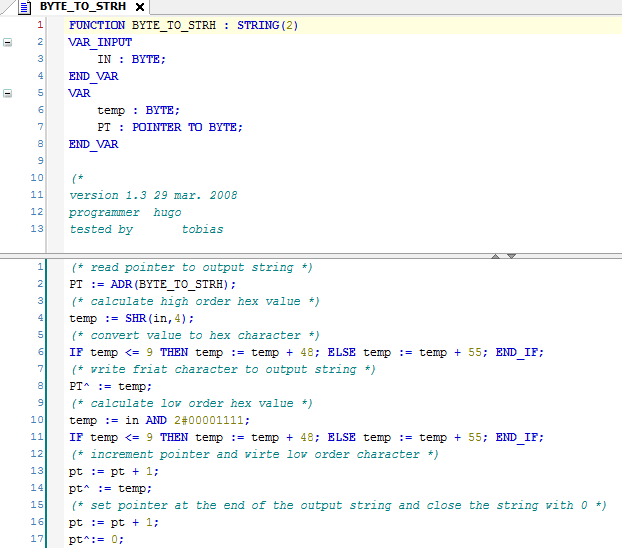


Рисунок 4.23 – Код функции BYTE\_TO\_STRH из библиотеки OSCAT

Теперь следует вернуться к коду шага **CREATE\_REQUEST**.

Кроме очистки буферов и формирования запроса на данном шаге происходит сброс флага окончания предыдущего цикла, после чего осуществляется переход к шагу [SEND\_REQUEST**.**](#_4.5.2._Шаг_SEND_REQUEST)

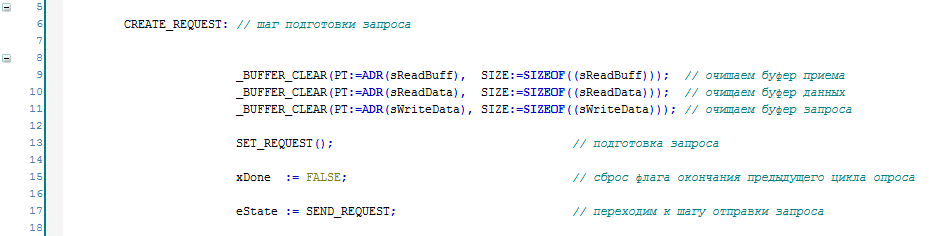


Рисунок 4.24 – Код шага CREATE\_REQUEST

### Отправка запроса (шаг SEND\_REQUEST)

На шаге **SEND\_REQUEST** происходит отправка запроса, сформированного на предыдущем шаге.

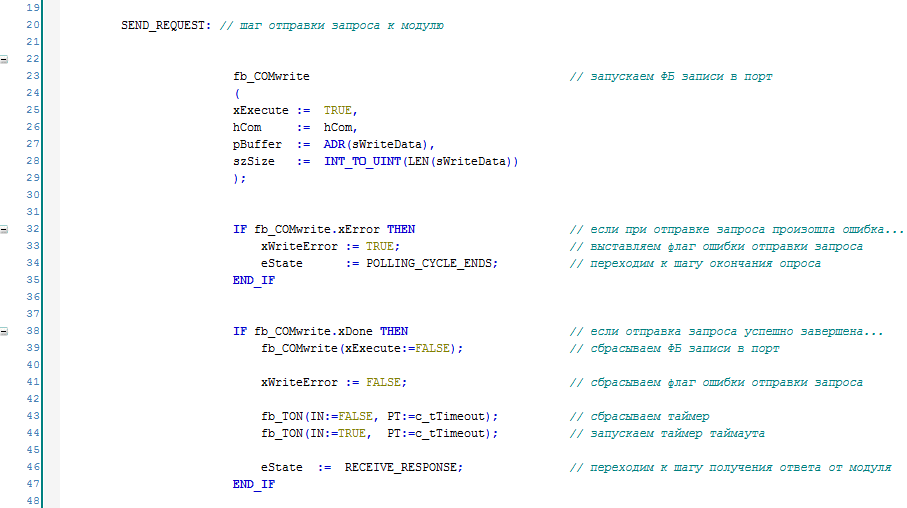


Рисунок 4.25 – Код шага SEND\_REQUEST

В этом шаге производится работа с экземпляром ФБ [COM.Write](#_3.5._ФБ_COM.Write).

Входными переменным блока являются сигнал его запуска (**xExecute**), дескриптор (handle) COM-порта (**hCom**), адрес буфера запроса (**pBuffer**) и размер буфера (**szSize**). В строковом протоколе размер запроса определяется числом его символов – поэтому следует использовать функцию **LEN**, которая возвращает длину строки.

После вызова блока отправки запроса следует проанализировать его результат. Если во время отправки произошла ошибка (у ФБ **fb\_COMwrite** выход **xError = TRUE**), то следует взвести флаг ошибки и перейти к шагу завершения опроса.

Если запрос успешно отправлен (у **fb\_COMwrite** выход **xDone = TRUE**), то следует завершить работу блока, сбросить флаг ошибки, запустить таймер таймаута (он потребуется на следующем шаге) и перейти к шагу [RECEIVE\_RESPONSE](#_4.5.3._Шаг_RECEIVE_RESPONSE).

### Получение ответа (шаг RECEIVE\_RESPONSE)

На шаге **RECEIVE\_RESPONSE** происходит получение ответа от модуля.

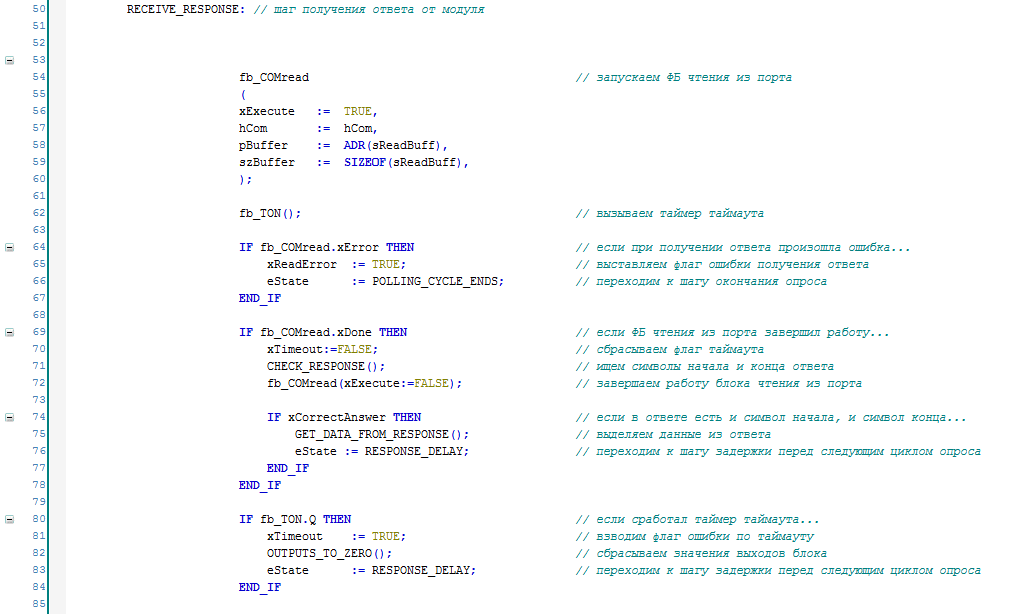


Рисунок 4.26 – Код шага RECEIVE\_RESPONSE

На данном шаге производится работа с экземпляром ФБ [COM.Read](#_3.6._ФБ_COM.Read).

Входными переменным блока являются сигнал его запуска (**xExecute**), дескриптор (handle) COM-порта (**hCom**), адрес буфера фрагмента ответа (**pBuffer**) и размер буфера (**szSize**). Вместо задания размера буфера с помощью числа рекомендуется использовать оператор **SIZEOF**, возвращающий размер переменной в байтах. В данном случае функция **LEN** не требуется, т. к. размер буфера приема в большинстве случаев является статической величиной, зависящей только от размера соответствующей переменной.

В ряде случаев подчиненное устройство не ответит на запрос (например, в случае некорректного запроса или неисправности устройства). Если не предусмотреть эту ситуацию, то программа может «застрять» на шаге получения ответа. Для предотвращения данной ситуации следует использовать контроль таймаута опроса – по истечению заданного времени программа выходит из текущего шага вне зависимости от факта получения ответа. Таймер таймаута был запущен в конце предыдущего шага, поэтому на данном шаге достаточно вызвать его с теми же аргументами. Время таймаута определяется с помощью константы **c\_tTimeout**.

После вызова блока получения ответа следует проанализировать его результат. Если при получении произошла ошибка (у **fb\_COMread** выход **xError = TRUE**), то следует взвести флаг ошибки и перейти к шагу завершения опроса.

Если ответ успешно получен (у **fb\_COMread** выход **xDone = TRUE**), то следует сбросить флаг ошибки таймаута, проверить корректность ответа и завершить работу блока.

Проверка корректности ответа вынесена в действие **CHECK\_RESPONSE**:

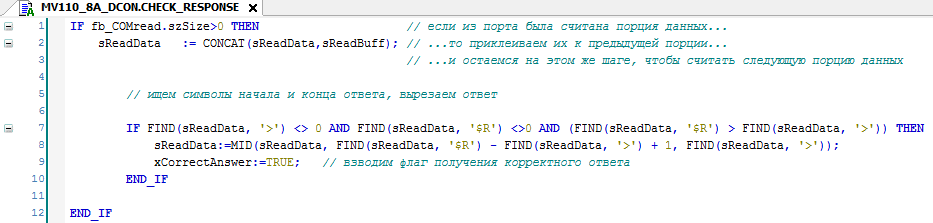


Рисунок 4.27 – Код действия CHECK\_RESPONSE

В некоторых случаях (в частности, при чтении большого количества данных), ответ от подчиненного устройства может прийти в виде нескольких фрагментов. Корректный способ обработки данной ситуации: пока не сработал таймер таймаута, каждый полученный фрагмент ответа следует «приклеивать» к предыдущему фрагменту с помощью функции **CONCAT** в буфере полного ответа (**sReadData**), после чего искать в буфере старт-символ и стоп-символ с помощью функции **FIND**. Если оба символа обнаружены, и позиция стоп символа больше позиции старт-символа, то можно сделать вывод, что в буфере находится ответ от подчиненного устройства. В ряде специфических случаев ответ может находиться не в начале буфера, поэтому следует вырезать его, ориентируясь на позиции старт- и стоп-символа. Вырезанный ответ помещается в тот же самый буфер, после чего взводится флаг получения корректного ответа (**xCorrectAnswer**).

**Вырезание из буфера полученного ответа (от старт-символа до стоп-символа)**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| … | Старт-символ | … | … | … | Стоп-символ | … |
| Старт- символ | … | … | … | Стоп-символ | … | … |

Если один из символов не найден (например, пришел первый фрагмент ответа, в котором содержится старт-символ, но стоп-символа еще нет), то программа остается на данном шаге до истечения таймаута, что позволяет получить следующие фрагменты.

Затем следует вернуться к шагу **RECEIVE\_RESPONSE** (см. [рисунок 4.26](#Ris4_26)). После выполнения кода действия **CHECK\_RESPONSE** следует завершить работу **ФБ COM.Read**. Если взведен флаг корректного ответа (**xCorrectAnswer**), то следует перейти к действию **GET\_DATA\_FROM\_RESPONSE**, в котором будет выполняться разбор полученного ответа.

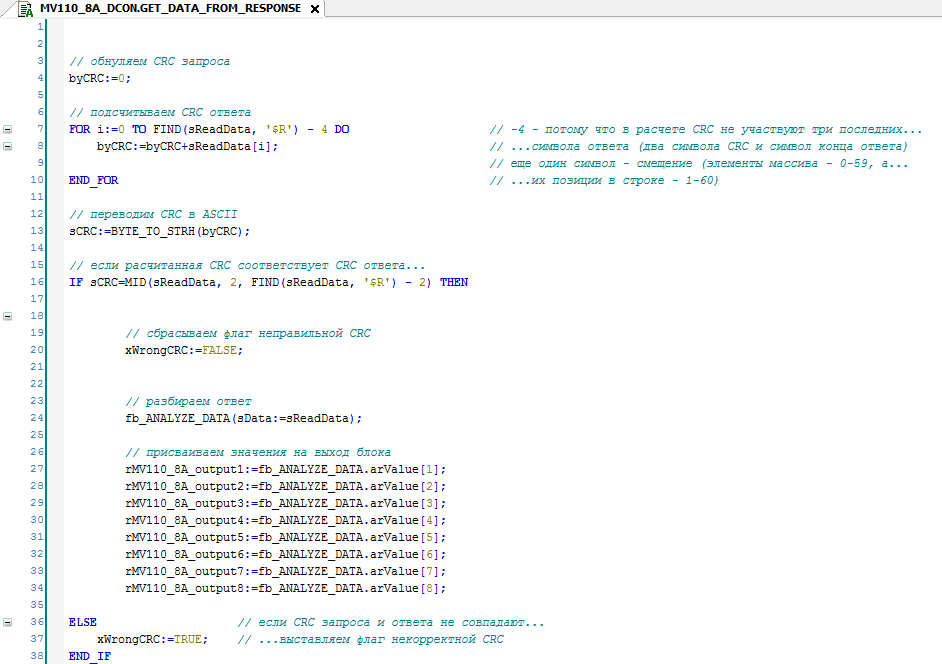


Рисунок 4.28 – Код действия GET\_DATA\_FROM\_RESPONSE

Сначала следует проверить контрольную сумму. Контрольная сумма представляет собой сумму ASCII кодов всех символов, расположенных перед контрольной суммой. Следует сравнить ее с контрольной суммой ответа, предварительно преобразовав в строковый вид с помощью функции **BYTE\_TO\_STRH** из библиотеки **Oscat** (из описания протокола известно, что контрольная сумма находится в двух байтах перед стоп-символом).

Если рассчитанная контрольная сумма совпадает с полученной, то можно переходить к разбору ответа. Предварительно следует сбросить флаг ошибки контрольной суммы.

Разбор ответа производится с помощью экземпляра ФБ **ANALYZE\_DATA**. Входной переменной для блока является ответ модуля в виде строки, выходной – массив из значений типа **REAL**.

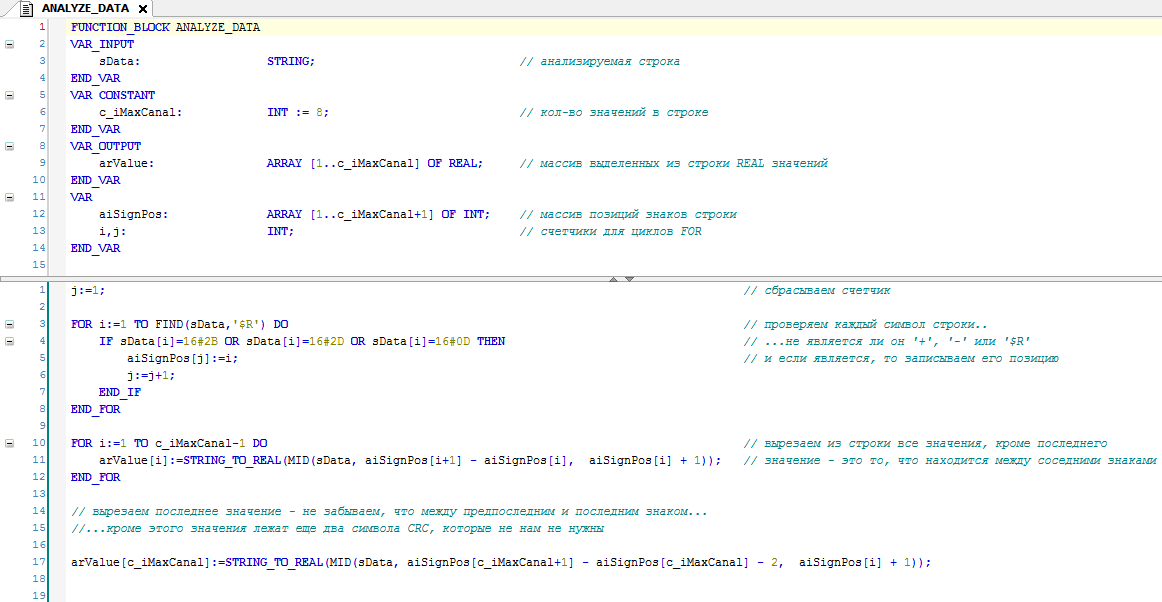


Рисунок 4.29 – Код ФБ ANALYZE\_DATA

Как преобразовать строку со значениями в переменные типа **REAL**, при условии, что количество символов для каждого значения нам заранее неизвестно? (например, оно может быть как таким: **11.22**, так и таким: **333.444**)

Для преобразования строки со значениями в переменные типа REAL, при условии, что количество символов для каждого значения заранее неизвестно, следует ориентироваться на специальные символы-разделители. В контексте чисел такими знаками являются «+» и «−». Можно сказать, что значение – это то, что расположено между двумя любыми знаками, включая первый знак.

