CODESYS V3.5

Настройка обмена по протоколу Modbus

Руководство пользователя

24.05.2022

версия 3.0

Оглавление

1 Цель документа. Способы работы с Modbus в CODESYS V3.5

Настоящее руководство описывает настройку обмена данными по протоколу **Modbus** для контроллеров ОВЕН, программируемых в среде **CODESYS V3.5**. Подразумевается, что читатель обладает базовыми навыками работы с **CODESYS**, поэтому общие вопросы (например, создание и загрузка проектов) в данном документе не рассматриваются – они подробно описаны в документах **CODESYS V3.5. Первый старт** и **CODESYS V3.5. FAQ**, которые доступны на сайте ОВЕН в разделе CODESYS V3/Документация. Документ соответствует версии CODESYS **3.5.17.3** и выше.

В зависимости от квалификации и потребностей пользователя имеется возможность выбрать наиболее подходящий для него способ организации связи по протоколу **Modbus**:

1. Для начинающих пользователей, работающих с модулями Mx110 и Mx210 – **шаблоны модулей**. Шаблоны представляют собой сконфигурированные компоненты **CODESYS**, добавляемые в проект несколькими кликами мыши, для которых следует указать только сетевой адрес модуля.

Преимущества	Ограничения
Простота использования	Строго заданная конфигурация регистров без возможности редактирования
Быстрое создание проекта	
Нет необходимости в дополнительном программировании	
Не надо разбираться с картами регистров модулей	

2. Для обычных пользователей – **стандартные средства конфигурирования CODESYS**.
С их помощью можно достаточно просто настроить обмен с любым устройством.

Преимущества	Ограничения
Возможность создания произвольной конфигурации	Требуются знания спецификации протокола Modbus
Возможность выбора регистров и функций для опроса	
Ручная настройка таймаутов и задержек для обеспечения корректной работы специфических устройств	Ограниченные возможности по переконфигурированию и управлению обменом из кода
Нет необходимости в дополнительном программировании	Ограничения по числу опрашиваемых устройств, каналов и т. д.

3. Для профессионалов – библиотека **OwenCommunication.**
Библиотека позволяет настроить обмен с любым устройством, но, в отличие от стандартных средств (пп. 2), лишена их ограничений и предоставляет дополнительный функционал.

Преимущества	Ограничения
Практически неограниченные возможности для работы с протоколом Modbus	Требуются хорошие навыки программирования и знание спецификации протокола Modbus
Создание любых конфигураций и методов опроса устройств	
Удобно при программировании модульных систем (когда в разные промежутки времени в работе находится разное оборудование)	Библиотека поддерживается только на контроллерах ОВЕН (затрудняется процесс переноса ПО на другие устройства)

2 Общие сведения

2.1 Основные сведения об интерфейсе RS-485

1. Интерфейс RS-485 подразумевает использование исключительно топологии типа «шина» (топологии типа «звезда» и «кольцо» не описаны в стандарте).
2. В сети может присутствовать только одно master-устройство, которое отсылает запросы и принимает ответы от подчиненных slave-устройств. Slave-устройства не могут являться инициаторами обмена.
3. Число slave-устройств в сегменте сети не должно превышать **32**. Сегменты могут быть соединены повторителями (например, ОВЕН АС5), но следует учитывать что так как опрос всех устройств происходит последовательно, то время одного полного цикла опроса может значительно увеличиться. Общее ограничение числа slave-устройств в сети для протокола Modbus – **247**.
4. На первом и последнем устройстве в сети рекомендуется устанавливать согласующий резистор (терминатор) с сопротивлением **120 Ом**.
5. Для линий связи RS-485 необходимо использовать экранированный кабель с витой парой, предназначенный для промышленного интерфейса RS-485 с волновым сопротивлением **120 Ом** (например, КИПЭВ). Экран кабеля должен быть соединен с функциональной землей только в одной точке.
6. Некоторые устройства имеют встроенные резисторы подтяжки интерфейса RS-485. Информация и рекомендации по их использованию приведены в руководстве по эксплуатации на соответствующие приборы.

2.2 Основные сведения о протоколе Modbus

Modbus – открытый коммуникационный протокол, основанный на архитектуре **Master-Slave** (ведущий-ведомый). Спецификация протокола доступна на сайте Modbus Organization.

Master (мастер, ведущее устройство) является инициатором обмена и может считывать и записывать данные в slave-устройства.

Slave (слэйв, подчиненное устройство) отвечает на запросы master-устройства, но не может самостоятельно инициировать обмен.

Существуют две основные реализации протокола:

1. **Modbus Serial** для передачи данных с использованием последовательных интерфейсов RS-232/RS-485;
2. **Modbus TCP** для передачи данных через сети TCP/IP.

Modbus Serial имеет два режима передачи данных:

1. **Modbus RTU** (передача данных в двоичном виде);
2. **Modbus ASCII** (передача данных в виде ASCII символов).

В случае использования протокола **Modbus** поверх интерфейса **RS-232/RS-485** в сети может присутствовать **только одно** master-устройство и несколько (до **247**) slave-устройств. Адрес **0** используется для широковещательной рассылки (команд записи, которую получат все slave-устройства).

В сети **Modbus TCP** нет явного ограничения на количество master- и slave-устройств. Кроме того, устройство может одновременно выполнять функции master и slave. В сети могут также существовать специальные шлюзы (**gateway**) для объединения сетей **Modbus Serial** и **Modbus TCP**.

Запрос master-устройства к slave-устройству содержит:

- **Slave ID** (адрес slave-устройства);
- **Код функции,** применяемой к slave-устройству;
- **Данные** – адрес первого регистра и их количество (в случае записи – также записываемые значения).
- Контрольную сумму.

Ответ slave-устройства имеет схожую структуру.

Запрос master-устройства представляет собой обращение к одной из **областей памяти** slave-устройства с помощью определенной **функции**. **Область памяти** характеризуется типом хранящихся в ней значений (биты/регистры) и типом доступа (только чтение/чтение и запись). Стандарт Modbus определяет 4 области памяти:

Таблица 2.1 – Области данных протокола Modbus

Область данных	Обозначение	Тип данных	Тип доступа
Coils (Регистры флагов)	0x	BOOL	Чтение/запись
Discrete Inputs (Дискретные входы)	1x	BOOL	Только чтение
Input Registers (Регистры ввода)	3x	WORD	Только чтение
Holding Registers (Регистры хранения)	4x	WORD	Чтение/запись

Каждая область памяти состоит из определенного (зависящего от конкретного устройства) количества ячеек. Каждая ячейка имеет уникальный адрес. Для конфигурируемых устройств (таких как ТРМ, ПЧВ и т. д.) производитель предоставляет **карту регистров**, в которой содержится информация об адресах и типах параметров устройства. Для программируемых устройств пользователь формирует такую карту самостоятельно с помощью среды разработки. Существуют устройства, в которых сочетаются оба рассмотренных случая – у их карты регистров есть фиксированная часть, которую пользователь может дополнить в соответствии со своей задачей (но адреса ячеек не должны пересекаться).

ПРИМЕЧАНИЕ

В некоторых устройствах области памяти наложены друг на друга (например, **0x** и **4x**) – т. е. пользователь может обращаться разными функциями к одним и тем же ячейкам памяти.

Функция определяет операцию (чтение/запись) и область памяти, с которой эта операция будет произведена. Ниже приведен список наиболее часто используемых функций:

Таблица 2.2 – Основные функции протокола Modbus

Код функции	Имя функции	Выполняемая команда
1 (0x01)	Read Coil Status	Чтение значений из регистров флагов
2 (0x02)	Read Discrete Inputs	Чтение значений из дискретных входов
3 (0x03)	Read Holding Registers	Чтение значений из регистров хранения
4 (0x04)	Read Input Registers	Чтение значений из регистров ввода
5 (0x05)	Write Single Coil	Запись значения в один регистр флага
6 (0x06)	Write Single Register	Запись значения в один регистр хранения
15 (0x0F)	Write Multiple Coils	Запись значений в несколько регистров флагов
16 (0x10)	Write Multiple Registers	Запись значений в несколько регистров хранения

ПРИМЕЧАНИЕ

Нельзя смешивать понятия области памяти и функции. У начинающих пользователей часто возникают проблемы при работе с **input** и **holding** регистрами, поскольку **область памяти** holding регистров имеет обозначение **4x**, а **функция чтения** holding регистров – **0x03** (может интуитивно показаться, что идентификатор области памяти и код функции должны совпадать – но на практике это не так).

Ниже приведен фрагмент карты регистров для модуля аналогового ввода ОВЕН МВ110-8А. В ней для каждого параметра указан адрес и тип данных (тип данных определяет число ячеек памяти, занимаемых параметром). В таблице не упомянуто, в какой области памяти расположены параметры – но в примечании указано, что обращаться к ним необходимо функциями **0x03** и **0x04** – значит области памяти **4x** и **3x** в устройстве наложены друг на друга.

Окончание таблицы В.4

Параметр	Тип	Адрес регистра	
		(hex)	(dec)
Положение десятичной точки в целом значении для входа 2 (знач. **DP**)	int16	0006	6
Целое значение измерение входа 2 со смещением точки	int16	0007	7
Статус измерения входа 2 (код исключительной ситуации)	int16	0008	8
Циклическое время измерения входа 2	int16	0009	9
Измерение входа 2 в представлении с плавающей точкой	Float32	000A,000B	10,11
...			
Положение десятичной точки в целом значении для входа 8 (знач. **DP**)	int16	002A	42
Целое значение измерение входа 8 со смещением точки	int16	002B	43
Статус измерения входа 8 (код исключительной ситуации)	int16	002C	44
Циклическое время измерения входа 8	int16	002D	45
Измерение входа 8 в представлении с плавающей точкой	Float32	002E,002F	46,47

Примечания

1 Все регистры только на чтение, чтение регистров осуществляется командами 03 или 04 (прибор поддерживает обе команды).

2 При передаче 4-х байтных значений (тип Float 32) старшее слово передается в регистре с меньшим номером.

Рисунок 2.1 – Фрагмент карты регистров модуля МВ110-8А

В различных документах идентичные обозначения могут иметь разный смысл в зависимости от контекста. Например, префикс **0x** часто используют как указание на шестнадцатеричную систему счисления, поэтому в одном случае **0x30** может обозначать «30-й бит области памяти **coils**», а в другом – «адрес 30 в шестнадцатеричной (HEX) системе счисления» (такой адрес может относиться к любой области памяти).

Другой пример необходимости уточнения контекста – принцип адресации регистров. В некоторых случаях в адреса битов/регистров закладывается префикс области памяти, в которых они находятся, например – **30101** (цифра **3** указывает на **input регистры**), **40202** (цифра **4** указывает на **holding регистры**). Обычно подразумевается, что адрес **30001** соответствует **нулевому input регистру**, а **40001** – **нулевому holding регистру**. То есть при опросе упомянутых регистров (**30101** и **40202**) в настройках master-устройства следует указать **input регистр** с адресом **100** и **holding регистр** с адресом **201**.

В то же время существуют устройства, для которых адрес **40202** может являться адресом любой области памяти (например, **input регистр** номер **40202**).

Система обозначений для адресов битов/регистров slave-устройств зависит от конкретного производителя (в некоторых случаях – даже от конкретного документа), в связи с чем от пользователя требуется четкое понимание контекста используемых обозначений и повышенное внимание к примечаниям и сноскам.

Ниже приведен пример различных вариантов обозначений для **holding регистра** с адресом **39**:

- регистр **4x39**
- регистр **39**, чтение осуществляется функцией **03**
- регистр **0x27**, чтение осуществляется функцией **0x03**
- регистр **40040**

Запрос к slave-устройству может быть **одиночным** или **групповым**. В случае **одиночных запросов** master-устройство считывает каждый из параметров slave-устройства отдельной командой. В случае **группового опроса** master-устройство считывает одной командой сразу несколько параметров, адреса которых в карте регистров расположены строго последовательно и не имеют разрывов. Групповой опрос позволяет уменьшить трафик в сети и время, затрачиваемое на опрос устройства, но в некоторых случаях его применение невозможно (или возможно с ограничениями) из-за индивидуальных особенностей устройства.