Сначала следует найти в строке данные знаки и запомнить их позиции. Также требуется найти стоп-символ для определения последнего значения в ответе.

В качестве примера приведем ответ, в котором содержатся 3 значения (1.2, −33.44 и 555.666). В реальном ответе от модуля будет 8 значений фиксированного размера, но принцип останется неизменным.

Затем следует найти позиции знаков и стоп-символа (они соответственно, равны 2, 6, 12, 22). Таким образом, первое значение занимает символы 2–5, второе – 6–11, а третье 12–19 (известно, что между последним значением и стоп-символом также находятся два символа CRC). Затем следует вырезать из строки соответствующие фрагменты и преобразовать их функцией **STRING\_TO\_REAL**. Ввиду упомянутых особенностей, последнее значение придется обрабатывать отдельно.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Символ ответа** | > | + | 1 | . | 2 | - | 3 | 3 | . | 4 | 4 | + | 5 | 5 | 5 | . | 6 | 6 | 6 | CRC1 | CRC2 | $R |
| **Позиция** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 |

Затем следует вернуться на уровень выше, в действие **GET\_DATA\_FROM\_RESPONSE** (см. [рисунок 4.28](#Ris4_28)). После анализа ответа получается массив из 8-и значений типа **REAL**. Значения массива следует присвоить выходным переменным ФБ **MV110\_8A\_DCON**.

Если рассчитанная контрольная сумма не совпала с контрольной суммой ответа, то нет смысла анализировать его – достаточно взвести флаг некорректной контрольной суммы (**xWrongCRC**).

Теперь следует перейти на уровень выше и вернуться в код шага **RECEIVE\_RESPONSE** (см. [рисунок 4.27](#Ris4_27)).

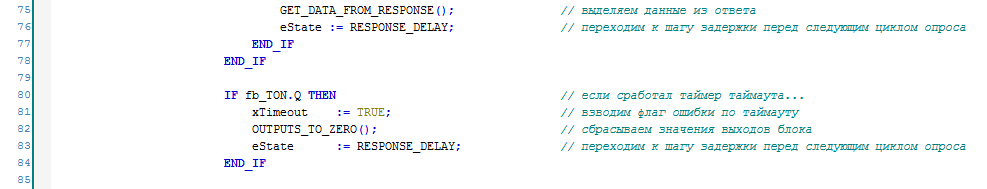


Рисунок 4.30 – Фрагмент кода шага RECCEIVE\_RESPONSE

После анализа ответа следует перейти к шагу задержки перед следующим циклом опроса.

Если сработал таймер таймаута (основная причина – отсутствие ответа от модуля в течение заданного интервала времени), то следует выставить флаг ошибки по таймауту, выполнить действие **OUTPUTS\_TO\_ZERO** (сброс выходных переменных ФБ **MV110\_8A\_DCON** со считанными с модуля значениями) и перейти к шагу **RESPONSE\_DELAY**.

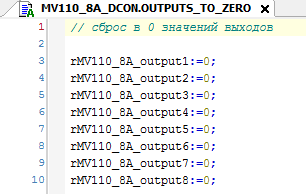


Рисунок 4.31 – Код действия OUTPUTS\_TO\_ZERO

### Организация задержки (шаг RESPONSE\_DELAY)

На шаге **RESPONSE\_DELAY** происходит запуск таймера задержки перед следующим циклом опроса. Задержка может понадобиться при опросе конкретных устройств, которые после отправки ответа блокируют линию связи на определенное время. Время задержки определяется с помощью константы **c\_tDelay**.

После запуска таймера происходит переход к шагу **POLLING\_CYCLE\_ENDS** .

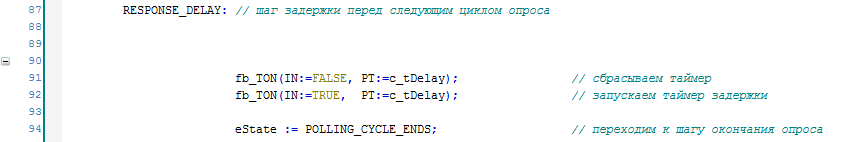


Рисунок 4.32 – Код шага RESPONSE\_DELAY

### Завершение цикла опроса (шаг POLLING\_CYCLE\_ENDS)

На шаге **POLLING\_CYCLE\_ENDS** следует вызвать таймер задержки перед следующим циклом опроса. После его срабатывания происходит завершение работы экземпляров ФБ чтения и записи [COM.Read](#_3.6._ФБ_COM.Read) и [COM.Write,](#_3.5._ФБ_COM.Write) сброс флага получения корректного ответа (в следующем цикле он должен быть просчитан заново), установка флага успешного завершения текущего цикла опроса и переход к шагу **CREATE\_REQUEST**.

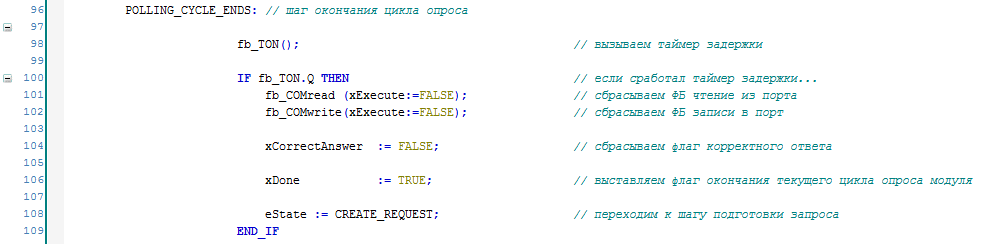


Рисунок 4.33 – Код шага POLLING\_CYCLE\_ENDS

После написания кода для всех шагов опроса модуля следует вернуться к записи алгоритма работы ФБ **MV110\_8A\_DCON** и вспомнить, что весь написанный код начинает выполняться только при условии **xEnable=TRUE**. В процессе работы программы может возникнуть ситуация, когда обмен следует остановить. Соответственно, в данном случае потребуется также выполнить ряд действий (в операторе **ELSE**) – сбросить флаги завершения опроса и получения корректного ответа, обнулить значения выходных переменных типа **REAL** (см. рисунок 4.31) и перейти на шаг **CREATE\_REQUEST** (чтобы при следующем запуске ФБ начать его выполнение с первого шага).



Рисунок 4.34 – Код ФБ MV110\_8A\_DCON (код отдельных шагов скрыт)

Функциональный блок опроса модуля **МВ110-8А** по протоколу **DCON** готов. Его листинг приведен в [приложении А.2](#_A.2._ФБ_MV110_8A_DCON).

## Программа опроса (PLC\_PRG)

Теперь следует вызвать созданные ранее ФБ в программе **PLC\_PRG**. Код программы выглядит следующим образом:

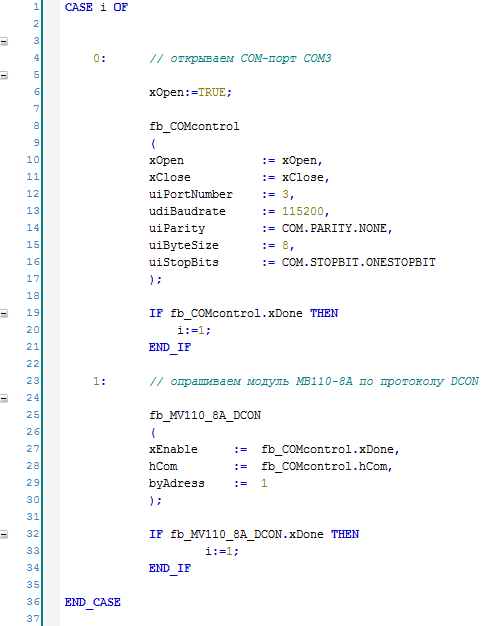


Рисунок 4.35 – Код программы PLC\_PRG

При запуске программы однократно выполняется код шага 0, что приводит к открытию COM-порта контроллера с заданными настройками с помощью ФБ **COM\_CONTROL**. После успешного открытия порта происходит переход на шаг 1, в котором с помощью ФБ **MV110\_8A\_DCON** организуется опрос модуля с заданным адресом. После окончания опроса модуля можно переходить к опросу следующего устройства. В примере других устройств нет, поэтому программа остается на текущем шаге и продолжает циклически опрашивать модуль.

Программа **PLC\_PRG** привязана к задаче с временем цикла **10 мс**. Рекомендуется привязывать программы обмена к задачам с наименьшим временем цикла (не более 10–20 мс).

На рисунке **4.36** приведен скриншот программы в процессе работы. Значение второго канала равно **99999** – что соответствует возникновению в канале исключительной ситуации (например, обрыв датчика). Это является особенностью модуля, а не протокола **DCON.**

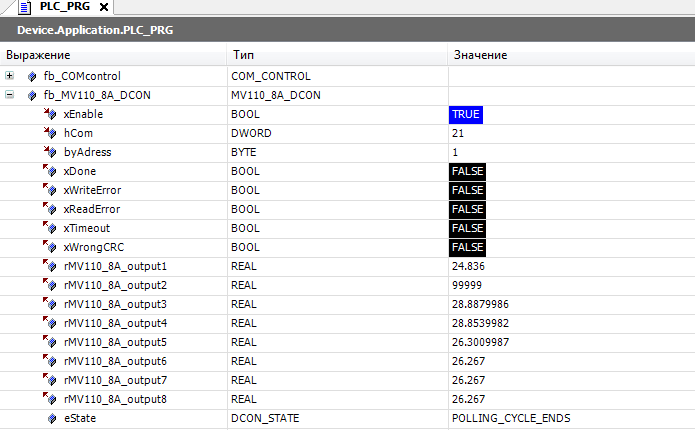


Рисунок 4.36 – Считанные значения каналов модуля

# Пример опроса счетчика СЭТ-4ТМ.03М

## Формулировка задачи

В качестве примера будет рассмотрен опрос счетчика [СЭТ-4ТМ.03М](http://www.nzif.ru/schetchiki-elektroenergii/trekhfaznye/mnogofunktsionalnye-set-psch/set-4tm-03m-set-4tm-02m.html) по нестандартному бинарному протоколу с помощью контроллера **СПК1хх [М01]**. В качестве опрашиваемого параметра используем текущее напряжение по фазе A **(Ua**). Счетчик подключен к порту **COM3**, его сетевые настройки приведены в таблице ниже.

**Таблица 5.1 – Сетевые настройки счетчика СЭТ-4ТМ**

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| *COM-порт контроллера* | СOM3 |
| Адрес модуля | 200 |
| Скорость обмена | 9600 |
| Количество бит данных | 8 |
| Контроль четности | Нечетный |
| Количество стоп-бит | 1 |

Пример создан в среде **CODESYS 3.5 SP11 Patch 5** и подразумевает запуск на **СПК1хх [М01]** с таргет-файлом **3.5.11.x.** В случае необходимости запуска проекта на другом устройстве следует изменить таргет-файл в проекте (**ПКМ** на узел **Device** – **Обновить устройство).**

Пример доступен для скачивания: [Example\_SET\_4TM.zip](https://owen.ua/uploads/101/example_set_4tm_3511v1.zip)

Листинг POU приведен в [приложении Б.](#_Б._Листинг_программы)

## Описание протокола

Счетчики **СЭТ-4ТМ.03М** поддерживают нестандартный Modbus-подобный двоичный протокол. Его полное описание может быть получено по запросу к [производителю](http://www.nzif.ru). Ниже приводятся ключевые моменты, на которые стоит обращать внимание при реализации обмена. Следует отметить, что в рамках примера будет использована не вся приведенная информация.

1. Критерием окончания ответа от счетчика является пауза длиной в 6–8 мс.
2. Максимальный размер буфера приема/передачи – 96 байт.
3. Структура запроса к счетчику:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Сетевой адрес** | **Код запроса** | **Код параметра** | **Параметры** | **Контрольная сумма** |
| 1 байт/5 байт | 1 байт/5 байт | 1 байт, может отсутствовать | До 91 байта, может отсутствовать | 2 байта |

Адрес и код запроса могут быть как обычными (занимать 1 байт), так и расширенными (занимать 5 байт). В рамках примера рассматривается работа только с обычными адресами и запросами.

1. Структура ответа счетчика:

|  |  |  |
| --- | --- | --- |
| **Сетевой адрес** | **Поле данных ответа** | **Контрольная сумма** |
| 1 байт/5 байт | до 93 байт | 2 байта |

1. Расчет контрольной суммы осуществляется аналогично протоколу Modbus (в спецификации протокола приведен алгоритм расчета).
2. Перед опросом счетчика следует открыть канал связи с помощью специального запроса следующего формата:

|  |  |  |  |
| --- | --- | --- | --- |
| **Сетевой адрес** | **Код запроса** | **Пароль** | **Контрольная сумма** |
| 1 байт/5 байт | 16#01 | 6 байт | 2 байта |

Пароль второго уровня (уровень «хозяин») доступа по умолчанию – **000000** в ASCII-кодах.

Если в течение 20 секунд счетчик не получает запросов, то канал связи закрывается.

1. Запрос для чтения текущего значения напряжения фазы А (Ua) выглядит следующим образом:

Код запроса – **16#08** (чтение параметров и данных);

Код параметра – **16#1B** (чтение данных в формате с плавающей точкой);

Параметры – один байт кода массива данных и один байт **RWRI** (регистр вспомогательных режимов измерения).

Код массива данных – **16#00**;

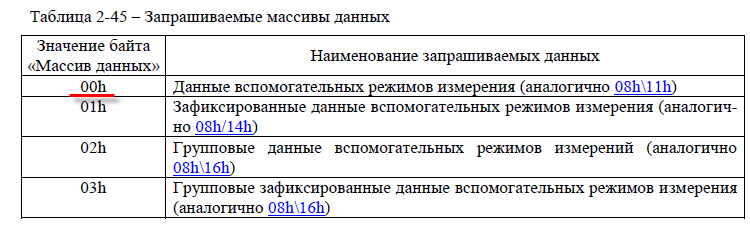


Рисунок 5.1 – Таблица кодов массивов данных из описания протокола

Значение RWRI – 0001 00 01 = **16#11**.

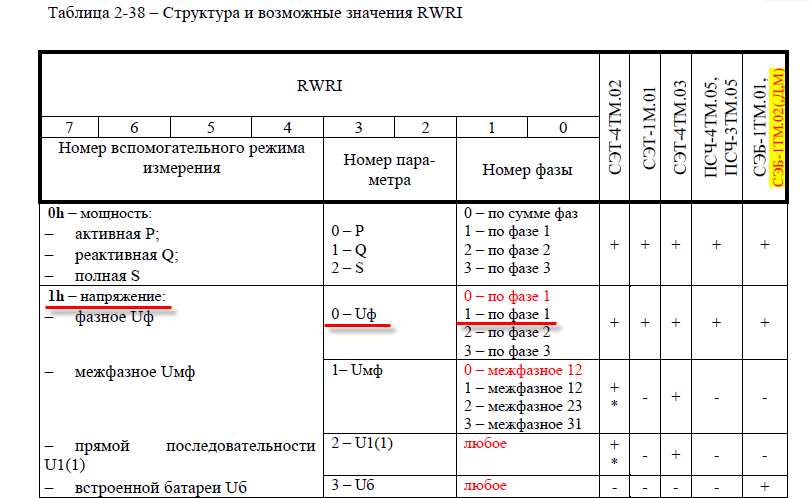


Рисунок 5.2 – Фрагмент таблицы RWRI из описания протокола

Зная структуру запроса, структуру ответа и методику расчета контрольной суммы можно начать реализацию протокола обмена в **CODESYS**.

## Алгоритмизация задачи

Процесс обмена данными через последовательный порт можно представить в виде циклически выполняемого алгоритма. На языке ST для реализации подобных алгоритмов рекомендуется использовать оператор условного выбора CASE.

Алгоритм решаемой в примере задачи можно представить следующим образом:

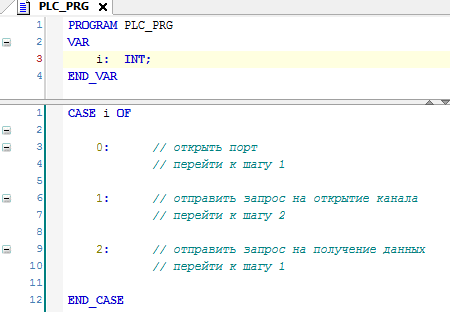


Рисунок 5.3 – Алгоритмизация задачи опроса устройства

Во время запуска программы будет однократно произведено открытие порта, после чего будет происходить циклический опрос счетчика. С целью упрощения примера, запрос на открытие канала к счетчику отправляется в каждом цикле.

Соответственно, задача сводится к написанию кода, который будет выполняться в шагах 0 и 1. Рекомендуется упаковать его в два функциональных блока:

1. ФБ управления портом, который будет вызываться на шаге 0.
2. ФБ опроса модуля, который будет вызываться на шаге 1.

Предварительно следует добавить в проект библиотеку [CAA SerialCom](#_3.1._Добавление_библиотеки), так как ее функции и ФБ потребуются при реализации протокола.

## ФБ управления портом (COM\_CONTROL)

Процесс разработки ФБ управления портом описан в [п. 4.4](#_4.4._ФБ_управления). После прочтения этого пункта следует перейти к [п. 5.5](#_ФБ_опроса_счетчика).

## ФБ опроса счетчика (SET\_4TM)

В первую очередь следует создать функциональный блок опроса счетчика **СЭТ-4ТМ.03М** по нестандартному протоколу с названием **SET\_4TM**. Для начала требуется определиться с набором входных и выходных переменных.

В рамках примера его входными переменными будут являться:

1. Вход управления блоком (старт/прекращение работы).
2. Адрес опрашиваемого устройства.
3. Дескриптор (handle) COM-порта.

К выходным переменным будут относиться:

1. Измеренное значение напряжение по фазе A (Ua).
2. Флаг окончания цикла опроса.
3. Флаги ошибок обмена.

Ниже также приводится список локальных переменных ФБ. Необходимость их объявления будет поясняться по мере описания принципов работы блока.

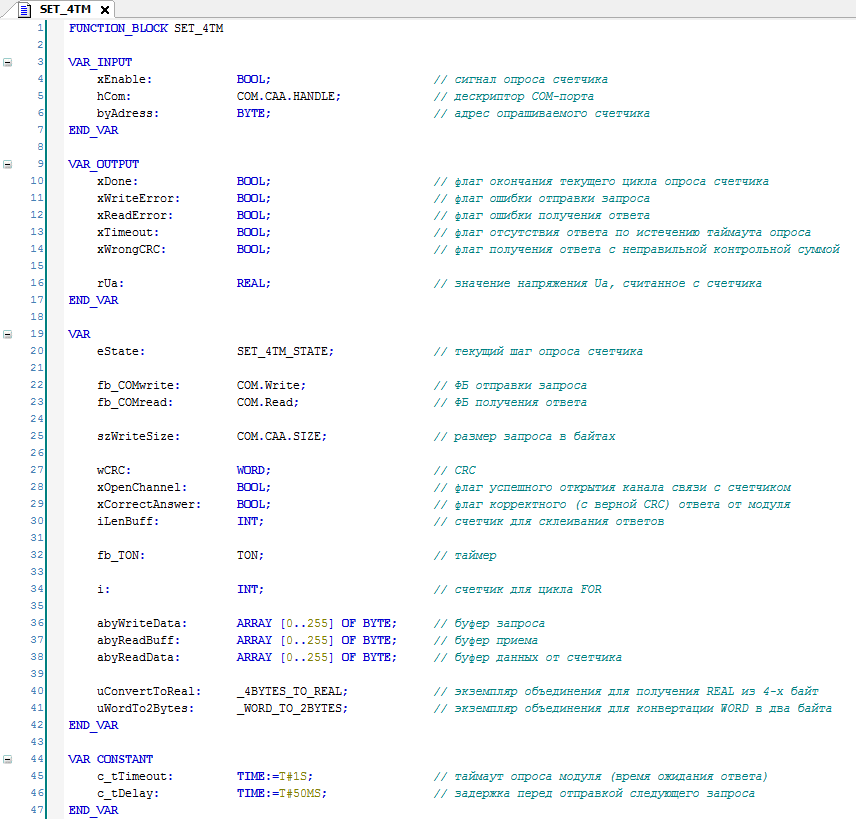


Рисунок 5.4 – Объявление переменных ФБ SET\_4TM

Для реализации блока будет использоваться пошаговый алгоритм с оператором CASE и **перечислением**, которое позволит использовать имена в качестве названий шагов. Экземпляр перечисления имеет название **eState**.

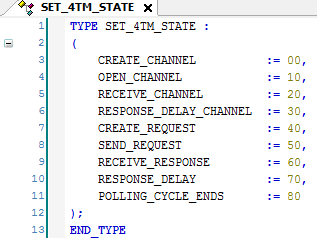


Рисунок 5.5 – Объявление перечисления SET\_4TM\_STATE

Алгоритм опроса модуля в общем виде:

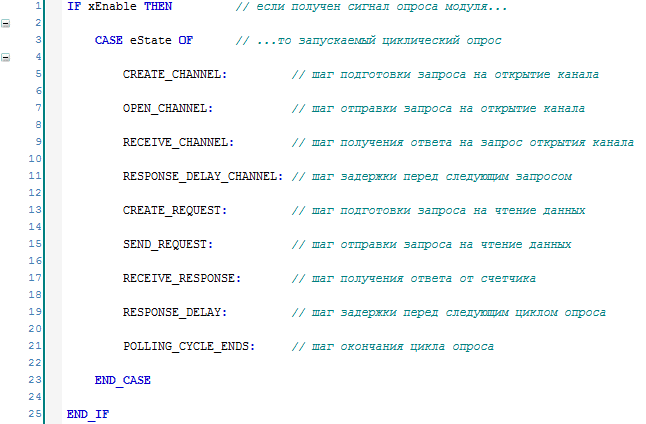


Рисунок 5.6 – Алгоритм опроса счетчика

На шаге **CREATE\_CHANNEL** будет происходить формирование запроса на открытие канала связи.

На шаге **OPEN\_CHANNEL** запрос будет отправлен в COM-порт.

На шаге **RECEIVE\_CHANNEL** будет происходить получение и анализ ответа от счетчика.

Шаг **RESPONSE\_DELAY\_CHANNEL** будет использоваться для создания задержки между получением ответа от счетчика на запрос открытия канала и отправки следующего запроса.

На шаге **CREATE\_REQUEST** будет происходить формирование запроса к счетчику.

На шаге **SEND\_REQUEST** этот запрос будет отправлен в COM-порт.

На шаге **RECEIVE\_RESPONSE** будет происходить получение и анализ ответа от счетчика.

Шаг **RESPONSE\_DELAY** будет использоваться для создания задержки между получением ответа от счетчика и отправки следующего запроса.

На шаге **POLLING\_CYCLE\_ENDS** будет происходить обновление выходов ФБ, после чего следует переход к начальному шагу.

|  |  |
| --- | --- |
|  | **ПРИМЕЧАНИЕ**  Необходимость наличия шага **RESPONSE\_DELAY** зависит от особенностей конкретного опрашиваемого устройства. Некоторые устройства удерживают линию связи фиксированное время после ответа, в связи с чем следует организовывать задержку перед отправкой следующего запроса. В случае опроса счетчика **СЭТ-4ТМ.03М** такая задержка не обязательна, но рассматривается в настоящем руководстве в качестве примера. |

Законченные фрагменты кода ФБ будут оформлены в виде **действий**:

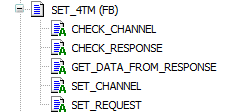


Рисунок 5.7 – Действия ФБ SET\_4TM

### Подготовка запроса на открытие канала (шаг СREATE\_CHANNEL)

На шаге **CREATE\_CHANNEL** происходит очистка буферов данных и подготовка запроса на открытие канала.

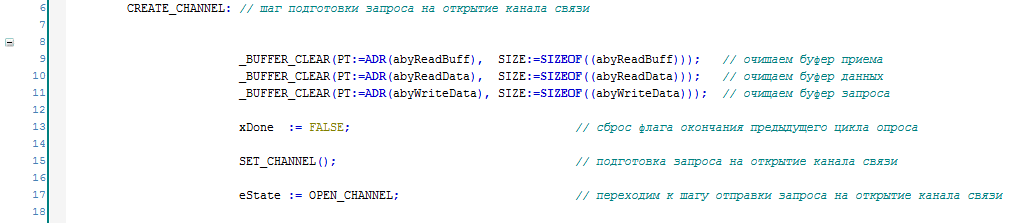


Рисунок 5.8 – Код шага CREATE\_CHANNEL

Под буфером понимается переменная, используемая для временного хранения данных. В данном случае такими переменными являются **abyReadBuff** (буфер фрагмента ответа), **abyReadData** (буфер полного ответа) и **abyWriteData** (буфер запроса). В рамках примера счетчику каждый раз отправляются одни и те же запросы, поэтому нет необходимости каждый раз очищать буфер запроса и формировать запрос заново – но в других ситуациях это может понадобиться (например, когда необходимо с помощью одного ФБ считывать несколько параметров устройства с помощью разных запросов).

Для очистки буфера используется функция **\_BUFFER\_CLEAR** из библиотеки **OSCAT**. Библиотека доступна для скачивания на сайте [oscat.de](http://www.oscat.de/) и [сайте компании Овен](http://www.owen.ua/) в разделе **CODESYS V3/Библиотеки**. Библиотека OSCAT имеет открытые исходные коды, поэтому во многих случаях удобнее копировать ее функции и ФБ в пользовательский проект (вместо добавления через **Менеджер библиотек**).

Функция **\_BUFFER\_CLEAR** имеет два входных параметра: адрес буфера и его размер. Вместо задания размера буфера с помощью числа рекомендуется использовать оператор **SIZEOF**, возвращающий размер переменной в байтах. Возвращаемое значение имеет тип **BOOL** и принимает значение **TRUE** в случае успешного завершения очистки буфера.

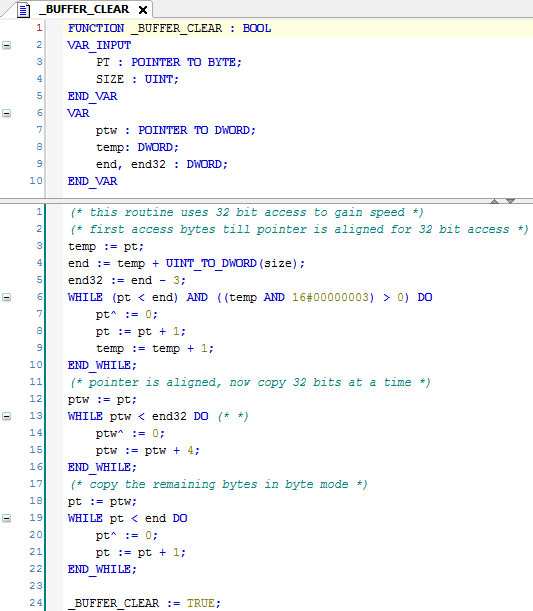


Рисунок 5.9 – Код функции \_BUFFER\_CLEAR из библиотеки OSCAT

Код функции скопирован из библиотеки **OSCAT** без каких-либо изменений.

Действие **SET\_CHANNEL** используется для формирования запроса на открытие канала. Структура запроса известна из описания протокола счетчика. Следует обратить внимание, что протокол обмена является бинарным, но пароль, определяющий уровень пользователя функции, передается в виде кодов ASCII-символов.

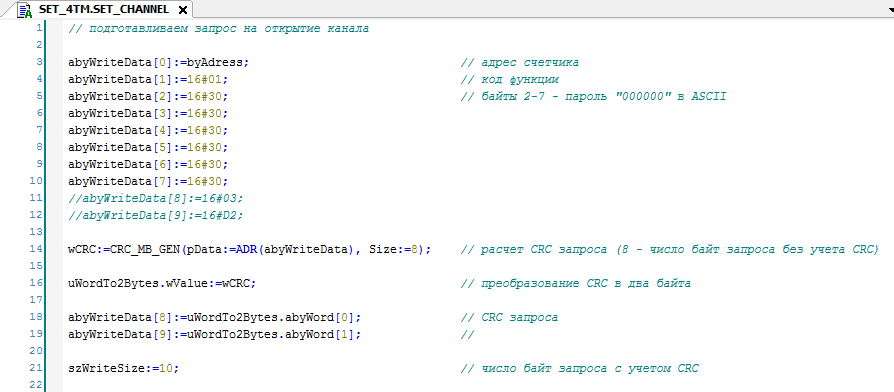


Рисунок 5.10 – Код действия SET\_CHANNEL

В протоколе счетчика используется та же методика расчета контрольной суммы, что и в протоколе Modbus – **CRC16** (циклический избыточный код). Описание алгоритма расчета можно найти в [Википедии](https://ru.wikipedia.org/wiki/%D0%A6%D0%B8%D0%BA%D0%BB%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%B7%D0%B1%D1%8B%D1%82%D0%BE%D1%87%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4), спецификации протокола Modbus, спецификации протокола счетчика и т. д. Следует создать ФБ **CRC\_MB\_GEN,** который будет выполнять эту задачу.

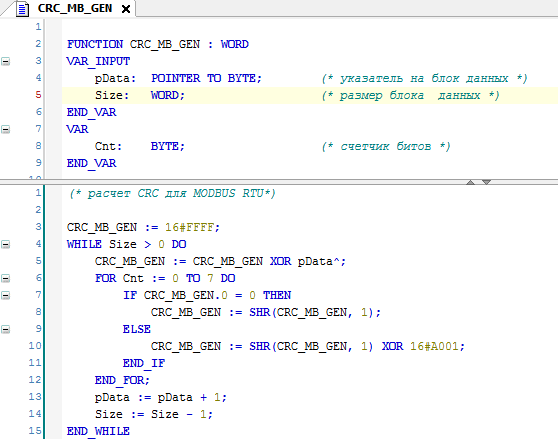


Рисунок 5.11 – Код ФБ CRC\_MB\_GEN

|  |  |
| --- | --- |
|  | **ПРИМЕЧАНИЕ**  У внимательного читателя, вероятно, возникнет вопрос – почему бы не воспользоваться готовым блоком расчета CRC из библиотеки **OSCAT**?  Да, действительно, в OSCAT есть ФБ **CRC\_GEN**, который выполняет данную задачу – но из-за особенностей реализации он не возвращает корректное значение для пакетов данных размером меньше, чем 4 байта. В нашем случае ответ от счетчика на запрос открытия канала связи занимает всего 2 байта – так что придется воспользоваться своим ФБ. |

Рассчитанная контрольная сумма представляет собой переменную типа **WORD** – но чтобы записать ее в массив, следует представить ее в виде двух байт. Для этого создается объединение **\_WORD\_TO\_2BYTES** (его экземпляр **uWordTo2Bytes** мы уже объявили в переменных **ФБ SET\_4TM**, см. рисунок 5.4).