2.3 Нумерация COM-портов в CODESYS

Во время настройки интерфейсов RS-232/RS-485 в **CODESYS** следует указывать номера портов. Для контроллеров ОВЕН эти номера приведены в таргет-файле устройства в узле **Device** на вкладке **Информация**:

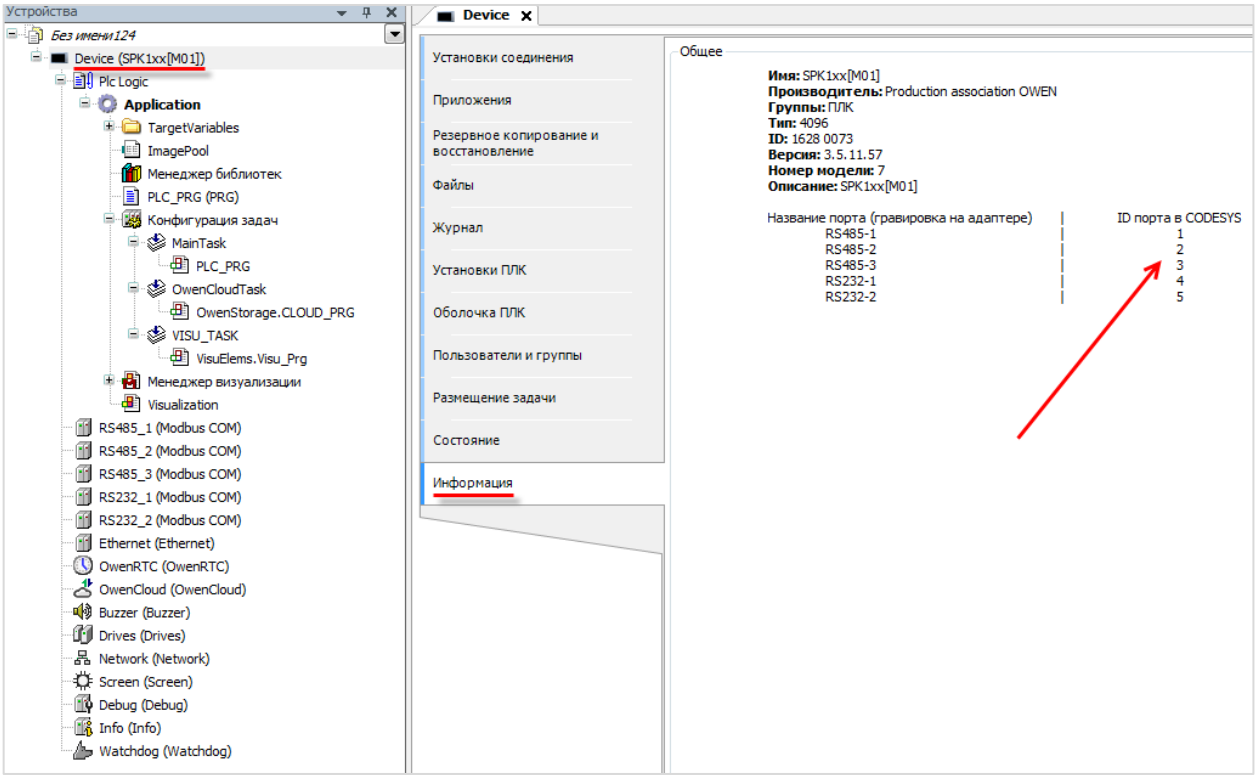


Рисунок 2.2 – Информация о нумерации COM-портов в таргет-файле

2.4 Особенности работы с модулями Mx110

Перед тем, как подключать модули Mx110 к контроллеру, их следует сконфигурировать с помощью программы **Конфигуратор Mx110**. Программа доступна на сайте ОВЕН на странице любого из модулей.

Для подключения к модулю следует указать его сетевые настройки. Если настройки неизвестны, то необходимо сбросить настройки на заводские (процесс сброса описан в руководстве по эксплуатации на модуль) и подключиться с помощью кнопки **Заводские сетевые настройки**.

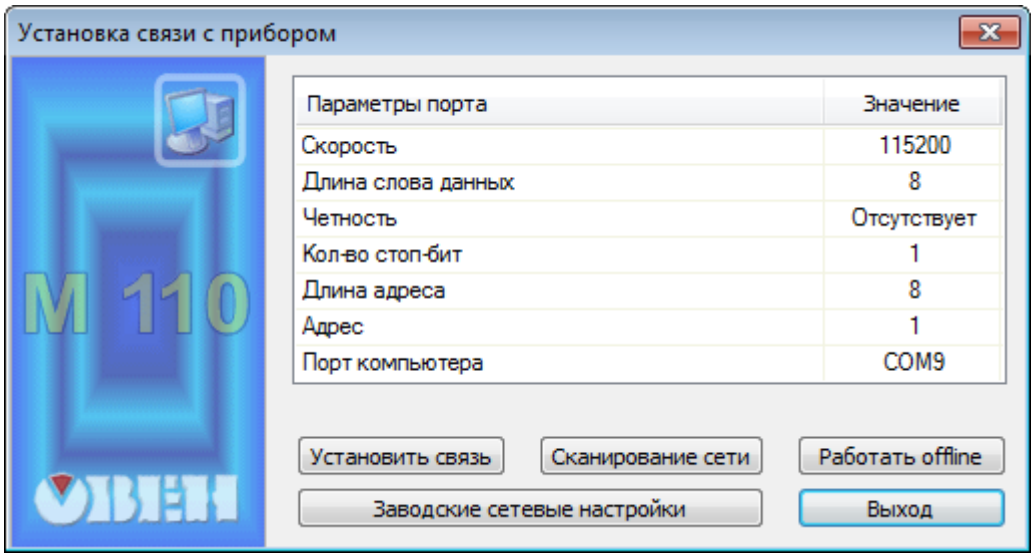


Рисунок 2.3 – Подключение к модулю Mx110 с помощью программы Конфигуратор Mx110

В конфигураторе задаются сетевые настройки модулей и параметры входов/выходов.

ПРИМЕЧАНИЕ
В пределах одной сети все модули должны иметь одинаковые сетевые настройки, за исключением адресов.

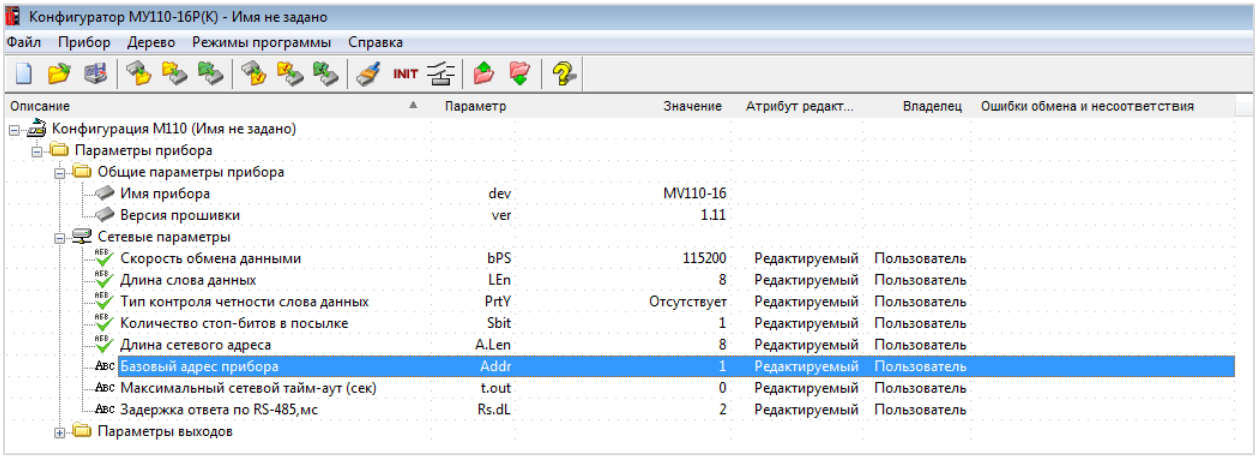


Рисунок 2.4 – Настройка модуля с помощью конфигуратора

Конфигурирование модулей происходит по протоколу **ОВЕН**. В связи с особенностями протокола во время конфигурирования каждый модуль занимает количество адресов, равное количеству его каналов. То есть в случае попытке настраивать модули, уже находящиеся в сети и имеющие

последовательные адреса (1, 2, 3), могут возникнуть ошибки. Если предполагается, что в будущем может потребоваться перенастройка модулей, то следует изначально задавать адреса модулей с промежутками, равными числу каналов в модулях.

Например, для связки МВ110-8А – МВ110-16Д – МВ110-16Р можно выбрать адреса 1 – 9 – 25.

После изменения настроек модуля через **Конфигуратор Mx110** следует перезагрузить его по питанию.

3 Шаблоны модулей Mx110 и Mx210

3.1 Установка шаблонов модулей в среду CODESYS

Шаблоны представляют собой уже сконфигурированные slave-устройства с фиксированными картами регистров. Их настройка крайне проста и сводится только к выбору адреса модуля и привязки переменных к нужным каналам. Компания ОВЕН предоставляет шаблоны для устройств с интерфейсом RS-485 (модулей Mx110, ПЧВ, ТРМ, датчиков с RS-485 и др.) и модулей Mx210.

Для работы с шаблонами требуется установить в среду программирования соответствующий пакет. В настоящем руководстве описывается работа с шаблонами версии **3.5.11.x**. Работа с более старыми версиями шаблонов описана в предыдущих версиях руководства.

Пакеты доступны на сайте компании ОВЕН в разделе CODESYS V3/Библиотеки и компоненты.

Информация о влиянии использования шаблонов на производительность контроллера: для шаблонов Mx110, для шаблонов Mx210

Для установки пакета в **CODESYS** в меню **Инструменты** следует выбрать пункт **Менеджер пакетов**, после чего указать путь к файлу пакета и нажать **Установить**.

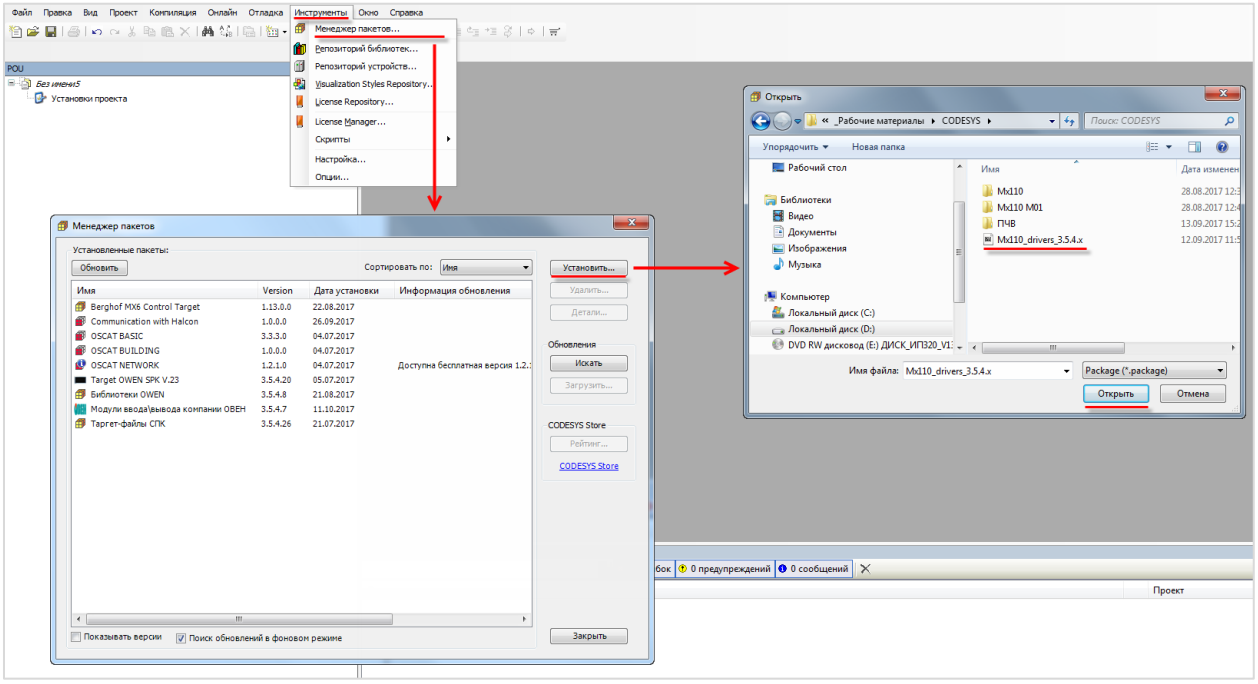


Рисунок 3.1.1 – Установка пакета шаблонов в среду CODESYS

В появившемся диалоговом окне следует выбрать пункт **Полная установка**, после чего нажать кнопку **Next**:

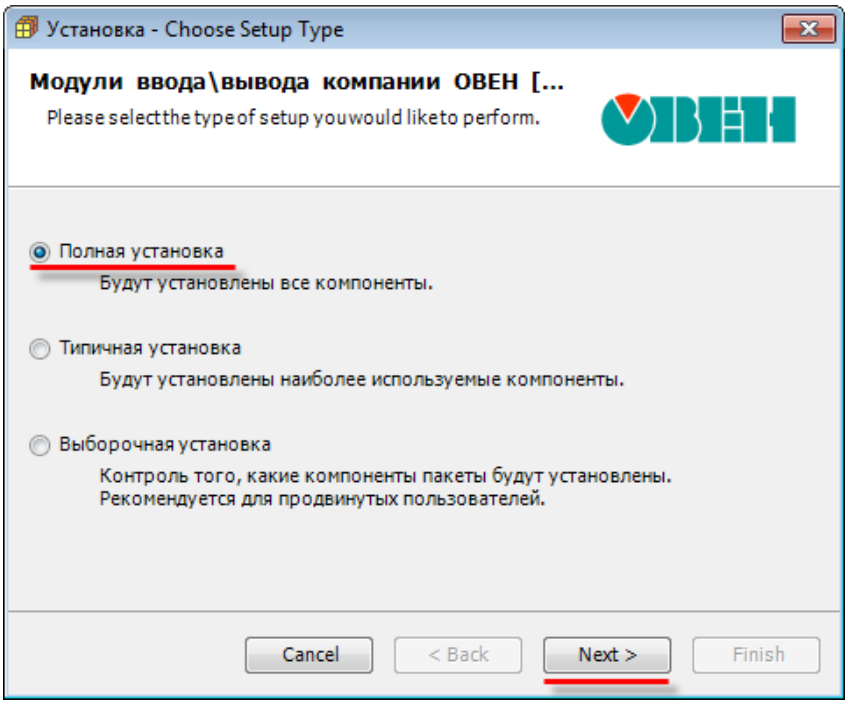


Рисунок 3.1.2 – Начало установки шаблонов модулей

После завершения установки следует закрыть диалоговое окно с помощью кнопки **Finish**:

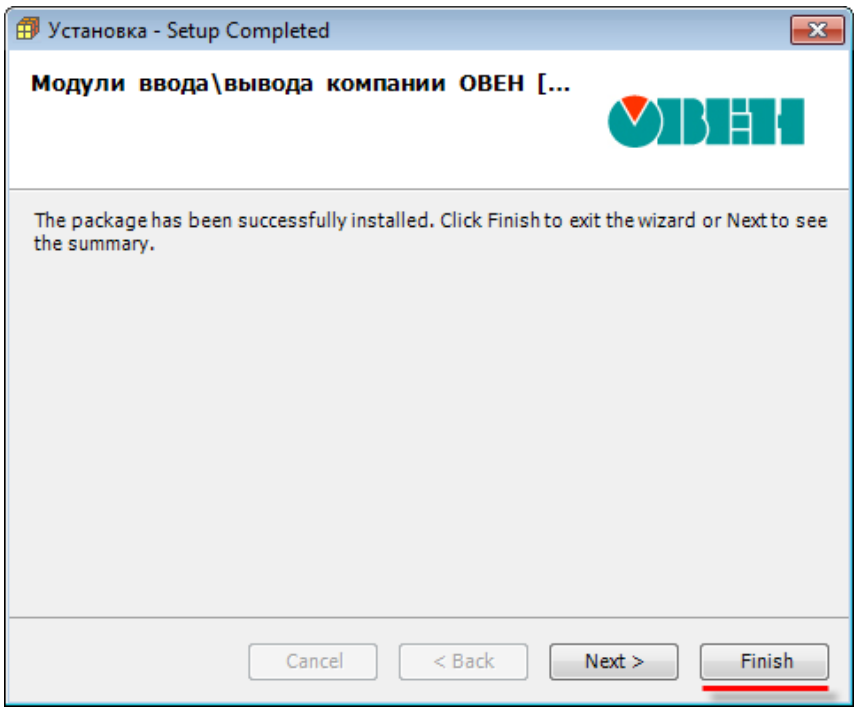


Рисунок 3.1.3 – Завершение установки шаблонов модулей

3.2 Пример: СПК1xx [M01] + модули Mx110

В качестве примера будет рассмотрена настройка обмена с модулями Mx110 (МВ110-8А, МВ110-16Д, МУ110-16Р) с использованием **шаблонов**. В примере используются шаблоны версии **3.5.11.8**.

Реализуемый алгоритм: если значение первого аналогового входа модуля **МВ110-8А** превышает **30** и при этом первый дискретный вход модуля **МВ110-16Д** имеет значение **TRUE** (замкнут), то первому дискретному выходу модуля **МУ110-16Р** присваивается значение **TRUE** (замкнут). Во всех остальных случаях дискретному выходу присваивается значение **FALSE** (разомкнут).

Структурная схема примера приведена на рисунке ниже:

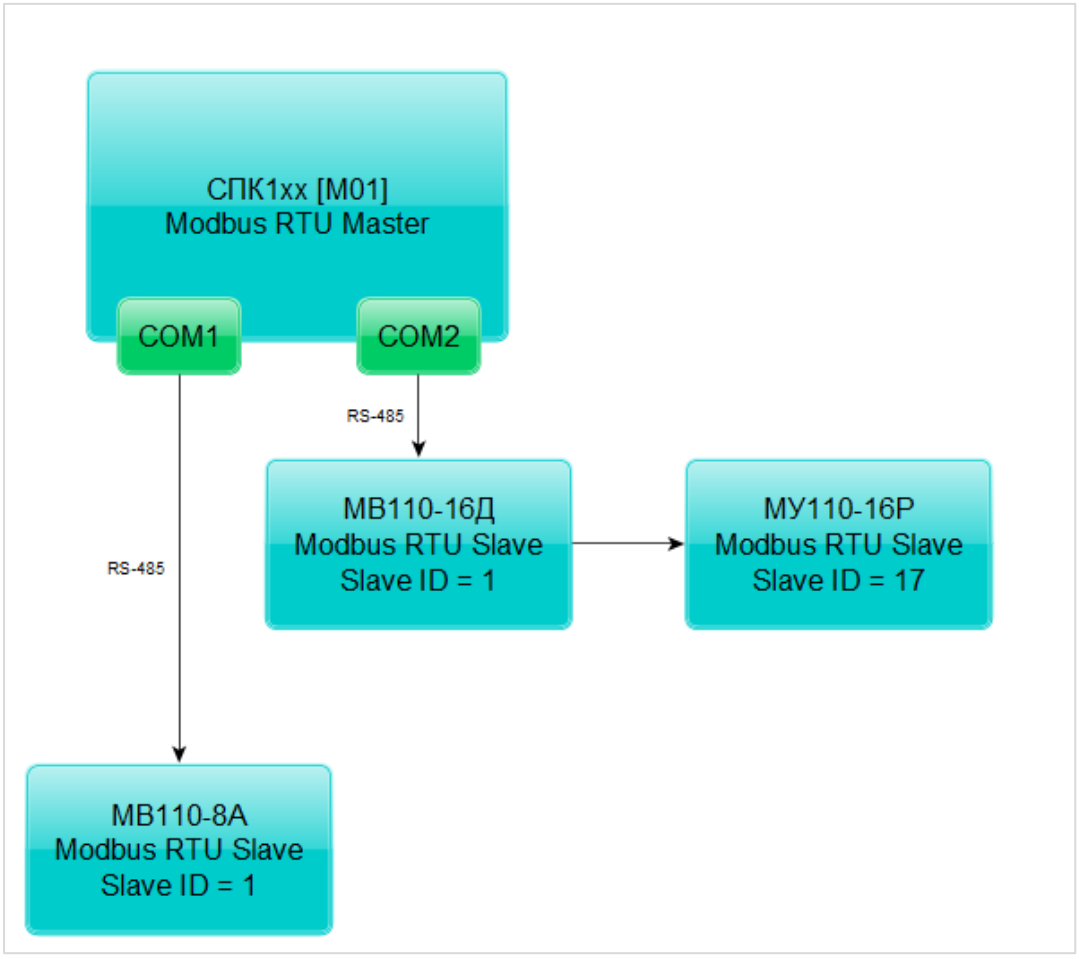


Рисунок 3.2.1 – Структурная схема примера

Пример создан в среде **CODESYS V3.5 SP17 Patch 3** и подразумевает запуск на **СПК1xx [M01]** с таргет-файлом **3.5.17.31.** 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_TemplatesMx110_3517v1.projectarchive

Видеоверсия примера доступна по ссылке.

Сетевые параметры модулей приведены в таблице ниже:

Таблица 3.2.1 – Сетевые параметры модулей Mx110

Параметр	МВ110-8А	МВ110-16Д	МУ110-16Р
COM-порт контроллера, к которому подключен модуль	COM1	COM2	
ID COM-порта	1	2	
Адрес модуля	1	1	17
Скорость обмена	115200		
Количество бит данных	8		
Контроль четности	Отсутствует		
Количество стоп-бит	1		

Для настройки обмена следует:

1. Настроить модули **Mx110** с помощью программы **Конфигуратор Mx110** в соответствии с таблицей 3.2.1. Подключить модули к COM-портам контроллера в соответствии с рисунком3.2.1.

2. Установить пакет шаблонов модулей **Mx110** в CODESYS (см. п. 3.1.1).

3. Создать новый проект **CODESYS** с программой **PLC_PRG** на языке **CFC**:

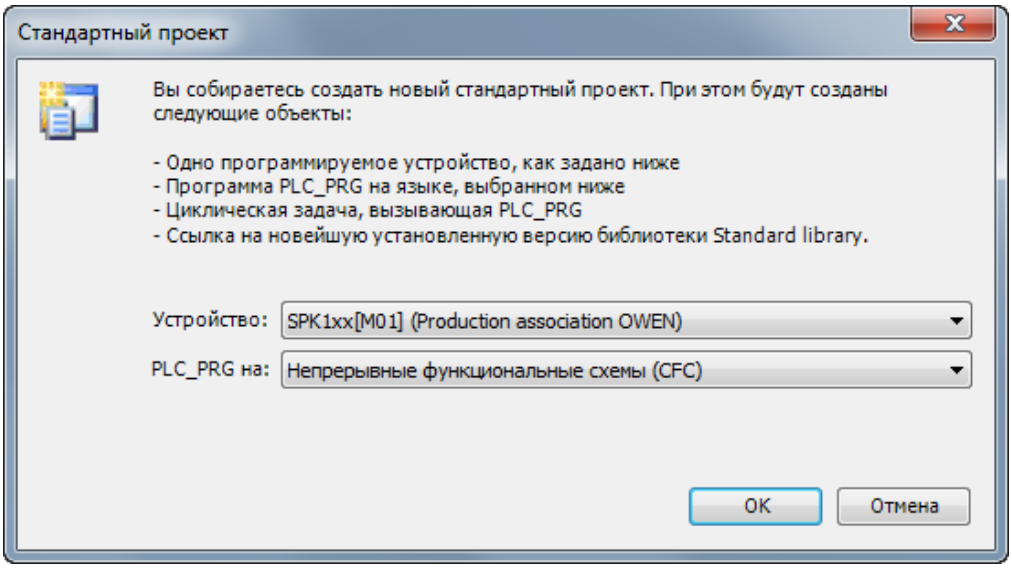


Рисунок 3.2.2 – Создание проекта CODESYS

4. Объявить в программе следующие переменные:

```
PLC_PRG
1  PROGRAM PLC_PRG
2  VAR
3      rAnalogInput1:          REAL;       // 1-й вход модуля MB110-8A
4      xDiscreteInput1:        BOOL;       // 1-й вход модуля MB110-16Д
5      xDiscreteOutput1:       BOOL;       // 1-й выход модуля MУ110-16P
6  END_VAR
7
```

Рисунок 3.2.3 – Объявление переменных в программе PLC_PRG

5. Добавить в проект два компонента **Modbus COM** с названиями **COM1** и **COM2**.

ПРИМЕЧАНИЕ

Версия компонента должна соответствовать версии таргет-файла. Для отображения всех доступных версий компонента следует установить галочку **Отображать все версии**. См. рекомендации в приложении А.

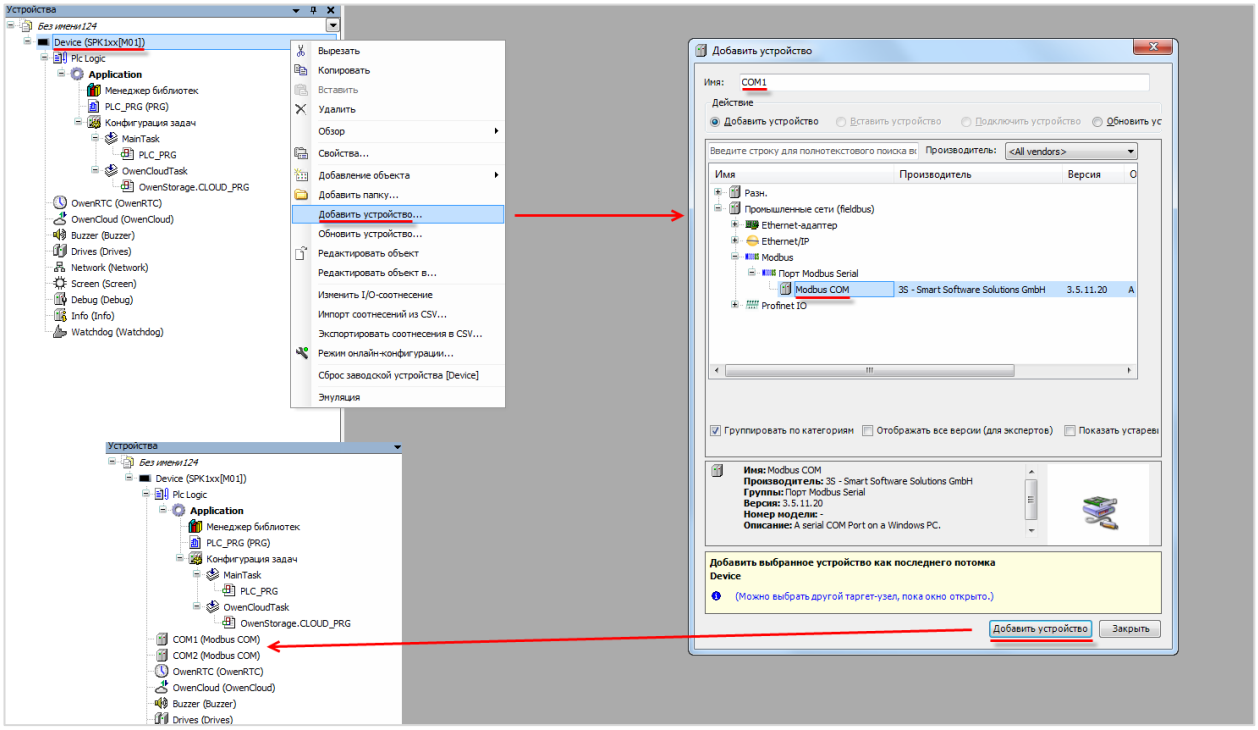


Рисунок 3.2.4 – Добавление компонента Modbus COM

В конфигурации COM-портов следует указать номера COM-портов и сетевые настройки в соответствии с таблицей 3.2.1:

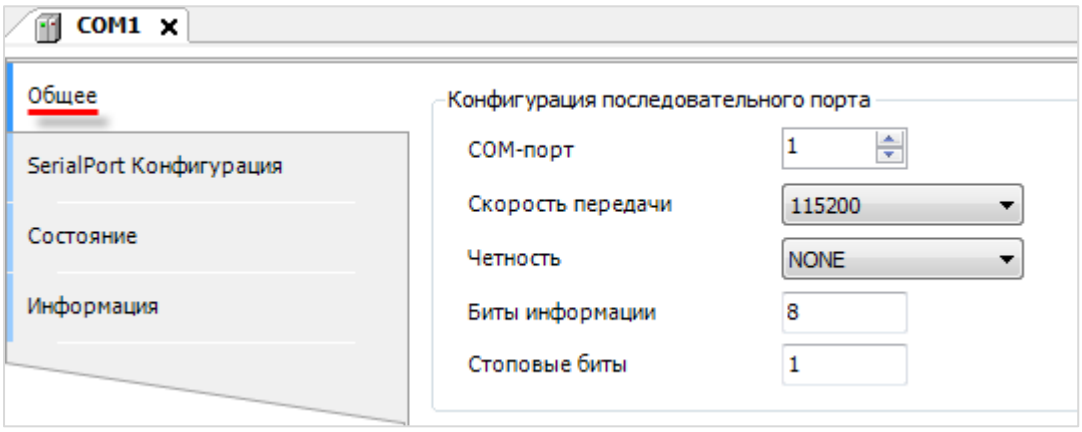


Рисунок 3.2.5 – Настройки COM-порта COM1

6. В каждый из COM-портов добавить компонент **Modbus Master**.

ПРИМЕЧАНИЕ

Версия компонента должна соответствовать версии таргет-файла. Для отображения всех доступных версий компонента следует установить галочку **Отображать все версии**. См. рекомендации в приложении А.

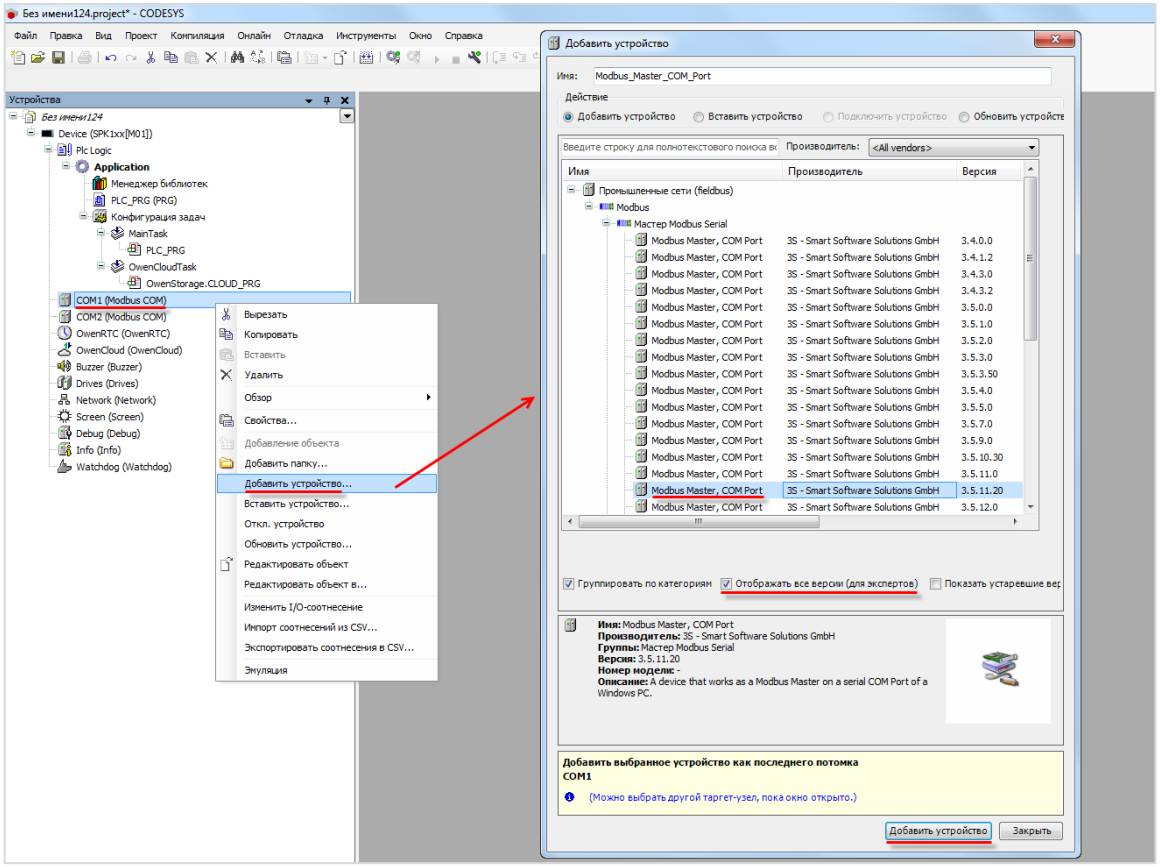


Рисунок 3.2.6 – Добавление компонента Modbus Master

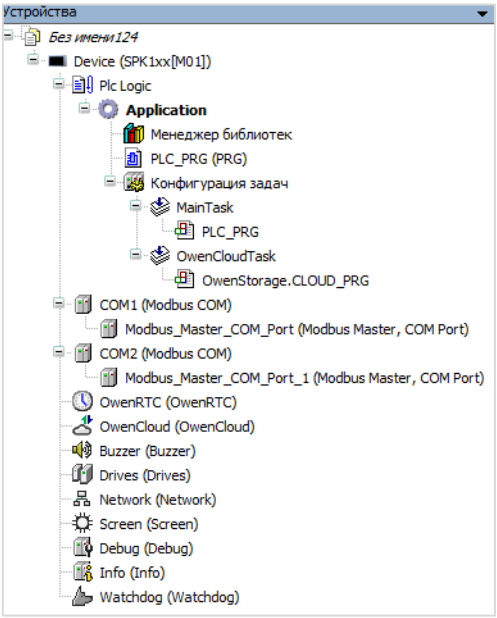


Рисунок 3.2.7 – Внешний вид дерева проекта после добавления Modbus Master

В настройках компонентов на вкладке **Общее** следует установить галочку **Автоперезапуск соединения**. В параметре **Время между фреймами** установить значение **20 мс**.

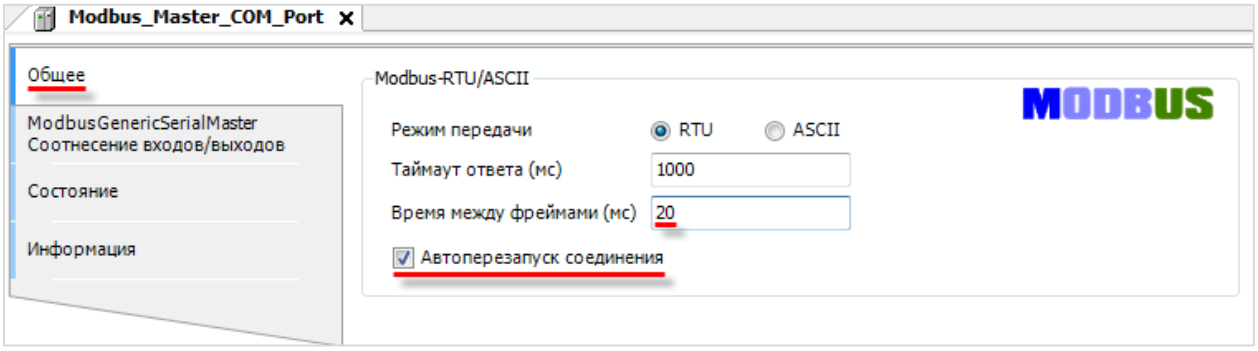


Рисунок 3.2.8 – Настройки компонентов Modbus Master

7. В компонент **Modbus Master** порта **COM1** следует добавить шаблон модуля **MB110-8A,** а в **Modbus Master** порта **COM2** – **MB110-16Д** и **МУ110-16Р**.

ПРИМЕЧАНИЕ

Версия компонента должна соответствовать версии таргет-файла. Для отображения всех доступных версий компонента следует установить галочку **Отображать все версии**. См. рекомендации в приложении А.

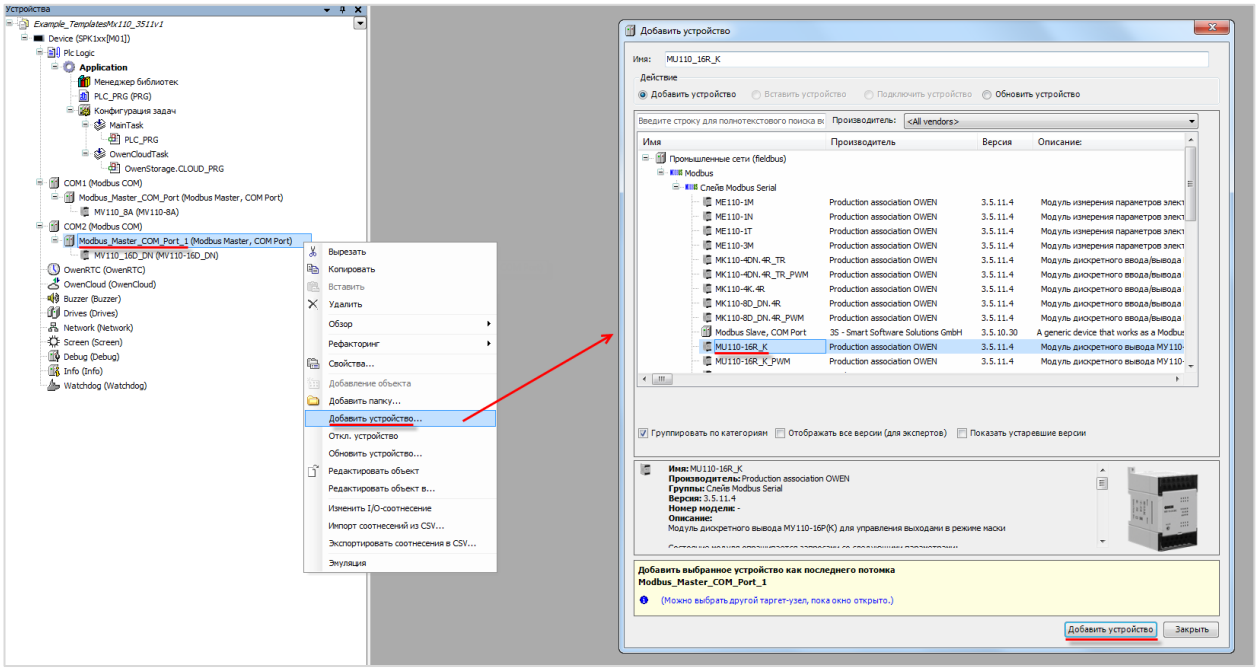


Рисунок 3.2.9 – Добавление шаблонов модулей в проект CODESYS

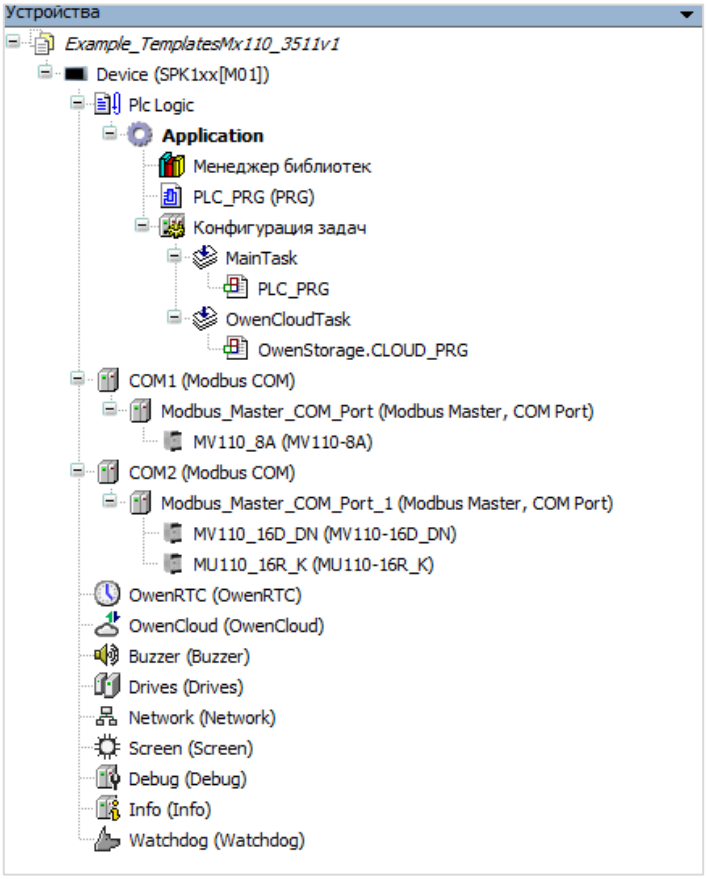


Рисунок 3.2.10 – Внешний вид дерева проекта после добавления шаблонов модулей

В настройках шаблонов следует указать адреса модулей согласно таблице 3.2.1 (МВ110-8А – адрес **1**, МВ110-16Д – адрес **1**, МУ110-16Р – адрес **17**):