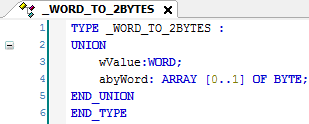


Рисунок 5.12 – Объявление объединения \_WORD\_TO\_2BYTES

Последняя операция в действии **SET\_CHANNEL** – указание числа байт запроса (переменная **szWriteSize**).

Затем следует вернуться к коду шага **CREATE\_CHANNEL**.

Кроме очистки буферов и формирования запроса на данном шаге происходит сброс флага окончания предыдущего цикла, после чего осуществляется переход к шагу [OPEN\_CHANNEL](#_5.5.2._Шаг_OPEN_CHANNEL)**.**

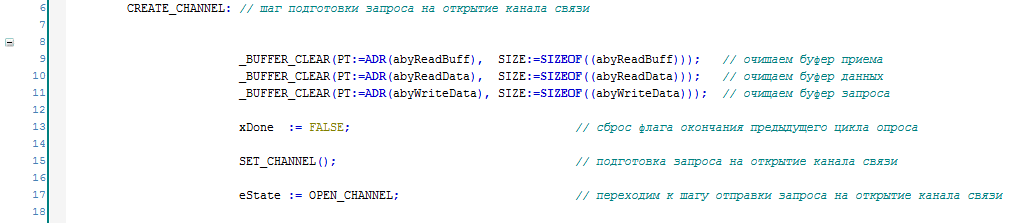


Рисунок 5.13 – Код шага CREATE\_CHANNEL

### Отправка запроса на открытие канала (шаг OPEN\_CHANNEL)

На шаге **OPEN\_CHANNEL** происходит отправка запроса, сформированного в предыдущем шаге.

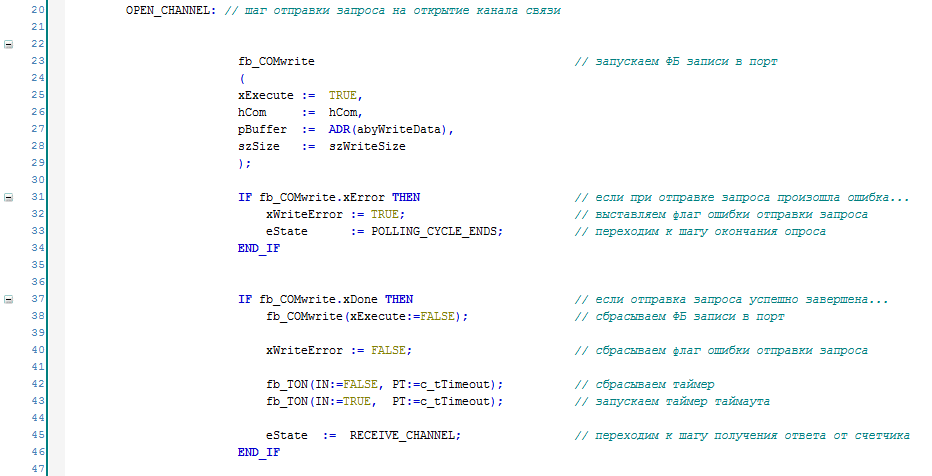


Рисунок 5.14 – Код шага SEND\_REQUEST

В данном шаге производится работа с экземпляром ФБ [COM.Write](#_3.5._ФБ_COM.Write).

Входными переменным блока являются сигнал его запуска (**xExecute**), дескриптор COM-порта (**hCom**), адрес буфера запроса (**pBuffer**) и размер буфера (**szWriteSize**). Размер буфера указывается в явном виде в коде действия **SET\_CHANNEL** (см. [рисунок 5.10](#Ris5_10)).

После вызова блока отправки запроса следует проанализировать его результат. Если во время отправки произошла ошибка (у ФБ **fb\_COMwrite** выход **xError = TRUE**), то следует взвести флаг ошибки и перейти к шагу завершения опроса.

Если запрос успешно отправлен (у **fb\_COMwrite** выход **xDone = TRUE**), то следует завершить работу блока, сбросить флаг ошибки, запустить таймер таймаута (он потребуется на следующем шаге) и перейти к шагу [RECEIVE\_CHANNEL](#_5.5.3._Шаг_RECEIVE_CHANNEL).

### Получение ответа на запрос открытия канала (шаг RECEIVE\_CHANNEL)

На шаге **RECEIVE\_СHANNEL** происходит получение ответа от счетчика на запрос открытия канала.

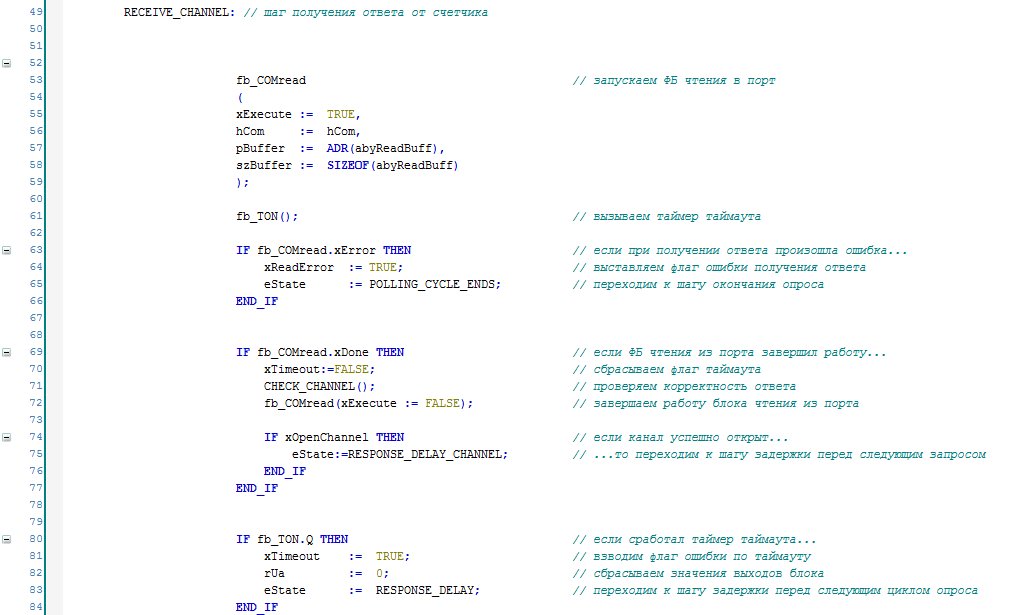


Рисунок 5.15 – Код шага RECEIVE\_CHANNEL

В данном шаге производится работа с экземпляром ФБ [COM.Read](#_3.6._ФБ_COM.Read).

Входными переменным блока являются сигнал его запуска (**xExecute**), дескриптор (handle) COM-порта (**hCom**), адрес буфера фрагмента ответа (**pBuffer**) и размер буфера (**szSize**). Вместо задания размера буфера с помощью рекомендуется использовать оператор **SIZEOF**, возвращающий размер переменной в байтах.

Следует понимать, что в ряде случаев подчиненное устройство не ответит на запрос (например, в случае некорректного запроса или неисправности устройства). Если не предусмотреть эту ситуацию, то программа можно «застрять» на шаге получения ответа. Для предотвращения данной ситуации используется контроль таймаута опроса – по истечению заданного времени осуществляется выход из текущего шага вне зависимости от факта получения ответа. Таймер таймаута был запущен в конце предыдущего шага, поэтому на данном шаге достаточно вызвать его с теми же аргументами. Время таймаута определяется с помощью константы **c\_tTimeout**.

После вызова блока получения ответа следует проанализировать его результат. Если при получении произошла ошибка (у **fb\_COMread** выход **xError = TRUE**), то следует взвести флаг ошибки и перейти к шагу завершения опроса.

Если ответ успешно получен (у **fb\_COMread** выход **xDone = TRUE**), то следует сбросить флаг ошибки таймаута, проверить корректность ответа и завершить работу блока.

Проверка корректности ответа вынесена в действие **CHECK\_CHANNEL**:

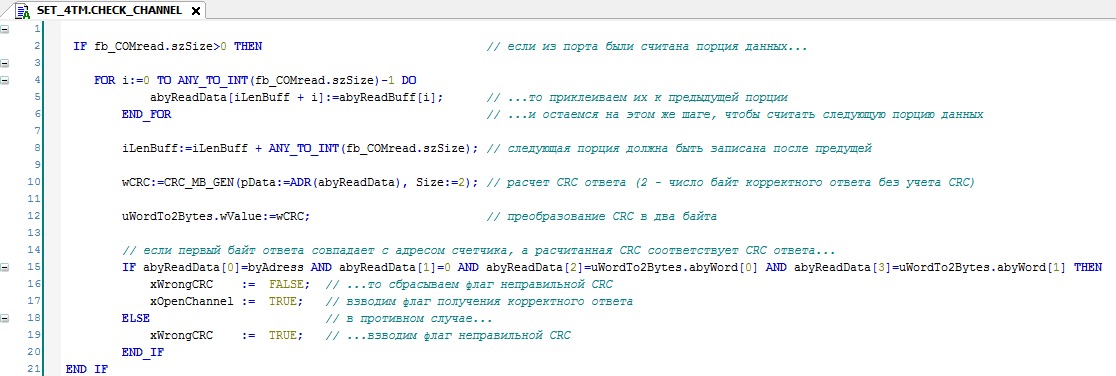


Рисунок 5.16 – Код действия CHECK\_CHANNEL

В некоторых случаях (в частности, при чтении большего количества данных), ответ от подчиненного устройства может прийти в виде нескольких фрагментов. Чтобы корректно обработать данную ситуацию следует каждый полученный фрагмент ответа «приклеивать» к предыдущему фрагменту в буфере полного ответа (**abyReadData**), управляя позицией записи в буфер с помощью переменной **iLenBuff**.

Затем следует проверить корректность ответа. Из описания протокола известно, что в случае успешного открытия канала, счетчик отвечает пакетом со следующей структурой:

|  |  |  |
| --- | --- | --- |
| **Адрес счетчика** | **16#00** | **СRC** |
| 1 байт | 1 байт | 2 байта |

Соответственно, следует проверить, что нулевой байт ответа содержит адрес счетчика, значение первого байта – 0, а во втором и третьем и байтах содержится корректная контрольная сумма. Для этого используется ФБ **CRC\_MB\_GEN** и объединение **\_WORD\_TO\_2BYTES** (см. рисунок 5.16).

Если все условия выполняются, то канал связи успешно открыт. Следует сбросить флаг некорректной CRC и взвести флаг открытия канала (**xOpenChannel**). В противном случае следует взвести флаг некорректной контрольной суммы (для упрощения не рассматривается ситуация, когда ответ содержит код ошибки и корректную контрольную сумму).

Затем следует вернуться к шагу **RECEIVE\_CHANNEL** (см. рисунок 5.17). После выполнения кода действия **CHECK\_CHANNEL** следует завершить работу ФБ **COM.Read**. Если взведен флаг корректного ответа (**xOpenChannel**), то следует к шагу [RESPONSE\_DELAY\_CHANNEL](#_5.5.4._Шаг_RESPONSE_DELAY_CHANNEL).

Если сработал таймер таймаута (основной причиной является отсутствие ответа от модуля в течение заданного интервала времени), то следует взвести флаг ошибки по таймауту, сбросить значение счетчика в 0 и перейти к шагу [RESPONSE\_DELAY](#_5.5.8._Шаг_RESPONSE_DELAY).



Рисунок 5.17 – Фрагмент кода шага RECEIVE\_CHANNEL

### Организация задержки (шаг RESPONSE\_DELAY\_CHANNEL)

На шаге **RESPONSE\_DELAY\_CHANNEL** происходит запуск таймера задержки перед следующим запросом. Задержка может понадобиться в случае опроса конкретных устройств, которые после отправки ответа блокируют линию связи на определенное время. Время задержки определяется с помощью константы **c\_tDelay**.

После запуска таймера происходит переход к шагу [CREATE\_REQUEST](#_5.5.5._Шаг_СREATE_REQUEST).

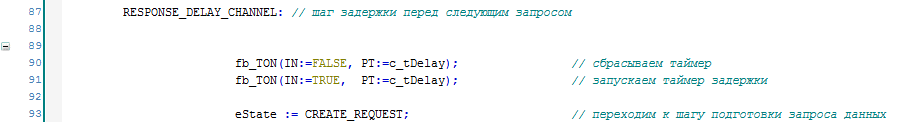


Рисунок 5.18 – Код шага RESPONSE\_DELAY\_CHANNEL

### Подготовка запроса на чтение данных (шаг СREATE\_REQUEST)

На шаге **CREATE\_REQUEST** происходит подготовка запроса на чтение данных счетчика.

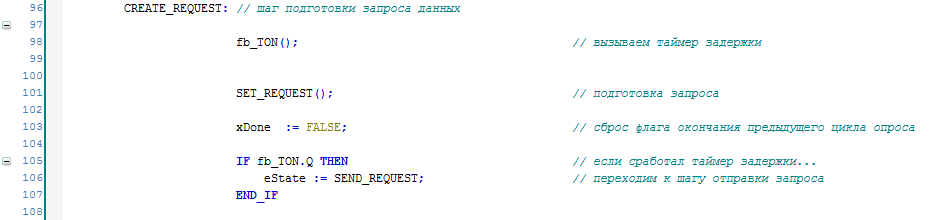


Рисунок 5.19 – Код шага CREATE\_REQUEST

Для формирования запроса используется действие **SET\_REQUEST**. Структура запроса известна из описания протокола счетчика (см. [п. 5.2](#_5.2._Описание_протокола)). Расчет контрольной суммы происходит по аналогии с шагом [CREATE\_CHANNEL](#_5.5.1._Шаг_СREATE_CHANNEL).

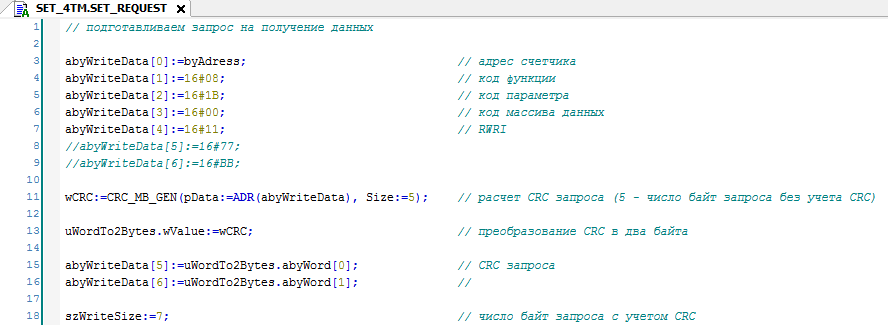


Рисунок 5.20 – Код действия SET\_REQUEST

Затем следует вернуться к коду шага **CREATE\_REQUEST**. Помимо формирования запроса, следует вызвать таймер задержки перед следующим запросом, запущенный на шаге **RESPONSE\_DELAY\_CHANNEL**. После его срабатывания следует перейти к шагу [SEND\_REQUEST](#_5.5.6._Шаг_SEND_REQUEST).

### Отправка запроса на чтение данных (шаг SEND\_REQUEST)

На шаге **SEND\_REQUEST** происходит отправка запроса, сформированного на предыдущем шаге.



Рисунок 5.21 – Код шага SEND\_REQUEST

В данном шаге производится работа с экземпляром ФБ [COM.Write](#_3.5._ФБ_COM.Write).

Входными переменным блока являются сигнал его запуска (**xExecute**), дескриптор (handle) COM-порта (**hCom**), адрес буфера запроса (**pBuffer**) и размер буфера (**szWriteSize**). Размер буфера указывается в явном виде в коде действия **SET\_REQUEST** (см. рисунок 5.20).

После вызова блока отправки запроса необходимо проанализировать его результат. Если при отправке произошла ошибка (у ФБ **fb\_COMwrite** выход **xError = TRUE**), то следует взвести флаг ошибки и перейти к шагу завершения опроса.

Если запрос успешно отправлен (у **fb\_COMwrite** выход **xDone = TRUE**), то следует завершить работу блока, сбросить флаг ошибки, запустить таймер таймаута (он потребуется на следующем шаге) и перейти к шагу [RECEIVE\_RESPONSE](#_5.5.7._Шаг_RECEIVE_RESPONSE).

На следующем шаге ожидается получение ответа от счетчика, поэтому следует обнулить переменную **iLenBuff** (она используется для склеивания фрагментов ответа от счетчика) и очистить буфер приема и буфер полученных данных (это уже производилось на шаге **CREATE\_CHANNEL** – но с того момента в буфер попал ответ счетчика на запрос открытия канала, поэтому следует очистить его повторно).

### Получение ответа (шаг RECEIVE\_RESPONSE)

На шаге **RECEIVE\_RESPONSE** происходит получение ответа от счетчика.

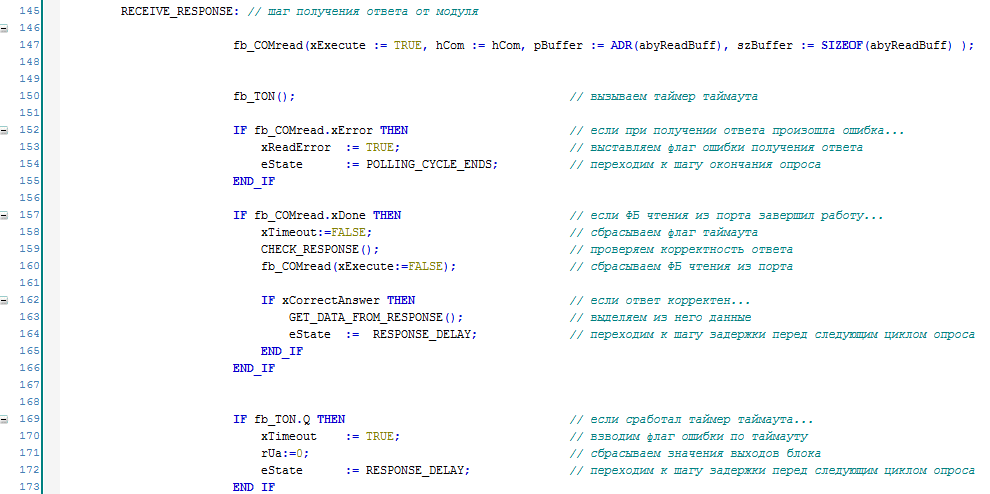


Рисунок 5.22 – Код шага RECEIVE\_RESPONSE

В данном шаге производится работа с экземпляром ФБ [COM.Read](#_3.6._ФБ_COM.Read).

Входными переменным блока являются сигнал его запуска (**xExecute**), дескриптор (handle) COM-порта (**hCom**), адрес буфера фрагмента ответа (**pBuffer**) и размер буфера (**szSize**). Вместо задания размера буфера с помощью числа рекомендуется использовать оператор **SIZEOF**, возвращающий размер переменной в байтах.

В ряде случаев подчиненное устройство не ответит на запрос (например, в случае некорректного запроса или неисправности устройства). Если не предусмотреть эту ситуацию, то программа можно «застрять» на шаге получения ответа. Для предотвращения данной ситуации используется контроль таймаута опроса – по истечению заданного времени осуществляется выход из текущего шага вне зависимости от факта получения ответа. Таймер таймаута был запущен в конце предыдущего шага, поэтому на данном шаге достаточно вызвать его с теми же аргументами. Время таймаута определяется с помощью константы **c\_tTimeout**.

После вызова блока получения ответа следует проанализировать его результат. Если при получении ответа произошла ошибка (у **fb\_COMread** выход **xError = TRUE),** то следует взвести флаг ошибки и перейти к шагу **RESPONSE\_DELAY**.

Если ответ успешно получен (у **fb\_COMread** выход **xDone = TRUE**), то следует сбросить флаг ошибки таймаута, проверить корректность ответа и завершить работу блока.

Проверка корректности ответа вынесена в действие **CHECK\_RESPONSE**:

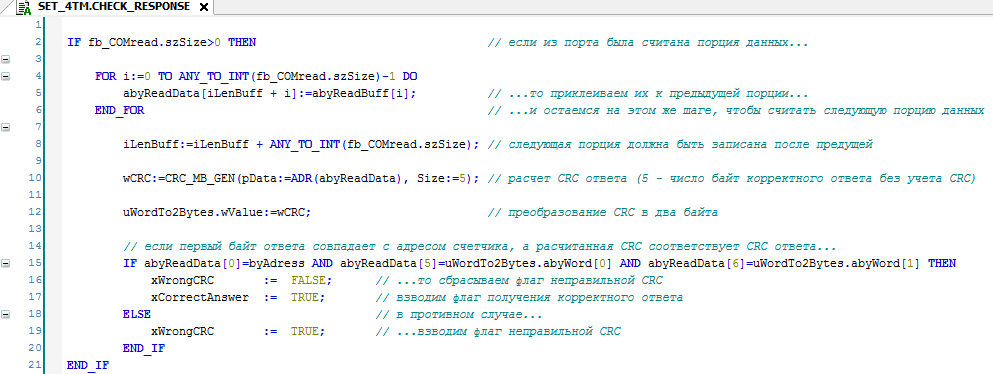


Рисунок 5.23 – Код действия CHECK\_RESPONSE

В некоторых случаях (в частности, при чтении большего количества данных), ответ от подчиненного устройства может прийти в виде нескольких фрагментов. Чтобы корректно обработать эту ситуацию, следует каждый полученный фрагмент ответа «приклеивать» к предыдущему фрагменту в буфере полного ответа (**abyReadData**), управляя позицией записи в буфер с помощью переменной **iLenBuff**.

Затем следует проверить корректность ответа. Из описания протокола известно, что в случае успешного открытия канал, счетчик отвечает пакетом со следующей структурой:

|  |  |  |
| --- | --- | --- |
| **Адрес счетчика** | **Значение Ua** | **СRC** |
| 1 байт | 4 байта | 2 байта |

Соответственно, следует проверить, что нулевой байт ответа содержит адрес счетчика, а в пятом и шестом и байтах содержится корректная CRC. Для этого используется ФБ **CRC\_MB\_GEN** и объединение **\_WORD\_TO\_2BYTES**.

Если все условия выполняются, то от счетчика получен ответ с корректной CRC. Следует сбросить флаг некорректной CRC и взвести флаг получения корректного ответа (**xCorrectAnswer**). В противном случае, следует взвести флаг некорректной CRC (для упрощения не рассматривается ситуация, когда ответ содержит код ошибки и корректную CRC).

Затем следует вернуться к коду шага **RECEIVE\_RESPONSE** (см. рисунок 5.22).

Если полученный от счетчика ответ корректен (**xCorrectAnswer=TRUE**), то можно приступать к его анализу. Анализ вынесен в действие **GET\_DATA\_FROM\_RESPONSE**.

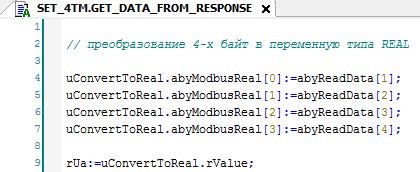


Рисунок 5.24 – Код действия GET\_DATA\_FROM\_RESPONSE

Ответ счетчика содержит измеренное напряжение по фазе A (Ua) в виде [значения с плавающей точной в формате IEEE 754](https://ru.wikipedia.org/wiki/%D0%A7%D0%B8%D1%81%D0%BB%D0%BE_%D0%BE%D0%B4%D0%B8%D0%BD%D0%B0%D1%80%D0%BD%D0%BE%D0%B9_%D1%82%D0%BE%D1%87%D0%BD%D0%BE%D1%81%D1%82%D0%B8), которое занимает 4 байта. Но в контроллере требуется отображать это значение в виде переменной типа **REAL**. Соответственно, следует преобразовать его с помощью **объединения**. Ранее объединение **\_WORD\_TO\_2BYTES** использовалось при расчете CRC. Создадим еще одно объединение с названием **\_4BYTES\_TO\_REAL**:

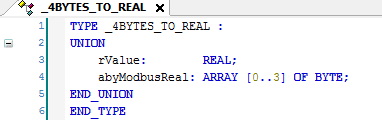


Рисунок 5.25 – Объявление объединения \_4BYTES\_TO\_REAL

Экземпляр **uConvertToReal** этого объединения уже объявлен в переменных ФБ **SET\_4TM**. Следует записать в него байты считанного значения из буфера приема (в нулевом байте содержится адрес счетчика, поэтому он не используется) и забрать из объединения значение типа **REAL**. Таким образом, в переменную **rUa** попадет измеренное значение напряжение по фазе А.

Затем следует вернуться к коду шага **RECEIVE\_RESPONSE**:

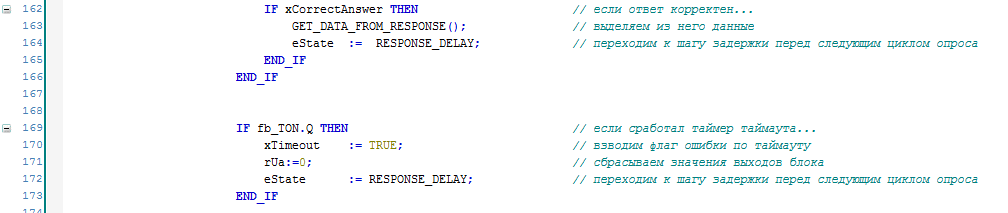


Рисунок 5.26 – Фрагмент кода шага RECEIVE\_RESPONSE

После анализа ответа следует перейти к шагу [RESPONSE\_DELAY](#_5.5.8._Шаг_RESPONSE_DELAY).

Если сработал таймер таймаута (основной причиной этого является отсутствие ответа от модуля в течение заданного интервала времени), то следует выставить флаг ошибки по таймауту, обнулить значение переменной **rUa** и перейти к шагу [RESPONSE\_DELAY](#_5.5.8._Шаг_RESPONSE_DELAY).

### Организация задержки (шаг RESPONSE\_DELAY)

На шаге **RESPONSE\_DELAY** происходит запуск таймера задержки перед следующим циклом опроса. Такая задержка может понадобиться для опроса конкретных устройств, которые после отправки ответа блокируют линию связи на определенное время. Время задержки определяется с помощью константы **c\_tDelay**.

После запуска таймера происходит переход к шагу [POLLING\_CYCLE\_ENDS](#_5.5.9._Шаг_POLLING_CYCLE_ENDS).

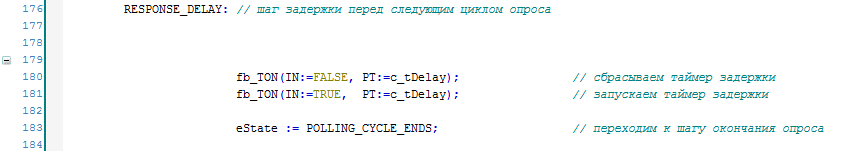


Рисунок 5.27 – Код шага RESPONSE\_DELAY

### Завершение цикла опроса (шаг POLLING\_CYCLE\_ENDS)

На шаге **POLLING\_CYCLE\_ENDS** следует вызвать таймер задержки перед следующим циклом опроса. После его срабатывания происходит завершение работы экземпляров ФБ **COM.Read** и **COM.Write**, обнуление переменной **iLenBuff,** сброс флага открытого канала, сброс флага получения корректного ответа (в следующем цикле они должны быть просчитаны заново), установка флага успешного завершения текущего цикла опроса и переход к шагу подготовки запроса на открытие канала.

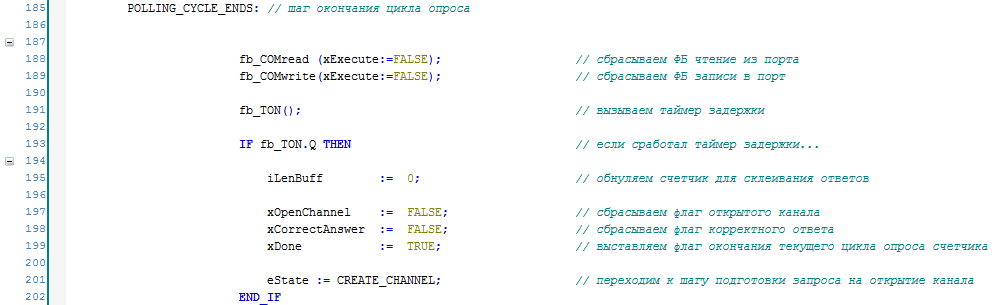


Рисунок 5.28 – Код шага POLLING\_CYCLE\_ENDS

После написания кода для всех шагов опроса модуля следует вернуться к записи алгоритма работы ФБ **SET\_4TM** и вспомнить, что весь написанный код начинает выполняться только при условии **xEnable=TRUE**. Но в процессе работы программы может возникнуть ситуация, когда обмен необходимо остановить. Соответственно, в данном случае потребуется также выполнить ряд действий (в операторе **ELSE**) – сбросить флаги завершения опроса, открытого канала и получения корректного ответа, обнулить значение выходных переменной ФБ и перейти на шаг [CREATE\_СHANNEL](#_5.5.1._Шаг_СREATE_CHANNEL) (чтобы при следующем запуске ФБ начать его выполнение с первого шага).

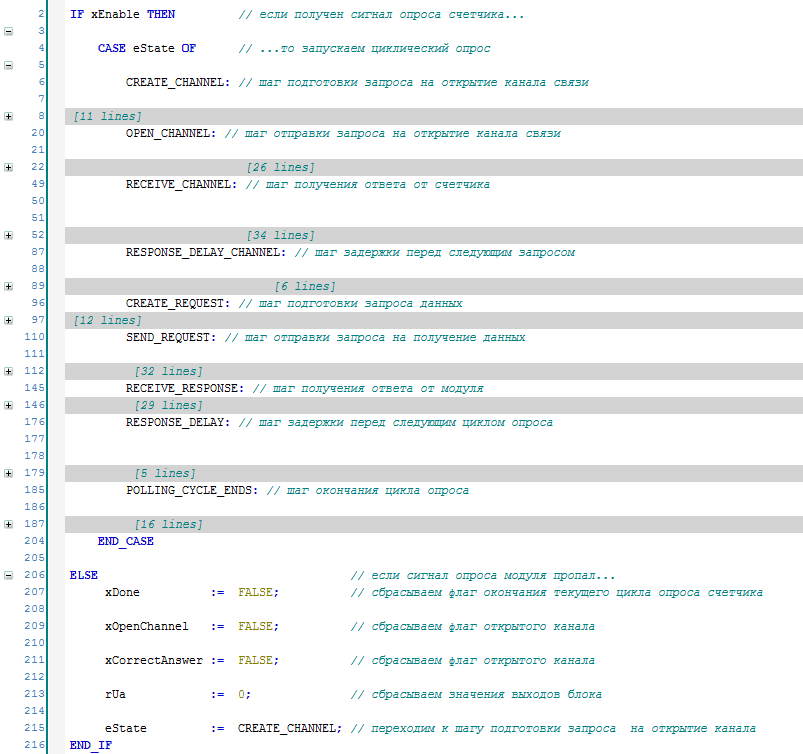


Рисунок 5.29 – Код ФБ SET\_4TM (без кода отдельных шагов)

Функциональный блок считывания измеренного напряжения фазы А с счетчика **СЭТ-4ТМ.03М** готов. Его листинг приведен в [приложении Б.2](#_Б.2._ФБ_SET_4TM).