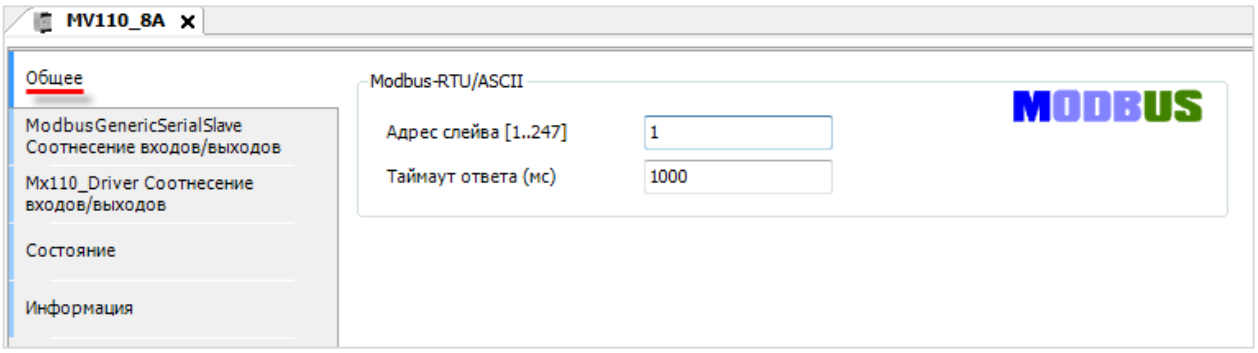


Рисунок 3.2.11 – Настройка шаблона модуля MV110_8A

8. На вкладке **Mx110_Driver Соотнесение входов/выходов** привязать переменные программы к каналам шаблонов в соответствии с данной таблицей:

Таблица 3.2.2 – Привязка переменных к каналам шаблонов

Переменная программы	Модуль	Канал
rAnalogInput1	MV110_8A	Вход 1/Измеренное значение
xDiscreteInput1	MV110_16D_DN	Маска входов/Вход 1
xDiscreteOutput1	MU110_16R_K	Маска выходов/Выход 1

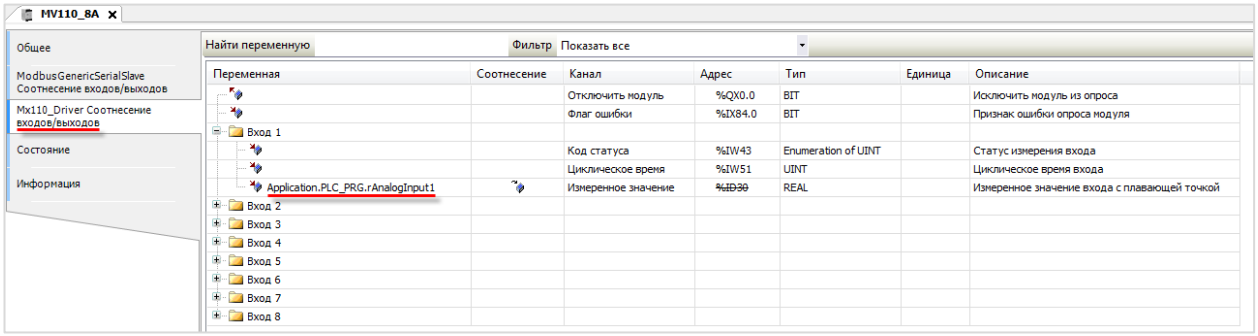


Рисунок 3.2.12 – Привязка переменной к шаблону модуля МВ110-8А

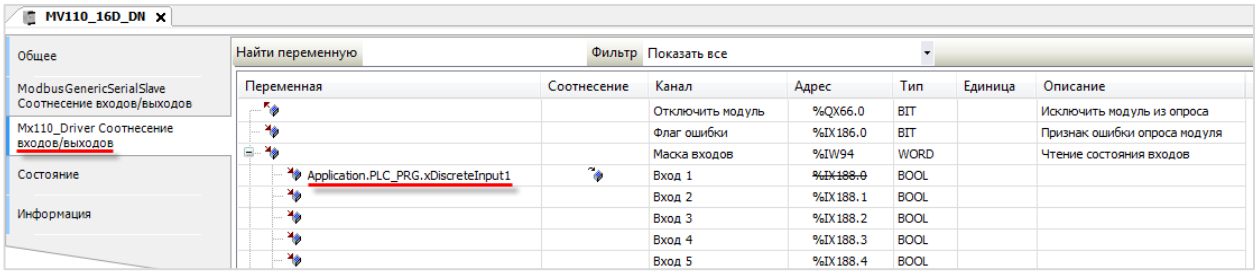


Рисунок 3.2.13 – Привязка переменной к шаблону модуля МВ110-16Д

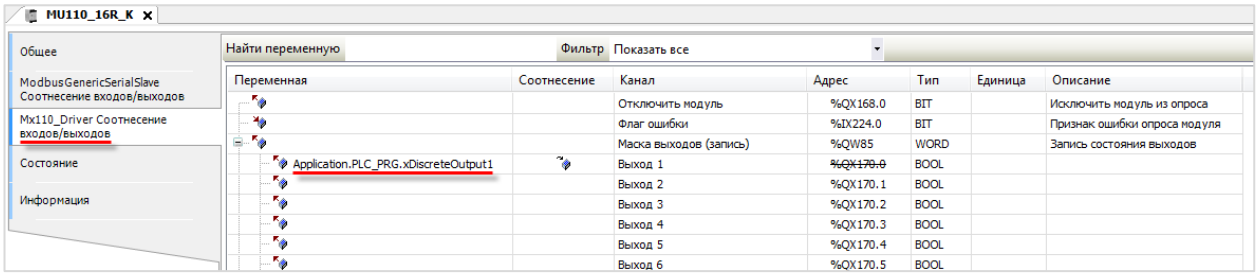


Рисунок 3.2.14 – Привязка переменной к шаблону модуля МУ110-16Р

9. Код программы **PLC_PRG** будет выглядеть следующим образом:

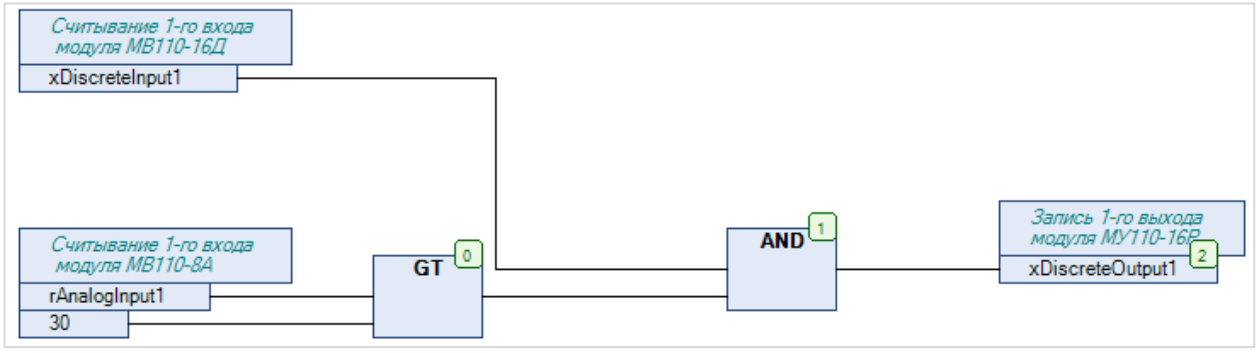


Рисунок 3.2.15 – Код программы на языке CFC

10. Загрузить проект в контроллер и запустить его.

В переменной **rAnalogInput1** будет отображаться текущее значение первого аналогового входа модуля **МВ110-8А**. В переменной **xDiscreteInput1** будет отображаться текущее значение первого дискретного входа модуля **МВ110-16Д**.

Если значение **rAnalogInput1** превысит **30** и при этом значение **xDiscreteInput1** будет равно **TRUE**, то в переменную **xDiscreteOutput1** будет записано значение **TRUE,** что приведет к замыканию первого

дискретного выхода модуля **МУ110-16Р**. Если одно из условий перестанет выполняться, то выход будет разомкнут.

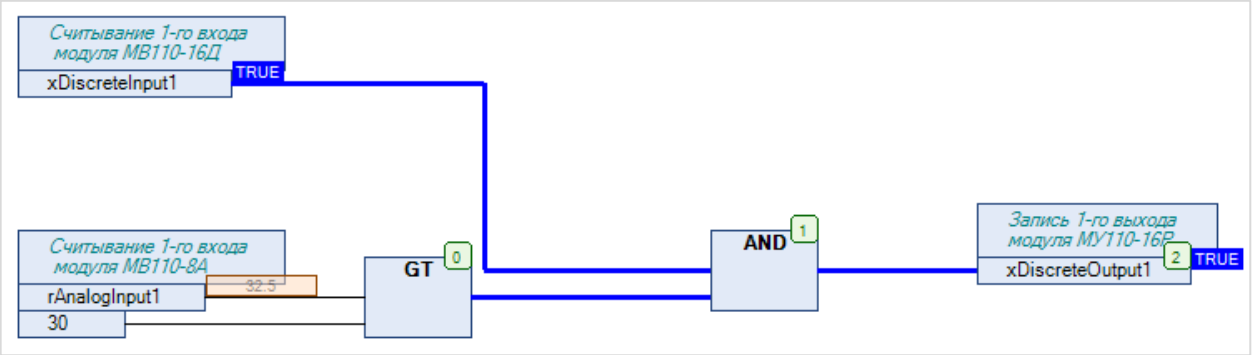


Рисунок 3.2.16 – Выполнение программы в режиме Online

ПРИМЕЧАНИЕ

Более подробная информация о настройках компонентов **ModbusCom** и **Modbus Master** приведена в п. 4.2.

3.3 Пример: СПК1xx [M01] + модули Mx210

В качестве примера будет рассмотрена настройка обмена с модулями Mx210 (MB210-101 и МК210-301) с использованием **шаблонов**. В примере используются шаблоны версии **3.5.11.8**.

Реализуемый алгоритм: если значение первого аналогового входа модуля **MB210-101** превышает **30** и при этом первый дискретный вход модуля **МК210-301** имеет значение **TRUE** (замкнут), то первому дискретному выходу модуля **МК210-301** присваивается значение **TRUE** (замкнут). Во всех остальных случаях дискретному выходу присваивается значение **FALSE** (разомкнут).

Структурная схема примера приведена на рисунке ниже:

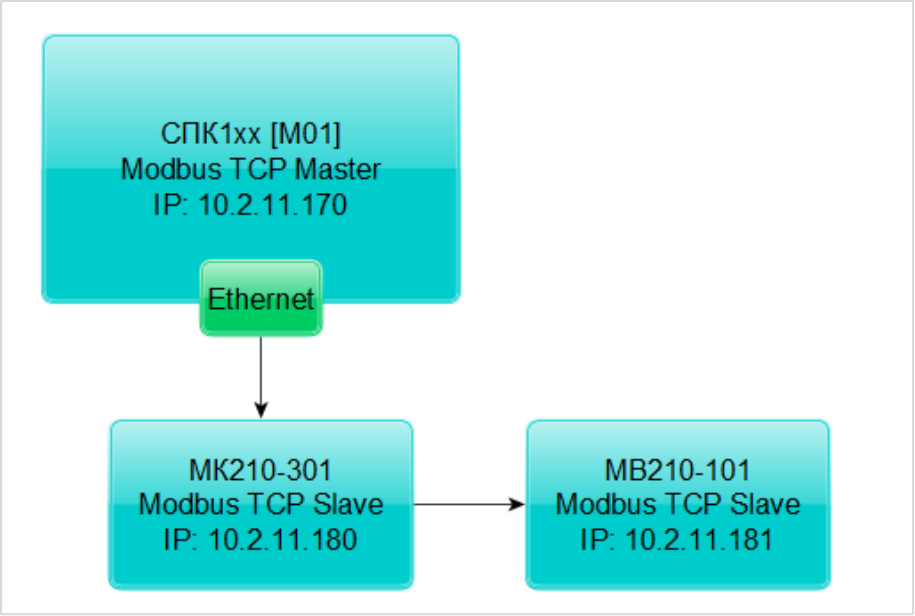


Рисунок 3.3.1 – Структурная схема примера

Пример создан в среде **CODESYS V3.5 SP17 Patch 3** и подразумевает запуск на **СПК1xx [M01]** с таргет-файлом **3.5.17.31.** 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_TemplatesMx210_3517v1.projectarchive

Видеоверсия примера доступна по ссылке.