## Программа опроса (PLC\_PRG)

Код программы **PLC\_PRG** выглядит следующим образом:

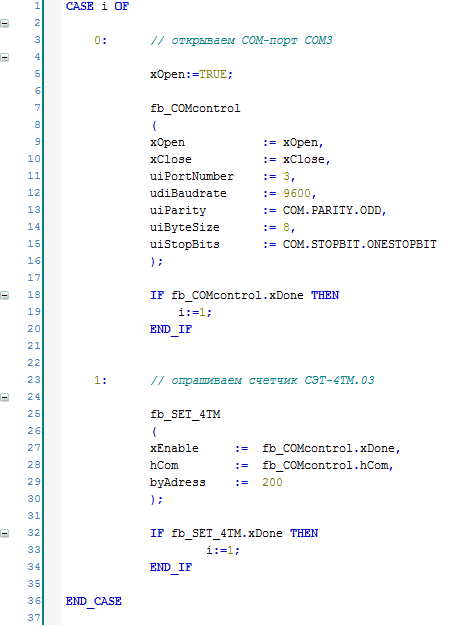


Рисунок 5.30 – Код программы PLC\_PRG

Во время запуска программы однократно выполняется код шага 0, что приводит к открытию COM-порта контроллера с заданными настройками при помощи ФБ **COM\_CONTROL**. После успешного открытия порта происходит переход на шаг 1, в котором с помощью ФБ **СЭТ-4ТМ.03М** организуется опрос модуля с заданным адресом. После окончания цикла опроса происходит переход к началу шага 1, что запускает новый цикл опроса.

Программа **PLC\_PRG** привязана к задаче с временем цикла 10 мс. Рекомендуется привязывать программы обмена к задачам с наименьшим временем цикла.

На рисунке ниже приведен скриншот программы в процессе работы.

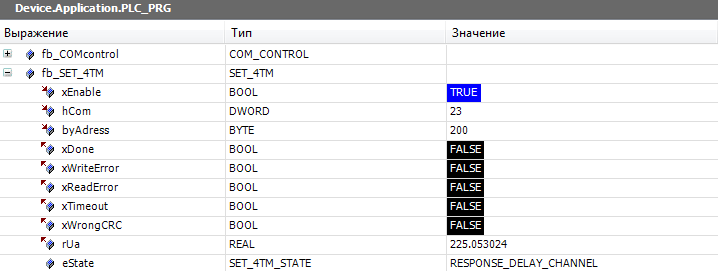


Рисунок 5.31 – Считанное значение напряжения Ua

# Рекомендации и замечания

Ниже кратко перечислены основные тезисы и рекомендации по реализации нестандартных протоколов, использованные в примерах данного документа:

1. ФБ и программы, участвующие в обмене, разбиваются на шаги, которые выполняются через оператор CASE.
2. Чтобы сделать прозрачным переходы между шагами, можно использовать перечисления.
3. Чтобы упростить отладку и повысить читабельность кода, можно выделять его законченные фрагменты в действия.
4. Переход к следующему шагу должен происходить только после окончания предыдущего. Контроль окончания шага, в частности, может осуществляться с помощью переменных **xDone** соответствующих ФБи таймеров.
5. Если возможна ситуация «застывания» на одном из шагов, необходимо использовать таймер таймаута, который позволяет выйти из шага после превышения заданного интервала времени.
6. Некоторые подчиненные устройства удерживают линию определенное время после ответа – следует предусмотреть задержку перед каждым следующим запросом.
7. После завершения цикла обмена, работа всех его ФБ должна быть завершена (обычно под этим понимается их вызов с параметром **xExecute=FALSE**).
8. В текстовых протоколах для разбора полученных данных используются функции работы со строками, в бинарных – объединения и указатели.
9. Операции чтения и записи должны выполняться в разных циклах. Утверждение справедливо и для операций открытия и закрытия порта. Реализовать разнесение операций по циклам помогает оператор CASE.

Следует также отметить ряд моментов, оставшихся за пределами примеров документа:

1. В случае необходимости опрашивать несколько устройств, следует увеличить число шагов в программе PLC\_PRG. Т. е. после опроса первого подчиненного устройства (шаг 1) должен происходить переход к опросу второго (шаг 2), потом третьего (шаг 3) и т. д. После опроса последнего устройства следует начать новый цикл опроса (с первого устройства).
2. В общем случае ФБ опроса должны быть как можно более универсальными и предоставлять пользователю возможность задавать адрес модуля, адрес и количество считываемых/записываемых регистров, используемую функцию и т. д.
3. Если описание протокола содержит объемный список кодов ошибок, то вместо битовых флагов (как в примерах) рекомендуется использовать одну WORD переменную, которая будет содержать код ошибки и перечисление с расшифровкой кодов.
4. В некоторых случаях требуется тщательная обработка ошибок обмена. Например, в случае отсутствия ответа от модуля можно сделать еще несколько попыток опроса перед переходом к опросу следующего устройства. В случае, когда ошибки обмена связаны с работой контроллера, рекомендуется закрыть COM-порт, очистить буферные переменные, сделать задержку в 100 мс и открыть порт снова.
5. Для представления пакетов данных хорошо подходят структуры – их использование увеличивает прозрачность работы программы.
6. Указатели являются эффективным средством для работы с большими объемами данных, но при некорректном использовании могут привести к утечкам памяти и зависанию программы – поэтому работа с ними подразумевает высокую квалификацию программиста и полное понимание производимых операций.

# Библиотека OwenCommunication

## Общая информация

Библиотека **OwenCommunication** содержит функциональные блоки, которые могут использоваться для реализации нестандартных протоколов в том случае, если контроллер работает в режиме Мастер по последовательному порту. Также библиотека включает в себя блоки для работы по протоколу Modbus, блоки реализации нестандартных протоколов поверх протоколов TCP/UDP и вспомогательные функции и ФБ для конвертации данных, обзор которых не является задачей данного руководства.

Библиотека доступна на сайте [ОВЕН](https://www.owen.ua/) в разделе **CODESYS V3/Библиотеки и компоненты**.

Для работы с последовательным портом используются следующие ФБ библиотеки:

* **СOM\_CONTROL** – блок управления COM-портом, по функционалу соответствует блоку **COM\_CONTROL** из [п. 4.3](#_ФБ_управления_портом);
* **UNM\_SerialRequest** – блок отправки произвольного запроса по последовательному порту и получения ответа. Включает в себя коммуникационный функционал блоков из [п. 4.4](#_ФБ_опроса_модуля) и [п. 5.5](#_ФБ_опроса_счетчика), но не содержит функционала формирования запроса и разбора ответа.

Блоки библиотеки **OwenCommunication** построены на базе ФБ библиотеки **CAA SerialCom** и предоставляют готовый функционал для реализации нестандартных протоколов. Описание блоков приведено ниже.

Пример из [п. 4](#_Пример_опроса_модуля), адаптированный с использованием библиотеки **OwenCommunication**:

[Example\_MV110\_8A\_DCON\_OCL.projectarchive](https://owen.ua/uploads/115/example_mv110_8a_dcon_ocl_3511v1.projectarchive)

Пример создан в среде **CODESYS 3.5 SP11 Patch 5** и подразумевает запуск на **СПК1хх [М01]** с таргет-файлом **3.5.11.x.** В случае необходимости запуска проекта на другом устройстве следует изменить таргет-файл в проекте (**ПКМ** на узел **Device** – **Обновить устройство).**

|  |  |
| --- | --- |
|  | **ПРИМЕЧАНИЕ**  Работа библиотеки поддерживается только на контроллерах ОВЕН и виртуальном контроллере **CODESYS Control Win V3**. |

## ФБ COM\_Control

Функциональный блок **COM\_Control** используется для открытия COM-порта с заданными настройками, а также его закрытия.

|  |  |
| --- | --- |
|  | **ПРИМЕЧАНИЕ**  Не допускается открытие уже используемого COM-порта (например, добавленного в проект с помощью стандартных средств конфигурирования). |

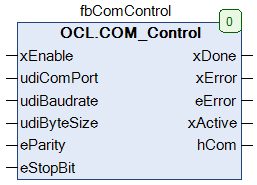


Рисунок 7.1 – Внешний вид ФБ COM\_Control на языке CFC

**Таблица 7.1 – Описание входов и выходов ФБ COM\_Control**

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип** | **Описание** |
| **Входы** | | |
| xEnable | BOOL | По переднему фронту происходит открытие COM-порта, по заднему – закрытие |
| udiComPort | UDINT | Номер COM-порта |
| udiBaudrate | UDINT | Скорость обмена в бодах. Стандартные возможные значения: **1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200** |
| udiByteSize | UDINT(7..8) | Число бит данных (**7** или **8**) |
| eParity | OCL.COM\_PARITY | Режим контроля четности |
| eStopBit | OCL.COM\_STOPBIT | Число стоп-бит |
| **Выходы** | | |
| xDone | BOOL | Принимает **TRUE** на один цикл ПЛК при успешном открытии порта |
| xError | BOOL | Принимает значение **TRUE** в случае возникновения ошибки |
| eError | OCL.ERROR | Статус работы ФБ (или код ошибки) |
| xActive | BOOL | Пока порт открыт, данный выход имеет значение **TRUE** |
| hCom | CAA.HANDLE | Дескриптор COM-порта |

## ФБ UNM\_SerialRequest

Функциональный блок **UNM\_SerialRequest** используется для реализации нестандартного протокола при обмене через COM-порт. По переднему фронту на входе **xExecute** происходит отправка содержимого буфера запроса, расположенного по указателю **pRequest**, размером **szRequest** байт через COM-порт, определяемый дескриптором **hCom**, полученным от ФБ [COM\_Control](#_ФБ_COM_Control). Ответ от slave-устройства ожидается в течение времени **tTimeout.** При получении ответа происходит его проверка на основании значений входов **szExpectedSize** и **wStopChar**:

* если **szExpectedSize** <> **0**, то ответ считается корректным, если его размер в байтах = **szExpectedSize**;
* если **szExpectedSize** = **0** и **wStopChar** <> **16#0000**, то последние один (при **wStopChar** = **16#00xx**) или два (при **wStopCha**r = **16#xxxx**) байта ответа (где x – произвольное значение) проверяются на равенство младшему или обоим байтам **wStopChar**. Это может использоваться при реализации строковых протоколов, в которых заранее известен стоп-символ;
* если **szExpectedSize** = **0** и **wStopChar** = **16#0000**, то любой полученный ответ считается корректным.

В случае получения корректного ответа выход **xDone** принимает значение **TRUE**, выход **eError** = **NO\_ERROR**, а на выходе **uiResponseSize** отображается размер ответа в байтах. Полученные данные помещается в буфер, расположенный по указателю **pResponse** и имеющий размер **szResponse** байт.

В случае отсутствия ответа ФБ повторяет запрос. Число переповторов определяется входом **usiRetry** (значение **0** соответствует отсутствию переповторов). Если ни на один из запросов не был получен ответ, то выход **xError** принимает значение **TRUE**, а выход **eError** = **TIME\_OUT**.

Для отправки нового запроса следует создать передний фронт на входе **xExecute**.

|  |  |
| --- | --- |
|  | **ПРИМЕЧАНИЕ**  В случае отправки запросов, для которых не подразумевается получение ответа, рекомендуется для входа **tTimeout** установить значение **T#1ms**. |

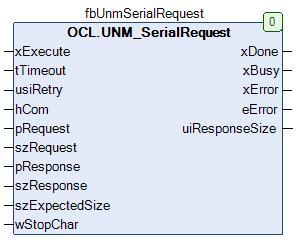


Рисунок 7.2 – Внешний вид ФБ UNM\_SerialRequest на языке CFC

**Таблица 7.2 – Описание входов и выходов ФБ UNM\_SerialRequest**

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип** | **Описание** |
| **Входы** | | |
| xExecute | BOOL | По переднему фронту происходит однократная (с возможностью переповторов в случае отсутствия ответа) отправка запроса |
| tTimeout | TIME | Таймаут ожидания ответа от slave-устройства (**T#0ms** – время ожидания не ограничено) |
| usiRetry | USINT | Число переповторов в случае отсутствия ответа |
| hCom | CAA.HANDLE | Дескриптор COM-порта, полученный от ФБ [COM\_Control](#_ФБ_COM_Control) |
| pRequest | CAA.PVOID | Указатель на буфер запроса |
| szRequest | CAA.SIZE | Размер буфера запроса в байтах |
| pResponse | CAA.PVOID | Указатель на буфер ответа |
| szResponse | CAA.SIZE | Размер буфера ответа в байтах |
| szExpectedSize | CAA.SIZE | Ожидаемый размер ответа в байтах (**0** – размер неизвестен) |
| wStopChar | WORD | Стоп-символы протокола. Для протокола с двумя стоп-символами оба байта переменной должны быть отличны от нуля. Для протокола с одним стоп-символом старший байт должен быть равен нулю, а младший быть отличным от нуля. Если в протоколе отсутствуют стоп-символы, то следует установить значение **0** |
| **Выходы** | | |
| xDone | BOOL | **TRUE** – получен корректный ответ от slave-устройства |
| xBusy | BOOL | **TRUE** – ФБ находится в работе |
| xError | BOOL | Принимает значение **TRUE** в случае возникновения ошибки |
| eError | OCL.ERROR | Статус работы ФБ (или код ошибки) |
| uiResponseSize | UINT | Размер полученного ответа в байтах |

# Листинг примера из п. 4

PROGRAM PLC\_PRG

VAR

fb\_COMcontrol: COM\_CONTROL;

fb\_MV110\_8A\_DCON: MV110\_8A\_DCON;

i: INT;

xOpen: BOOL;

xClose: BOOL;

END\_VAR

CASE i OF

0: // открываем COM-порт COM3

xOpen:=TRUE;

fb\_COMcontrol

(

xOpen := xOpen,

xClose := xClose,

uiPortNumber := 3,

udiBaudrate := 115200,

uiParity := COM.PARITY.NONE,

uiByteSize := 8,

uiStopBits := COM.STOPBIT.ONESTOPBIT

);

IF fb\_COMcontrol.xDone THEN

i:=1;

END\_IF

1: // опрашиваем модуль МВ110-8А по протоколу DCON

fb\_MV110\_8A\_DCON

(

xEnable := fb\_COMcontrol.xDone,

hCom := fb\_COMcontrol.hCom,

byAdress := 1

);

IF fb\_MV110\_8A\_DCON.xDone THEN

i:=1;

END\_IF

END\_CASE

### ФБ COM\_CONTROL

FUNCTION\_BLOCK COM\_CONTROL

VAR\_IN\_OUT

xOpen: BOOL;

xClose: BOOL;

END\_VAR

VAR\_INPUT

uiPortNumber: UINT;

udiBaudrate: DINT;

uiParity: COM.PARITY;

uiByteSize: UINT;

uiStopBits: COM.STOPBIT;

END\_VAR

VAR\_OUTPUT

xDone: BOOL;

hCom: COM.CAA.HANDLE;

xOpenError: BOOL;

xCloseError: BOOL;

END\_VAR

VAR

aComParams: ARRAY [1..5] OF COM.PARAMETER;

fb\_COMopen: COM.Open;

fb\_COMclose: COM.Close;

eState: COM\_STATE;

END\_VAR

CASE eState OF

INITIALIZE:

fb\_COMopen (xExecute:=FALSE);

fb\_COMclose (xExecute:=FALSE);

eState:=WAITING\_FOR\_SIGNAL;

WAITING\_FOR\_SIGNAL:

IF xOpen THEN

IF hCom=0 OR hCom=16#FFFFFFFF THEN

eState := OPEN\_PORT;

ELSE xOpen := FALSE; xDone := TRUE; eState := INITIALIZE;

END\_IF

END\_IF

IF xClose THEN

IF hCom>0 AND hCom<16#FFFFFFFF THEN

eState := CLOSE\_PORT; ELSE xClose := FALSE; xDone := FALSE; eState := INITIALIZE; END\_IF

END\_IF

OPEN\_PORT:

SETTINGS();

OPEN();

IF fb\_COMopen.xDone AND fb\_COMopen.xExecute THEN xOpen := FALSE;

fb\_COMopen(xExecute:=FALSE);

xDone := TRUE; eState := INITIALIZE;

END\_IF

IF fb\_COMopen.xError AND fb\_COMopen.xExecute THEN xOpenError := TRUE; fb\_COMopen(xExecute:=FALSE); xDone := FALSE; eState := INITIALIZE;

END\_IF

CLOSE\_PORT:

CLOSE();

IF fb\_COMclose.xDone AND fb\_COMclose.xExecute THEN xClose := FALSE;

xDone := FALSE;

hCom := 0;

fb\_COMclose(xExecute:=FALSE); eState := INITIALIZE; END\_IF

IF fb\_COMclose.xError AND fb\_COMclose.xExecute THEN xCloseError := TRUE;

fb\_COMclose(xExecute:=FALSE);

END\_IF

END\_CASE

#### Действие SETTINGS

aComParams[1].udiParameterId := COM.CAA\_Parameter\_Constants.udiPort;

aComParams[1].udiValue := uiPortNumber;

aComParams[2].udiParameterId := COM.CAA\_Parameter\_Constants.udiBaudrate;

aComParams[2].udiValue := udiBaudrate;

aComParams[3].udiParameterId := COM.CAA\_Parameter\_Constants.udiParity;

aComParams[3].udiValue := ANY\_TO\_UDINT(uiParity);

aComParams[4].udiParameterId := COM.CAA\_Parameter\_Constants.udiByteSize;

aComParams[4].udiValue := uiByteSize;

aComParams[5].udiParameterId := COM.CAA\_Parameter\_Constants.udiStopBits;

aComParams[5].udiValue := ANY\_TO\_UDINT(uiStopBits);

#### Действие OPEN

fb\_COMopen.usiListLength := UINT\_TO\_USINT(SIZEOF(aComParams) / SIZEOF(COM.PARAMETER));

fb\_COMopen.pParameterList := ADR(aComParams);

fb\_COMopen.xExecute := TRUE;

fb\_COMopen();

hCom := fb\_COMopen.hCom;

#### Действие CLOSE

fb\_COMclose.hCom := hCom;

fb\_COMclose.xExecute := TRUE;

fb\_COMclose();

### ФБ MV110\_8A\_DCON

FUNCTION\_BLOCK MV110\_8A\_DCON

VAR\_INPUT

xEnable: BOOL; // сигнал опроса модуля

hCom: COM.CAA.HANDLE; // дескриптор COM-порта

byAdress: BYTE; // адрес опрашиваемого модуля

END\_VAR

VAR\_OUTPUT

xDone: BOOL; // флаг окончания текущего цикла опроса модуля

xWriteError: BOOL; // флаг ошибки отправки запроса

xReadError: BOOL; // флаг ошибки получения ответа

xTimeout: BOOL; // флаг отсутствия ответа по истечению таймаута опроса

xWrongCRC: BOOL; // флаг получения ответа с неправильной контрольной суммой

rMV110\_8A\_output1: REAL; // считанные значения каналов модуля

rMV110\_8A\_output2: REAL;

rMV110\_8A\_output3: REAL;

rMV110\_8A\_output4: REAL;

rMV110\_8A\_output5: REAL;

rMV110\_8A\_output6: REAL;

rMV110\_8A\_output7: REAL;

rMV110\_8A\_output8: REAL;

END\_VAR

VAR

eState: DCON\_state; // текущий шаг опроса модуля

fb\_COMwrite: COM.Write; // ФБ отправки запроса

fb\_COMread: COM.Read; // ФБ получения ответа

sWriteData: STRING(255); // запрос к модулю в виде строки

sReadData: STRING(255); // полный ответ модуля в виде строки

sReadBuff: STRING(255); // часть ответа модуля

sAdress: STRING(2); // адрес опрашиваемого модуля в виде строки

byCRC: BYTE; // СRC

sCRC: STRING(2); // СRC в виде строки

xCorrectAnswer: BOOL; // флаг корректного (c верной CRC) ответа от модуля

i: INT; // счетчик для цикла FOR

fb\_TON: TON; // таймер

fb\_ANALYZE\_DATA: ANALYZE\_DATA; // ФБ анализа ответа модуля

END\_VAR

VAR CONSTANT

c\_tTimeout: TIME:=T#1S; // таймаут опроса модуля (время ожидания ответа)

c\_tDelay: TIME:=T#50MS; // задержка перед отправкой следующего запроса

END\_VAR

IF xEnable THEN

CASE eState OF

CREATE\_REQUEST:

\_BUFFER\_CLEAR(PT:=ADR(sReadBuff), SIZE:=SIZEOF((sReadBuff)));

\_BUFFER\_CLEAR(PT:=ADR(sReadData), SIZE:=SIZEOF((sReadData))); \_BUFFER\_CLEAR(PT:=ADR(sWriteData), SIZE:=SIZEOF((sWriteData)));

SET\_REQUEST();

xDone := FALSE;

eState := SEND\_REQUEST;

SEND\_REQUEST:

fb\_COMwrite

(

xExecute := TRUE,

hCom := hCom,

pBuffer := ADR(sWriteData),

szSize := INT\_TO\_UINT(LEN(sWriteData))

);

IF fb\_COMwrite.xError THEN

xWriteError := TRUE;

eState := POLLING\_CYCLE\_ENDS;

END\_IF

IF fb\_COMwrite.xDone THEN

fb\_COMwrite(xExecute:=FALSE);

xWriteError := FALSE;

fb\_TON(IN:=FALSE, PT:=c\_tTimeout);

fb\_TON(IN:=TRUE, PT:=c\_tTimeout);

eState := RECEIVE\_RESPONSE;

END\_IF

RECEIVE\_RESPONSE:

fb\_COMread

(

xExecute := TRUE,

hCom := hCom,

pBuffer := ADR(sReadBuff),

szBuffer := SIZEOF(sReadBuff),

);

fb\_TON();

IF fb\_COMread.xError THEN

xReadError := TRUE;

eState := POLLING\_CYCLE\_ENDS;

END\_IF

IF fb\_COMread.xDone THEN

xTimeout:=FALSE;

CHECK\_RESPONSE();

fb\_COMread(xExecute:=FALSE);

IF xCorrectAnswer THEN GET\_DATA\_FROM\_RESPONSE();

eState := RESPONSE\_DELAY; END\_IF

END\_IF

IF fb\_TON.Q THEN

xTimeout := TRUE;

OUTPUTS\_TO\_ZERO();

eState := RESPONSE\_DELAY;

END\_IF

RESPONSE\_DELAY:

fb\_TON(IN:=FALSE, PT:=c\_tDelay);

fb\_TON(IN:=TRUE, PT:=c\_tDelay);

eState := POLLING\_CYCLE\_ENDS;

POLLING\_CYCLE\_ENDS:

fb\_TON();

IF fb\_TON.Q THEN

fb\_COMread (xExecute:=FALSE);

fb\_COMwrite(xExecute:=FALSE);

xCorrectAnswer := FALSE;

xDone := TRUE;

eState := CREATE\_REQUEST;

END\_IF

END\_CASE

ELSE

xDone := FALSE;

xCorrectAnswer := FALSE;

OUTPUTS\_TO\_ZERO();

eState := CREATE\_REQUEST;

END\_IF

#### Действие SET\_REQUEST

sWriteData[0]:=16#23;

sAdress:=BYTE\_TO\_STRH(byAdress);

sWriteData[1]:=sAdress[0];

sWriteData[2]:=sAdress[1];

byCRC:=sWriteData[0]+sWriteData[1]+sWriteData[2];

sCRC:=BYTE\_TO\_STRH(byCRC);

sWriteData[3]:=sCRC[0];

sWriteData[4]:=sCRC[1];

sWriteData[5]:=16#D;

#### Действие CHECK\_RESPONSE

IF fb\_COMread.szSize>0 THEN

sReadData := CONCAT(sReadData,sReadBuff);

IF FIND(sReadData, '>') <> 0 AND FIND(sReadData, '$R') <>0 AND

(FIND(sReadData, '$R') > FIND(sReadData, '>')) THEN

sReadData:=MID(sReadData, FIND(sReadData, '$R') - FIND(sReadData, '>') + 1, FIND(sReadData, '>'));

xCorrectAnswer:=TRUE;

END\_IF

END\_IF

#### Действие GET\_DATA\_FROM\_RESPONSE

byCRC:=0;

FOR i:=0 TO FIND(sReadData, '$R') - 4 DO byCRC:=byCRC+sReadData[i];

END\_FOR

sCRC:=BYTE\_TO\_STRH(byCRC);

IF sCRC=MID(sReadData, 2, FIND(sReadData, '$R') - 2) THEN

xWrongCRC:=FALSE;

fb\_ANALYZE\_DATA(sData:=sReadData);

rMV110\_8A\_output1:=fb\_ANALYZE\_DATA.arValue[1];

rMV110\_8A\_output2:=fb\_ANALYZE\_DATA.arValue[2];

rMV110\_8A\_output3:=fb\_ANALYZE\_DATA.arValue[3];

rMV110\_8A\_output4:=fb\_ANALYZE\_DATA.arValue[4];

rMV110\_8A\_output5:=fb\_ANALYZE\_DATA.arValue[5];

rMV110\_8A\_output6:=fb\_ANALYZE\_DATA.arValue[6];

rMV110\_8A\_output7:=fb\_ANALYZE\_DATA.arValue[7];

rMV110\_8A\_output8:=fb\_ANALYZE\_DATA.arValue[8];

ELSE

xWrongCRC:=TRUE;

END\_IF

#### Действие OUTPUTS\_TO\_ZERO

rMV110\_8A\_output1:=0;

rMV110\_8A\_output2:=0;

rMV110\_8A\_output3:=0;

rMV110\_8A\_output4:=0;

rMV110\_8A\_output5:=0;

rMV110\_8A\_output6:=0;

rMV110\_8A\_output7:=0;

rMV110\_8A\_output8:=0;

### ФБ ANALYZE\_DATA

FUNCTION\_BLOCK ANALYZE\_DATA

VAR\_INPUT

sData: STRING;

END\_VAR

VAR CONSTANT

c\_iMaxCanal: INT := 8;

END\_VAR

VAR\_OUTPUT

arValue: ARRAY [1..c\_iMaxCanal] OF REAL;

END\_VAR

VAR

aiSignPos: ARRAY [1..c\_iMaxCanal+1] OF INT;

i,j: INT;

END\_VAR

j:=1;

FOR i:=1 TO FIND(sData,'$R') DO

IF sData[i]=16#2B OR sData[i]=16#2D OR sData[i]=16#0D THEN

aiSignPos[j]:=i;

j:=j+1;

END\_IF

END\_FOR

FOR i:=1 TO c\_iMaxCanal-1 DO

arValue[i]:= STRING\_TO\_REAL(MID(sData, aiSignPos[i+1] - aiSignPos[i], aiSignPos[i] + 1));

END\_FOR

arValue[c\_iMaxCanal]:=STRING\_TO\_REAL(

MID(sData, aiSignPos[c\_iMaxCanal+1] - aiSignPos[c\_iMaxCanal] - 2, aiSignPos[i] + 1));

### Функция BYTE\_TO\_STRH

FUNCTION BYTE\_TO\_STRH : STRING(2)

VAR\_INPUT

IN : BYTE;

END\_VAR

VAR

temp : BYTE;

PT : POINTER TO BYTE;

END\_VAR

PT := ADR(BYTE\_TO\_STRH);

temp := SHR(in,4);

IF temp <= 9 THEN temp := temp + 48; ELSE temp := temp + 55; END\_IF;

PT^ := temp;

temp := in AND 2#00001111;

IF temp <= 9 THEN temp := temp + 48; ELSE temp := temp + 55; END\_IF;

pt := pt + 1;

pt^ := temp;

pt := pt + 1;

pt^:= 0;

### Функция \_BUFFER\_CLEAR

FUNCTION \_BUFFER\_CLEAR : BOOL

VAR\_INPUT

PT : POINTER TO BYTE;

SIZE : UINT;

END\_VAR

VAR

ptw : POINTER TO DWORD;

temp: DWORD;

end, end32 : DWORD;

END\_VAR

temp := pt;

end := temp + UINT\_TO\_DWORD(size);

end32 := end - 3;

WHILE (pt < end) AND ((temp AND 16#00000003) > 0) DO

pt^ := 0;

pt := pt + 1;

temp := temp + 1;

END\_WHILE;

ptw := pt;

WHILE ptw < end32 DO

ptw^ := 0;

ptw := ptw + 4;

END\_WHILE;

pt := ptw;

WHILE pt < end DO

pt^ := 0;

pt := pt + 1;

END\_WHILE;

\_BUFFER\_CLEAR := TRUE;

### Перечисление COM\_STATE

TYPE COM\_STATE :

(

INITIALIZE := 00,

WAITING\_FOR\_SIGNAL := 10,

OPEN\_PORT := 20,

CLOSE\_PORT := 30

);

END\_TYPE

### Перечисление DCON\_STATE

TYPE DCON\_STATE :

(

CREATE\_REQUEST := 00,

SEND\_REQUEST := 10,

RECEIVE\_RESPONSE := 20,

RESPONSE\_DELAY := 30,

POLLING\_CYCLE\_ENDS := 40

);

END\_TYPE

# Листинг примера из п. 5

PROGRAM PLC\_PRG

VAR

fb\_COMcontrol: COM\_CONTROL;

fb\_SET\_4TM: SET\_4TM;

i: INT;

xOpen: BOOL;

xClose: BOOL;

END\_VAR

CASE i OF

0: // открываем COM-порт COM3

xOpen:=TRUE;

fb\_COMcontrol

(

xOpen := xOpen,

xClose := xClose,

uiPortNumber := 3,

udiBaudrate := 9600,

uiParity := COM.PARITY.ODD,

uiByteSize := 8,

uiStopBits := COM.STOPBIT.ONESTOPBIT

);

IF fb\_COMcontrol.xDone THEN

i:=1;

END\_IF

1: // опрашиваем счетчик СЭТ-4ТМ.03

fb\_SET\_4TM

(

xEnable := fb\_COMcontrol.xDone,

hCom := fb\_COMcontrol.hCom,

byAdress := 200

);

IF fb\_SET\_4TM.xDone THEN

i:=1;