Информация о влиянии шаблонов на время цикла контроллера доступна по ссылке.

Сетевые параметры устройств приведены в таблице ниже:

Таблица 3.3.1 – Сетевые параметры устройств

Параметр	СПК1xx [M01]	МК210-301	MB210-101
IP-адрес	10.2.11.170	10.2.11.180	10.2.11.181
Маска подсети	255.255.0.0		
IP-адрес шлюза	10.2.1.1		

Для настройки обмена следует:

1. Настроить модули **Mx210** с помощью программы **ОВЕН Конфигуратор** в соответствии с таблицей 3.3.1 (см. руководство **Mx210. Примеры настройки обмена**). Подключить модули к контроллеру.

2. Установить пакет шаблонов модулей **Mx210** в CODESYS (см. п. 3.1.1).

3. Создать новый проект **CODESYS** с программой **PLC_PRG** на языке **CFC**:

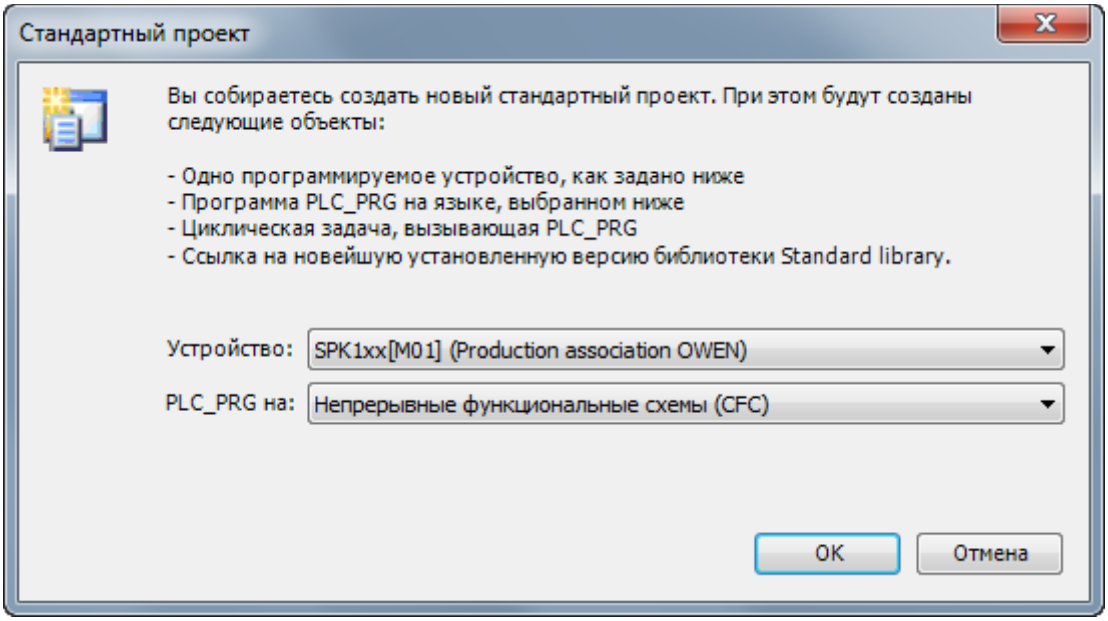


Рисунок 3.3.2 – Создание проекта CODESYS

4. Объявить в программе следующие переменные:

```
PLC_PRG
1  PROGRAM PLC_PRG
2  VAR
3      rAnalogInput1:      REAL;   // 1-й вход модуля MB210-101
4      xDiscreteInput1:    BOOL;   // 1-й вход модуля MK201-301
5      xDiscreteOutput1:   BOOL;   // 1-й выход модуля MK201-301
6  END_VAR
```

Рисунок 3.3.3 – Объявление переменных в программе PLC_PRG

5. Добавить в проект компонент **Ethernet**.

ПРИМЕЧАНИЕ

Версия компонента должна соответствовать версии таргет-файла. Для отображения всех доступных версий компонента следует установить галочку **Отображать все версии**. См. рекомендации в приложении А.

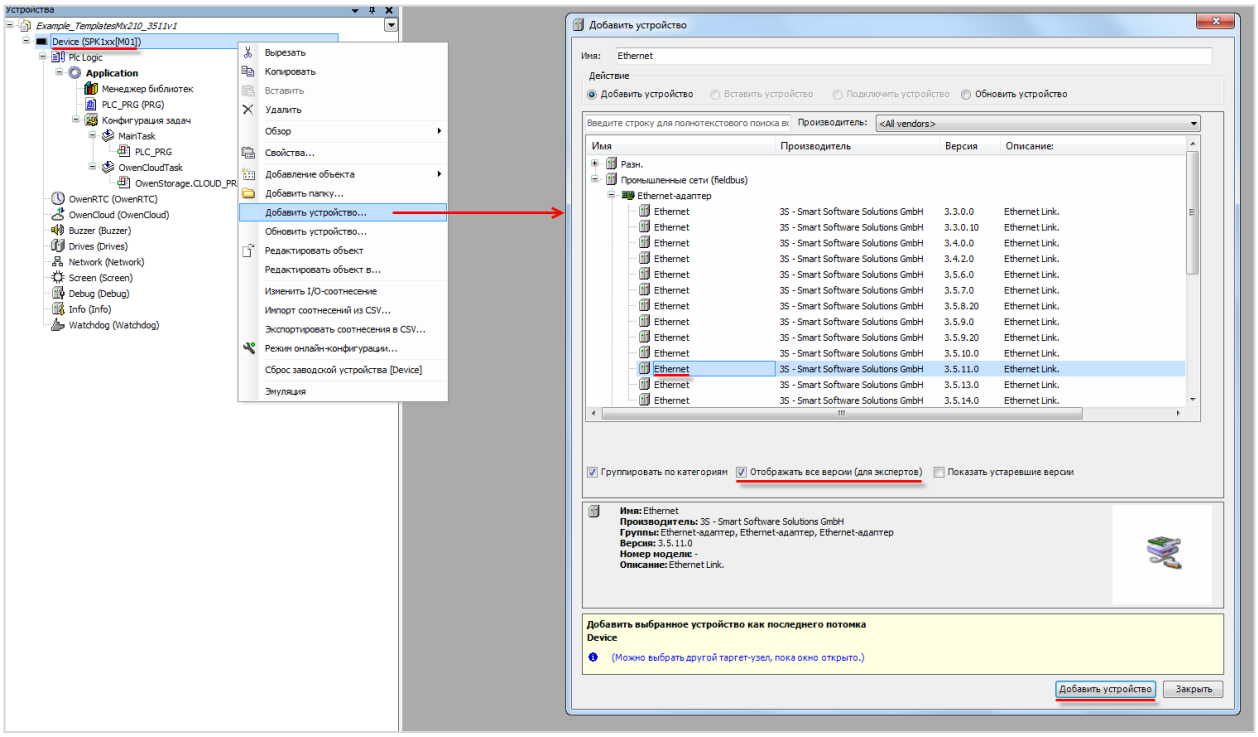


Рисунок 3.3.4 – Добавление компонента Ethernet

Затем следует установить соединение с контроллером, не загружая в него проект (**Device – Установка соединения – Сканировать сеть**) и в компоненте **Ethernet** на вкладке **Конфигурация Ethernet** выбрать нужный интерфейс.

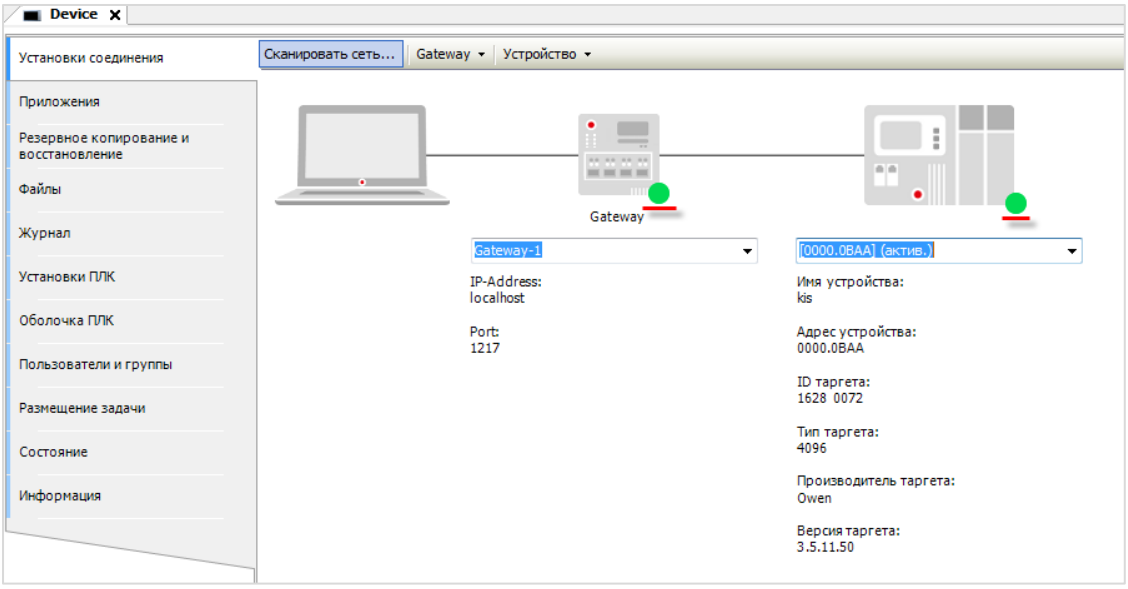


Рисунок 3.3.5 – Подключение к контроллеру

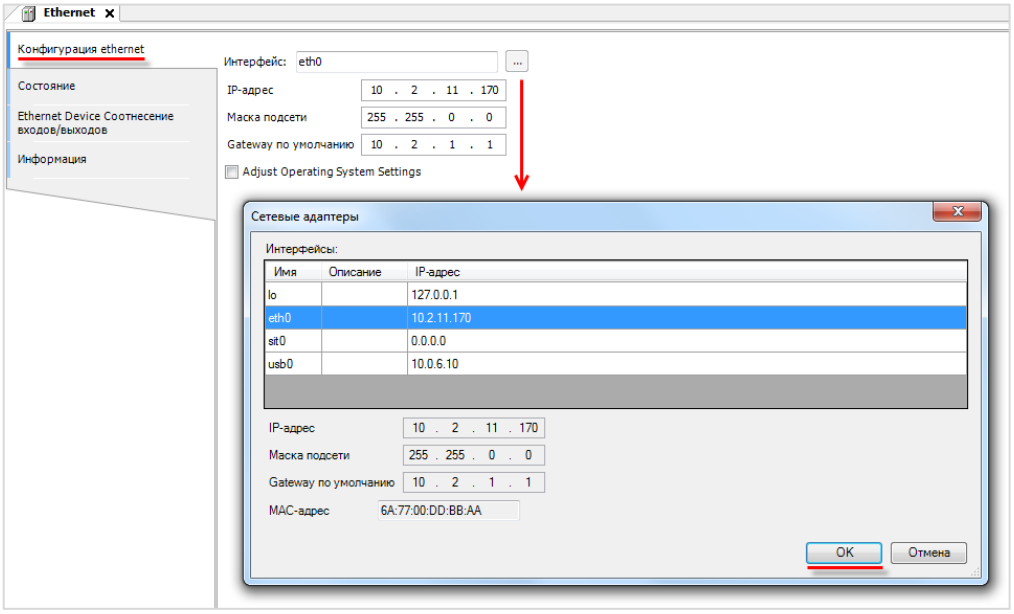


Рисунок 3.3.6 – Выбор используемого интерфейса

ПРИМЕЧАНИЕ

Настройки интерфейса задаются в конфигураторе контроллера (см. документ **CODESYS V3.5. FAQ**).

6. В компонент **Ethernet** добавить компонент **Modbus TCP Master**.

ПРИМЕЧАНИЕ

Версия компонента должна соответствовать версии таргет-файла. Для отображения всех доступных версий компонента следует установить галочку **Отображать все версии**. См. рекомендации в приложении А.

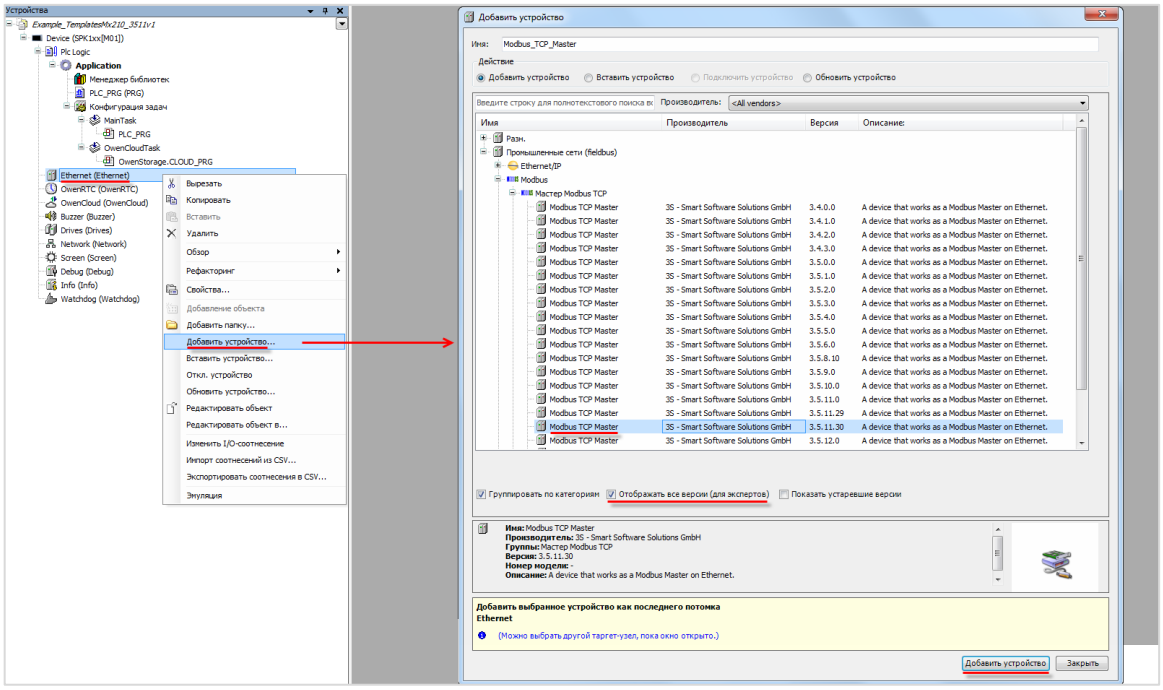


Рисунок 3.3.7 – Добавление компонента Modbus TCP Master

В настройках компонента вкладке **Общее** следует установить галочку **Автоподключение.**

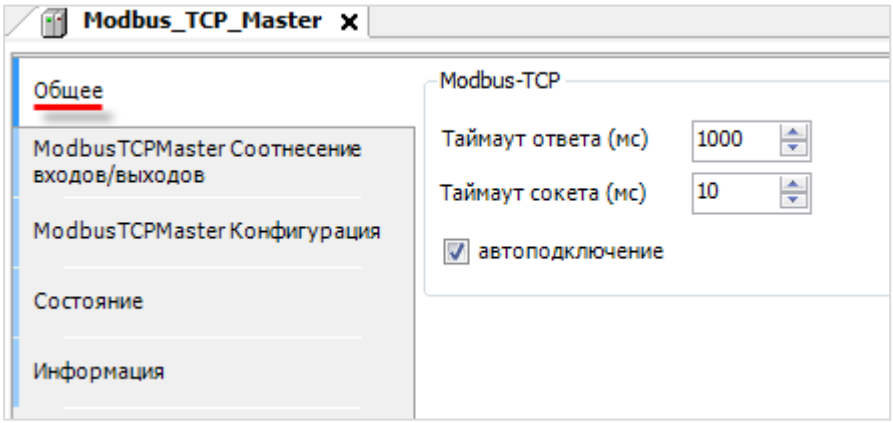


Рисунок 3.3.8 – Настройки компонента Modbus TCP Master

7. В компонент **Modbus TCP Master** следует добавить шаблоны модулей **МК210-301** и **МВ210-101**.

ПРИМЕЧАНИЕ

Версия компонента должна соответствовать версии таргет-файла. Для отображения всех доступных версий компонента следует установить галочку **Отображать все версии**. См. рекомендации в приложении А.

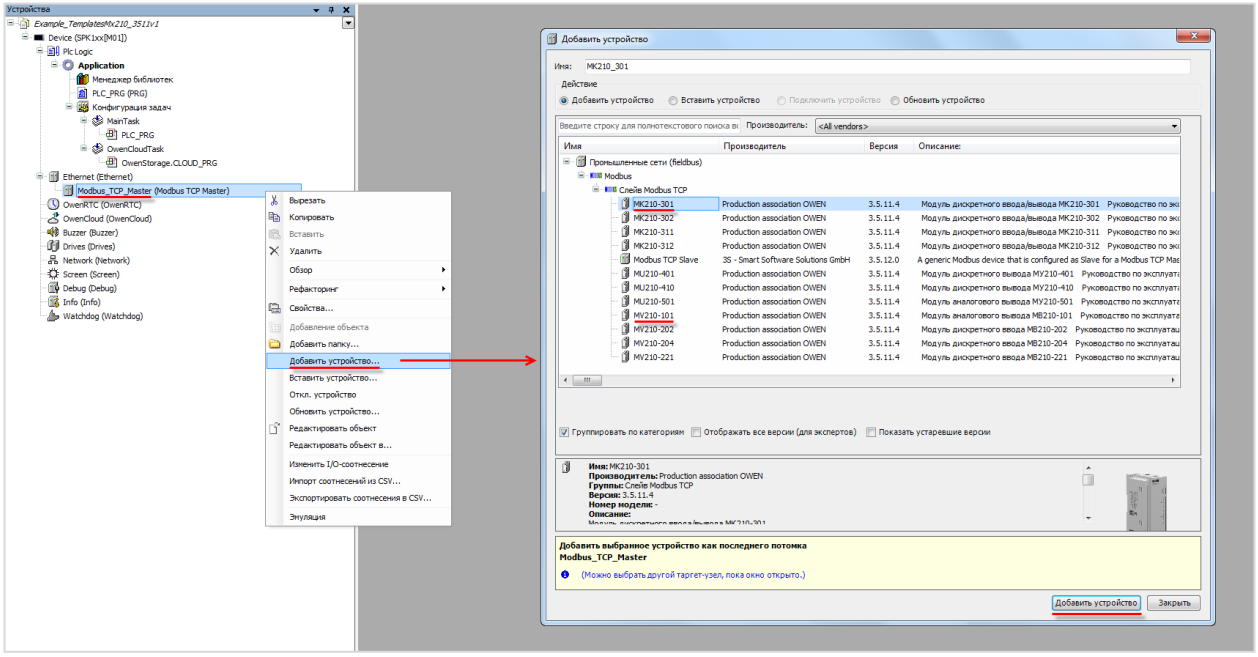


Рисунок 3.3.9 – Добавление шаблонов модулей в проект CODESYS

В настройках модулей следует указать их IP-адреса согласно таблице 3.3.1 (МК210-301 – **10.2.11.180,** МВ210-101 – **10.2.11.181**).

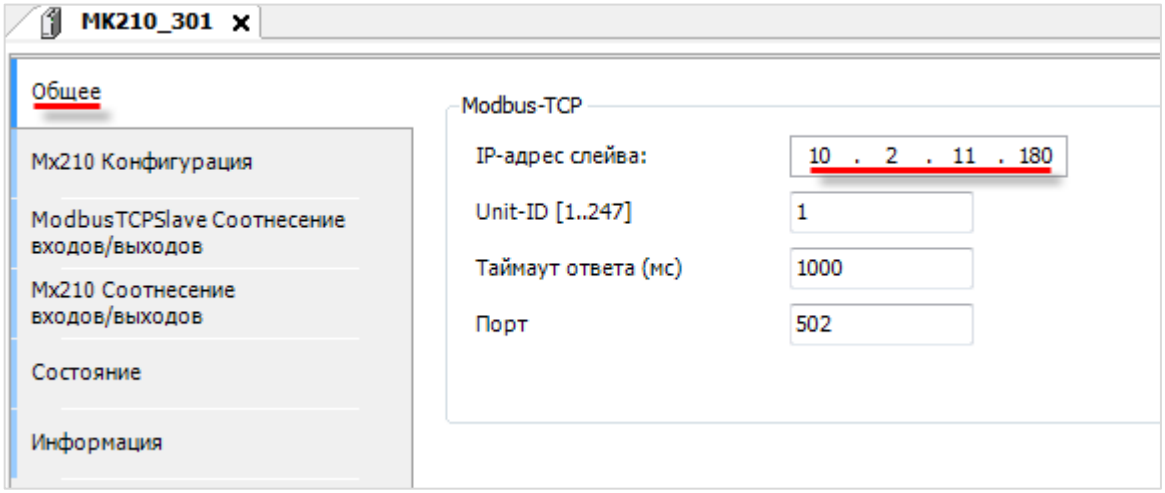


Рисунок 3.3.10 – Настройка шаблона модуля MK210_301

8. На вкладке **Mx210 Соотнесение входов/выходов** привязать переменные программы к каналам шаблонов в соответствии с данной таблицей:

Таблица 3.3.2 – Привязка переменных к каналам шаблонов

Переменная программы	Модуль	Канал
rAnalogInput1	MV210_101	Входы/Вход 1/Измеренное значение
xDiscreteInput1	MK210_301	Входы/Битовая маска входов/Вход 1
xDiscreteOutput1		Выходы/Битовая маска выходов (запись)/Выход 1

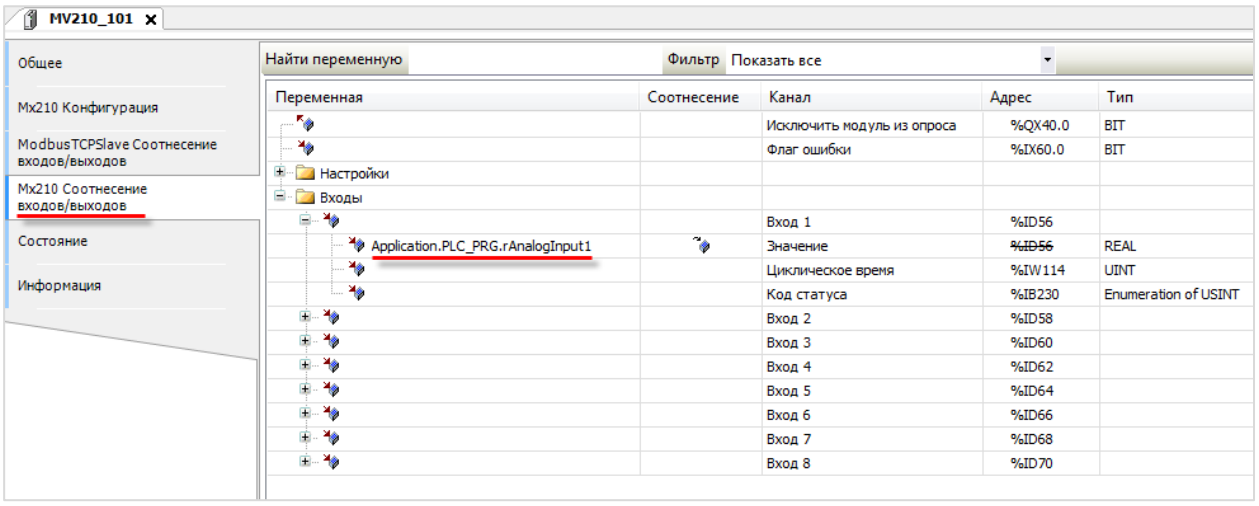


Рисунок 3.3.11 – Привязка переменной к шаблону модуля MB210-101

ПРИМЕЧАНИЕ

Конфигурационные параметры модулей доступны для изменения на вкладке **Mx210 Конфигурация**. В частности, на этой вкладке задается минимальный период опроса модулей и режим записи выходов (для модулей с выходами) – циклически или по изменению. Для модуля **MB210-101** на этой вкладке можно задать настройки аналоговых входов (начиная с версии шаблона **3.5.11.6**). По умолчанию при старте проекта настройки входов модуля будут перезаписаны настройками входов, заданными на этой вкладке. Чтобы отключить это поведение – необходимо задать параметру **Использовать настройки конфигурации** значение **FALSE**.