END\_IF

END\_CASE

### ФБ COM\_CONTROL

FUNCTION\_BLOCK COM\_CONTROL

VAR\_IN\_OUT

xOpen: BOOL; // сигнал открытия порта

xClose: BOOL; // сигнал закрытия порта

END\_VAR

VAR\_INPUT

uiPortNumber: UINT; // номер порта

udiBaudrate: DINT; // скорость передачи данных

uiParity: COM.PARITY; // четность

uiByteSize: UINT; // кол-во бит данных в байте

uiStopBits: COM.STOPBIT; // кол-во стоп бит

END\_VAR

VAR\_OUTPUT

xDone: BOOL; // флаг успешного открытия порта

hCom: COM.CAA.HANDLE; // дескриптор порта

xOpenError: BOOL; // флаг ошибки открытия порта

xCloseError: BOOL; // флаг ошибки закрытия порта

END\_VAR

VAR

aComParams: ARRAY [1..5] OF COM.PARAMETER; // структура настроек порта

fb\_COMopen: COM.Open; // ФБ открытия порта

fb\_COMclose: COM.Close; // ФБ закрытия порта

eState: COM\_STATE; // текущий шаг работы с портом

END\_VAR

CASE eState OF

INITIALIZE:

fb\_COMopen (xExecute:=FALSE);

fb\_COMclose (xExecute:=FALSE);

eState:=WAITING\_FOR\_SIGNAL;

WAITING\_FOR\_SIGNAL:

IF xOpen THEN

IF hCom=0 OR hCom=16#FFFFFFFF THEN

eState := OPEN\_PORT;

ELSE xOpen := FALSE; xDone := TRUE; eState := INITIALIZE;

END\_IF

END\_IF

IF xClose THEN

IF hCom>0 AND hCom<16#FFFFFFFF THEN

eState := CLOSE\_PORT; ELSE xClose := FALSE; xDone := FALSE; eState := INITIALIZE; END\_IF

END\_IF

OPEN\_PORT:

SETTINGS();

OPEN();

IF fb\_COMopen.xDone AND fb\_COMopen.xExecute THEN xOpen := FALSE;

fb\_COMopen(xExecute:=FALSE);

xDone := TRUE; eState := INITIALIZE;

END\_IF

IF fb\_COMopen.xError AND fb\_COMopen.xExecute THEN xOpenError := TRUE; fb\_COMopen(xExecute:=FALSE); xDone := FALSE; eState := INITIALIZE;

END\_IF

CLOSE\_PORT:

CLOSE();

IF fb\_COMclose.xDone AND fb\_COMclose.xExecute THEN xClose := FALSE;

xDone := FALSE;

hCom := 0;

fb\_COMclose(xExecute:=FALSE); eState := INITIALIZE; END\_IF

IF fb\_COMclose.xError AND fb\_COMclose.xExecute THEN xCloseError := TRUE;

fb\_COMclose(xExecute:=FALSE);

END\_IF

END\_CASE

#### Действие SETTINGS

aComParams[1].udiParameterId := COM.CAA\_Parameter\_Constants.udiPort;

aComParams[1].udiValue := uiPortNumber;

aComParams[2].udiParameterId := COM.CAA\_Parameter\_Constants.udiBaudrate;

aComParams[2].udiValue := udiBaudrate;

aComParams[3].udiParameterId := COM.CAA\_Parameter\_Constants.udiParity;

aComParams[3].udiValue := ANY\_TO\_UDINT(uiParity);

aComParams[4].udiParameterId := COM.CAA\_Parameter\_Constants.udiByteSize;

aComParams[4].udiValue := uiByteSize;

aComParams[5].udiParameterId := COM.CAA\_Parameter\_Constants.udiStopBits;

aComParams[5].udiValue := ANY\_TO\_UDINT(uiStopBits);

#### Действие OPEN

fb\_COMopen.usiListLength := UINT\_TO\_USINT(SIZEOF(aComParams) / SIZEOF(COM.PARAMETER));

fb\_COMopen.pParameterList := ADR(aComParams);

fb\_COMopen.xExecute := TRUE;

fb\_COMopen();

hCom := fb\_COMopen.hCom;

#### Действие CLOSE

fb\_COMclose.hCom := hCom;

fb\_COMclose.xExecute := TRUE;

fb\_COMclose();

### ФБ SET\_4TM

FUNCTION\_BLOCK SET\_4TM

VAR\_INPUT

xEnable: BOOL;

hCom: COM.CAA.HANDLE;

byAdress: BYTE;

END\_VAR

VAR\_OUTPUT

xDone: BOOL;

xWriteError: BOOL;

xReadError: BOOL;

xTimeout: BOOL;

xWrongCRC: BOOL;

rUa: REAL;

END\_VAR

VAR

eState: SET\_4TM\_STATE;

fb\_COMwrite: COM.Write;

fb\_COMread: COM.Read;

szWriteSize: COM.CAA.SIZE;

wCRC: WORD;

xOpenChannel: BOOL;

xCorrectAnswer: BOOL;

iLenBuff: INT;

fb\_TON: TON;

i: INT;

abyWriteData: ARRAY [0..255] OF BYTE;

abyReadBuff: ARRAY [0..255] OF BYTE;

abyReadData: ARRAY [0..255] OF BYTE;

uConvertToReal: \_4BYTES\_TO\_REAL;

uWordTo2Bytes: \_WORD\_TO\_2BYTES;

END\_VAR

VAR CONSTANT

c\_tTimeout: TIME:=T#1S;

c\_tDelay: TIME:=T#50MS;

END\_VAR

IF xEnable THEN

CASE eState OF

CREATE\_CHANNEL:

\_BUFFER\_CLEAR(PT:=ADR(abyReadBuff), SIZE:=SIZEOF((abyReadBuff)));

\_BUFFER\_CLEAR(PT:=ADR(abyReadData), SIZE:=SIZEOF((abyReadData)));

\_BUFFER\_CLEAR(PT:=ADR(abyWriteData), SIZE:=SIZEOF((abyWriteData)));

xDone := FALSE;

SET\_CHANNEL();

eState := OPEN\_CHANNEL;

OPEN\_CHANNEL: // шаг отправки запроса на открытие канала связи

fb\_COMwrite

(

xExecute := TRUE,

hCom := hCom,

pBuffer := ADR(abyWriteData),

szSize := szWriteSize

);

IF fb\_COMwrite.xError THEN

xWriteError := TRUE;

eState := POLLING\_CYCLE\_ENDS;

END\_IF

IF fb\_COMwrite.xDone THEN

fb\_COMwrite(xExecute:=FALSE);

xWriteError := FALSE;

fb\_TON(IN:=FALSE, PT:=c\_tTimeout);

fb\_TON(IN:=TRUE, PT:=c\_tTimeout);

eState := RECEIVE\_CHANNEL; END\_IF

RECEIVE\_CHANNEL:

fb\_COMread

(

xExecute := TRUE,

hCom := hCom,

pBuffer := ADR(abyReadBuff),

szBuffer := SIZEOF(abyReadBuff)

);

fb\_TON();

IF fb\_COMread.xError THEN

xReadError := TRUE;

eState := POLLING\_CYCLE\_ENDS;

END\_IF

IF fb\_COMread.xDone THEN

xTimeout:=FALSE;

CHECK\_CHANNEL();

fb\_COMread(xExecute := FALSE);

IF xOpenChannel THEN

eState:=RESPONSE\_DELAY\_CHANNEL;

END\_IF

END\_IF

IF fb\_TON.Q THEN

xTimeout := TRUE;

rUa := 0; eState := RESPONSE\_DELAY;

END\_IF

RESPONSE\_DELAY\_CHANNEL:

fb\_TON(IN:=FALSE, PT:=c\_tDelay);

fb\_TON(IN:=TRUE, PT:=c\_tDelay);

eState := CREATE\_REQUEST;

CREATE\_REQUEST:

fb\_TON();

SET\_REQUEST();

xDone := FALSE;

IF fb\_TON.Q THEN

eState := SEND\_REQUEST;

END\_IF

SEND\_REQUEST:

fb\_COMwrite

(

xExecute := TRUE,

hCom := hCom,

pBuffer := ADR(abyWriteData),

szSize := szWriteSize

);

IF fb\_COMwrite.xError THEN

xWriteError := TRUE;

eState := POLLING\_CYCLE\_ENDS;

END\_IF

IF fb\_COMwrite.xDone THEN

fb\_COMwrite(xExecute:=FALSE);

xWriteError := FALSE;

fb\_TON(IN:=FALSE, PT:=c\_tTimeout);

fb\_TON(IN:=TRUE, PT:=c\_tTimeout);

eState := RECEIVE\_RESPONSE;

END\_IF

iLenBuff := 0;

\_BUFFER\_CLEAR(PT:=ADR(abyReadBuff), SIZE:=SIZEOF((abyReadBuff))); \_BUFFER\_CLEAR(PT:=ADR(abyReadData), SIZE:=SIZEOF((abyReadData)));

RECEIVE\_RESPONSE: // шаг получения ответа от модуля

fb\_COMread

(

xExecute := TRUE,

hCom := hCom,

pBuffer := ADR(abyReadBuff),

szBuffer := SIZEOF(abyReadBuff)

);

fb\_TON();

IF fb\_COMread.xError THEN

xReadError := TRUE;

eState := POLLING\_CYCLE\_ENDS;

END\_IF

IF fb\_COMread.xDone THEN xTimeout:=FALSE;

CHECK\_RESPONSE();

fb\_COMread(xExecute:=FALSE);

IF xCorrectAnswer THEN

GET\_DATA\_FROM\_RESPONSE();

eState := RESPONSE\_DELAY;

END\_IF

END\_IF

IF fb\_TON.Q THEN

xTimeout := TRUE;

rUa:=0;

eState := RESPONSE\_DELAY;

END\_IF

RESPONSE\_DELAY:

fb\_TON(IN:=FALSE, PT:=c\_tDelay);

fb\_TON(IN:=TRUE, PT:=c\_tDelay);

eState := POLLING\_CYCLE\_ENDS;

POLLING\_CYCLE\_ENDS:

fb\_COMread (xExecute:=FALSE);

fb\_COMwrite(xExecute:=FALSE);

fb\_TON();

IF fb\_TON.Q THEN

iLenBuff := 0;

xOpenChannel := FALSE;

xCorrectAnswer := FALSE;

xDone := TRUE;

eState := CREATE\_CHANNEL;

END\_IF

END\_CASE

ELSE

xDone := FALSE;

xOpenChannel := FALSE;

xCorrectAnswer := FALSE;

rUa := 0;

eState := CREATE\_CHANNEL;

END\_IF

#### Действие SET\_CHANNEL

abyWriteData[0]:=byAdress; abyWriteData[1]:=16#01;

abyWriteData[2]:=16#30; abyWriteData[3]:=16#30;

abyWriteData[4]:=16#30;

abyWriteData[5]:=16#30;

abyWriteData[6]:=16#30;

abyWriteData[7]:=16#30;

wCRC:=CRC\_MB\_GEN(pData:=ADR(abyWriteData), Size:=8);

uWordTo2Bytes.wValue:=wCRC;

abyWriteData[8]:=uWordTo2Bytes.abyWord[0];

abyWriteData[9]:=uWordTo2Bytes.abyWord[1];

szWriteSize:=10;

#### Действие CHECK\_CHANNEL

IF fb\_COMread.szSize>0 THEN

FOR i:=0 TO ANY\_TO\_INT(fb\_COMread.szSize)-1 DO

abyReadData[iLenBuff + i]:=abyReadBuff[i];

END\_FOR

iLenBuff:=iLenBuff + ANY\_TO\_INT(fb\_COMread.szSize);

wCRC:=CRC\_MB\_GEN(pData:=ADR(abyReadData), Size:=2);

uWordTo2Bytes.wValue:=wCRC;

IF abyReadData[0]=byAdress AND abyReadData[1]=0 AND abyReadData[2]=uWordTo2Bytes.abyWord[0] AND abyReadData[3]=uWordTo2Bytes.abyWord[1] THEN

xWrongCR := FALSE;

xOpenChannel := TRUE;

ELSE

xWrongCRC := TRUE;

END\_IF

END\_IF

#### Действие SET\_REQUEST

abyWriteData[0]:=byAdress;

abyWriteData[1]:=16#08;

abyWriteData[2]:=16#1B;

abyWriteData[3]:=16#00;

abyWriteData[4]:=16#11;

wCRC:=CRC\_MB\_GEN(pData:=ADR(abyWriteData), Size:=5);

uWordTo2Bytes.wValue:=wCRC;

abyWriteData[5]:=uWordTo2Bytes.abyWord[0];

abyWriteData[6]:=uWordTo2Bytes.abyWord[1];

szWriteSize:=7;

#### Действие CHECK\_RESPONSE

IF fb\_COMread.szSize>0 THEN

FOR i:=0 TO ANY\_TO\_INT(fb\_COMread.szSize)-1 DO

abyReadData[iLenBuff + i]:=abyReadBuff[i];

END\_FOR

iLenBuff:=iLenBuff + ANY\_TO\_INT(fb\_COMread.szSize);

wCRC:=CRC\_MB\_GEN(pData:=ADR(abyReadData), Size:=5);

uWordTo2Bytes.wValue:=wCRC;

IF abyReadData[0]=byAdress AND abyReadData[5]=uWordTo2Bytes.abyWord[0] AND abyReadData[6]=uWordTo2Bytes.abyWord[1] THEN

xWrongCRC := FALSE;

xCorrectAnswer := TRUE;

ELSE

xWrongCRC := TRUE;

END\_IF

END\_IF

#### Действие GET\_DATA\_FROM\_RESPONSE

uConvertToReal.abyModbusReal[0]:=abyReadData[1];

uConvertToReal.abyModbusReal[1]:=abyReadData[2];

uConvertToReal.abyModbusReal[2]:=abyReadData[3];

uConvertToReal.abyModbusReal[3]:=abyReadData[4];

rUa:=uConvertToReal.rValue;

### Функция CRC\_MG\_GEN

FUNCTION CRC\_MB\_GEN : WORD

VAR\_INPUT

pData: POINTER TO BYTE;

Size: WORD;

END\_VAR

VAR

Cnt: BYTE;

END\_VAR

CRC\_MB\_GEN := 16#FFFF;

WHILE Size > 0 DO

CRC\_MB\_GEN := CRC\_MB\_GEN XOR pData^;

FOR Cnt := 0 TO 7 DO

IF CRC\_MB\_GEN.0 = 0 THEN

CRC\_MB\_GEN := SHR(CRC\_MB\_GEN, 1);

ELSE

CRC\_MB\_GEN := SHR(CRC\_MB\_GEN, 1) XOR 16#A001;

END\_IF

END\_FOR;

pData := pData + 1;

Size := Size - 1;

END\_WHILE

### Функция \_BUFFER\_CLEAR

FUNCTION \_BUFFER\_CLEAR : BOOL

VAR\_INPUT

PT : POINTER TO BYTE;

SIZE : UINT;

END\_VAR

VAR

ptw : POINTER TO DWORD;

temp: DWORD;

end, end32 : DWORD;

END\_VAR

temp := pt;

end := temp + UINT\_TO\_DWORD(size);

end32 := end - 3;

WHILE (pt < end) AND ((temp AND 16#00000003) > 0) DO

pt^ := 0;

pt := pt + 1;

temp := temp + 1;

END\_WHILE;

ptw := pt;

WHILE ptw < end32 DO

ptw^ := 0;

ptw := ptw + 4;

END\_WHILE;

pt := ptw;

WHILE pt < end DO

pt^ := 0;

pt := pt + 1;

END\_WHILE;

\_BUFFER\_CLEAR := TRUE;

### Перечисление COM\_STATE

TYPE COM\_STATE :

(

INITIALIZE := 00,

WAITING\_FOR\_SIGNAL := 10,

OPEN\_PORT := 20,

CLOSE\_PORT := 30

);

END\_TYPE

### Перечисление SET\_4TM\_STATE

TYPE SET\_4TM\_STATE :

(

CREATE\_CHANNEL := 00,

OPEN\_CHANNEL := 10,

RECEIVE\_CHANNEL := 20,

RESPONSE\_DELAY\_CHANNEL := 30,

CREATE\_REQUEST := 40,

SEND\_REQUEST := 50,

RECEIVE\_RESPONSE := 60,

RESPONSE\_DELAY := 70,

POLLING\_CYCLE\_ENDS := 80

);

END\_TYPE