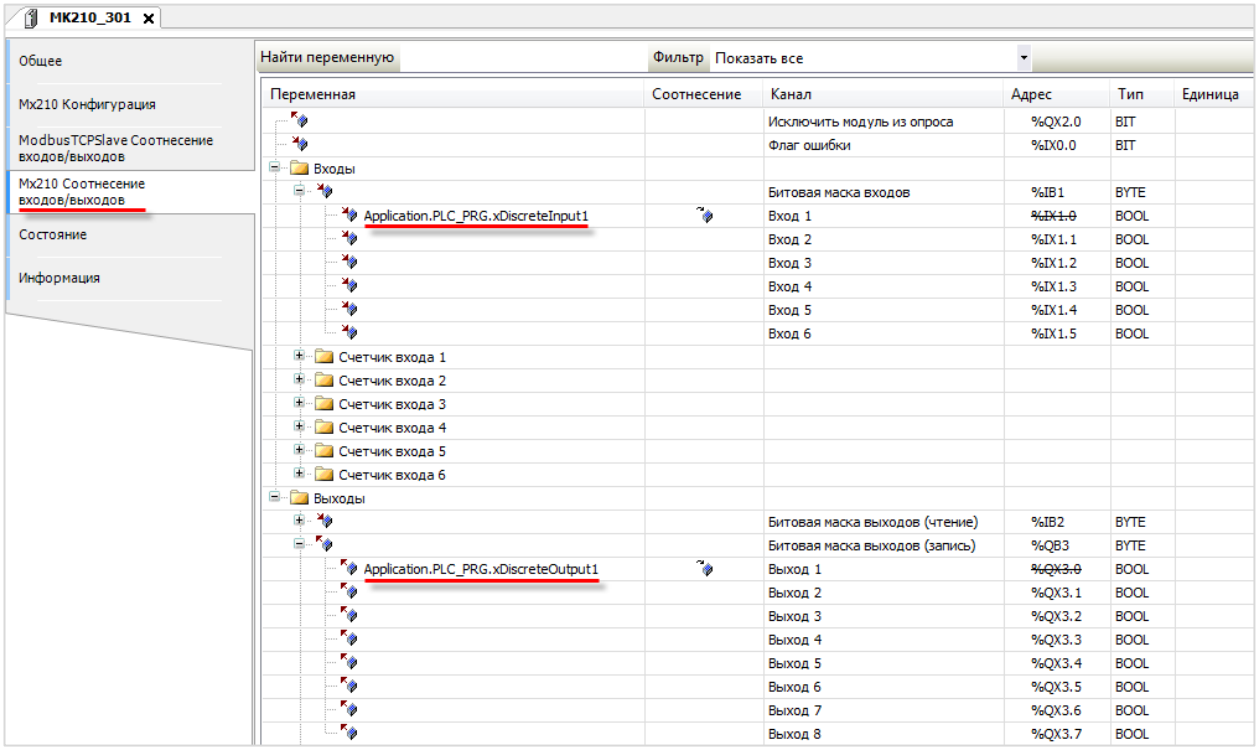


Рисунок 3.3.12 – Привязка переменной к шаблону модуля МК210-301

9. Код программы **PLC_PRG** будет выглядеть следующим образом:

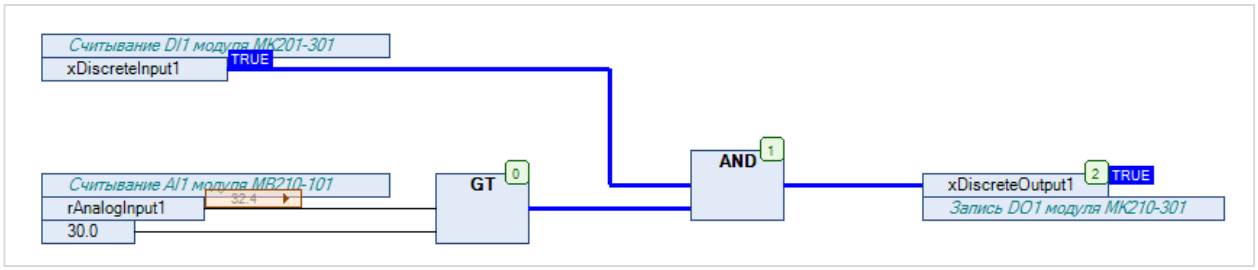


Рисунок 3.3.13 – Код программы на языке CFC

10. Загрузить проект в контроллер и запустить его.

В переменной **rAnalogInput1** будет отображаться текущее значение первого аналогового входа модуля **MB210-101**. В переменной **xDiscreteInput1** будет отображаться текущее значение первого дискретного входа модуля **МК210-301**.

Если значение **rAnalogInput1** превысит 30 и при этом значение **xDiscreteInput1** будет равно **TRUE**, то в переменную **xDiscreteOutput1** будет записано значение **TRUE,** что приведет к замыканию первого дискретного выхода модуля **МК210-301**. Если одно из условий перестанет выполняться, то выход будет разомкнут.

ПРИМЕЧАНИЕ

Более подробная информация о настройках компонентов **Ethernet** и **Modbus TCP Master** приведена в п. 4.4.

3.4 Диагностика и управление обменом

Шаблон каждого модуля содержит каналы **Флаг ошибки** и **Исключить модуль из опроса**.

Канал **Флаг ошибки** принимает значение **TRUE** в случае ошибки обмена (например, ответ от модуля не пришел или из-за действия помех на линию связи пришел некорректный ответ).

Канал **Исключить модуль из опроса** позволяет остановить опрос модуля: пока канал имеет значение **TRUE,** то модуль не опрашивается.

Шаблоны модулей **Mx210** включают в себя переменные диагностики. Для их использования следует в нужном месте программы ввести имя модуля из дерева проекта с постфиксом _**OwenDriver**, поставить точку и из выпадающего списка выбрать нужную переменную:

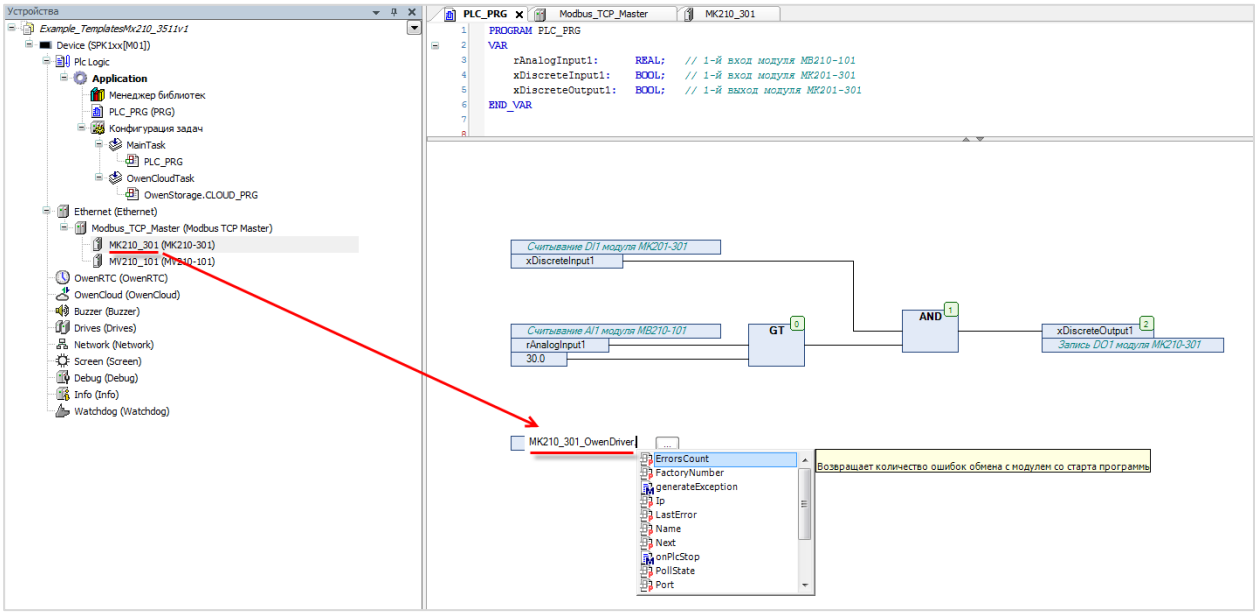


Рисунок 3.4.1 – Переменные диагностики модулей Mx210

Таблица 3.4.1 – Список переменных диагностики шаблонов модулей Mx210

Название	Тип	Описание
ErrorsCount	UDINT	Количество ошибок связи с модулем с момента последней загрузки контроллера
FactoryNumber	STRING	Заводской номер модуля
Ip	ARRAY [0..3] OF BYTE	IP-адрес модуля, заданный в шаблоне
LastError	IoDrvModbusTCP.MB_ErrorCodes	Код последней ошибки обмена
Name	STRING	Имя модуля, заданное в его прошивке
PollState	IoDrvMx210.MODULE_STATE	Состояние обмена с модулем
Port	UINT	Порт Modbus TCP, заданный в шаблоне
ProjectName	STRING	Имя модуля в дереве проекта
SoftwareVersion	STRING	Версия прошивки модуля

Параметры **FactoryNumber**, **Name** и **SoftwareVersion** считываются при каждом возобновлении обмена с модулем (то есть при физической замене одного модуля на другой идентичной модели – значения параметров будут актуализированы).

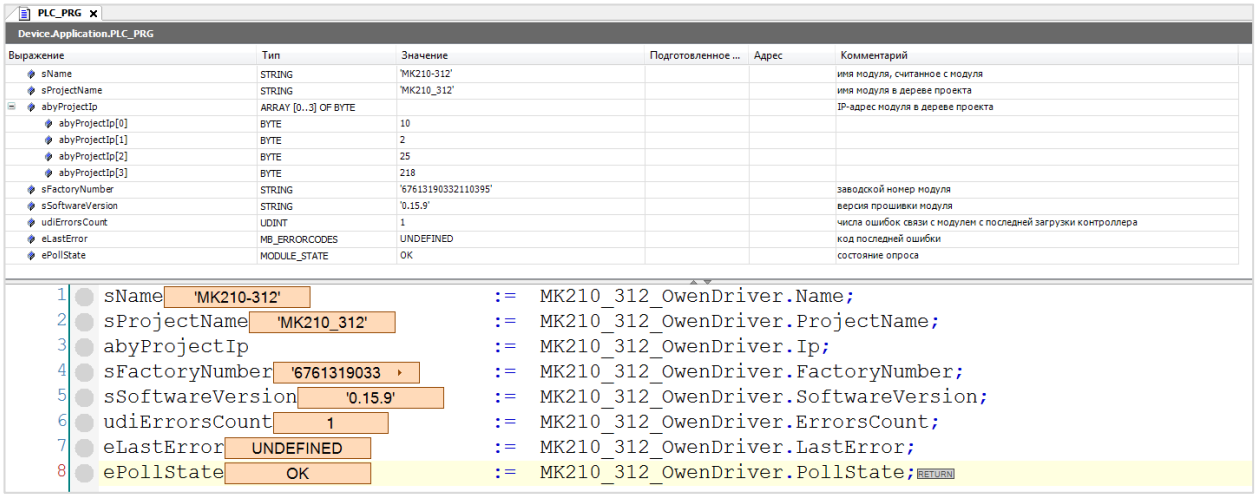


Рисунок 3.4.2 – Использование переменных диагностики

В процессе отладки может потребоваться информация о внутреннем устройстве шаблона (используемые коды функций, количество регистров в запросе и т. д.). Эти сведения приведены на вкладке **Информация**:

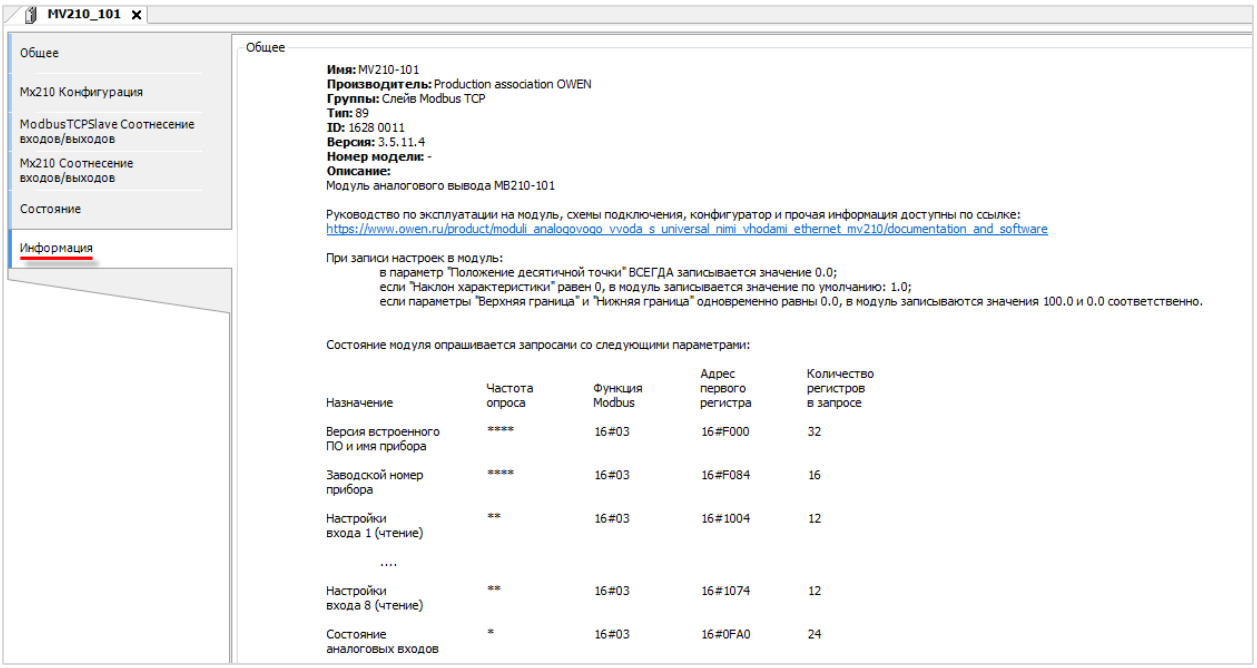


Рисунок 3.4.3 – Информация об устройстве шаблона

Информация о переменных диагностики стандартных компонентов (**Modbus Master, Modbus Slave** и т. д.) приведена в п. 4.6.

3.5 Библиотеки Mx Assistant

При добавлении в проект шаблонов модулей аналогового ввода и вывода в **Менеджере библиотек** будет автоматически добавлена библиотека **Mx110 Assistant** (для шаблонов модулей **Mx110**) или **Mx210 Assistant** (для шаблонов модулей **Mx210**).

В случае необходимости библиотека может быть добавлена в проект вручную (**Менеджер библиотек – Добавить библиотеку**).

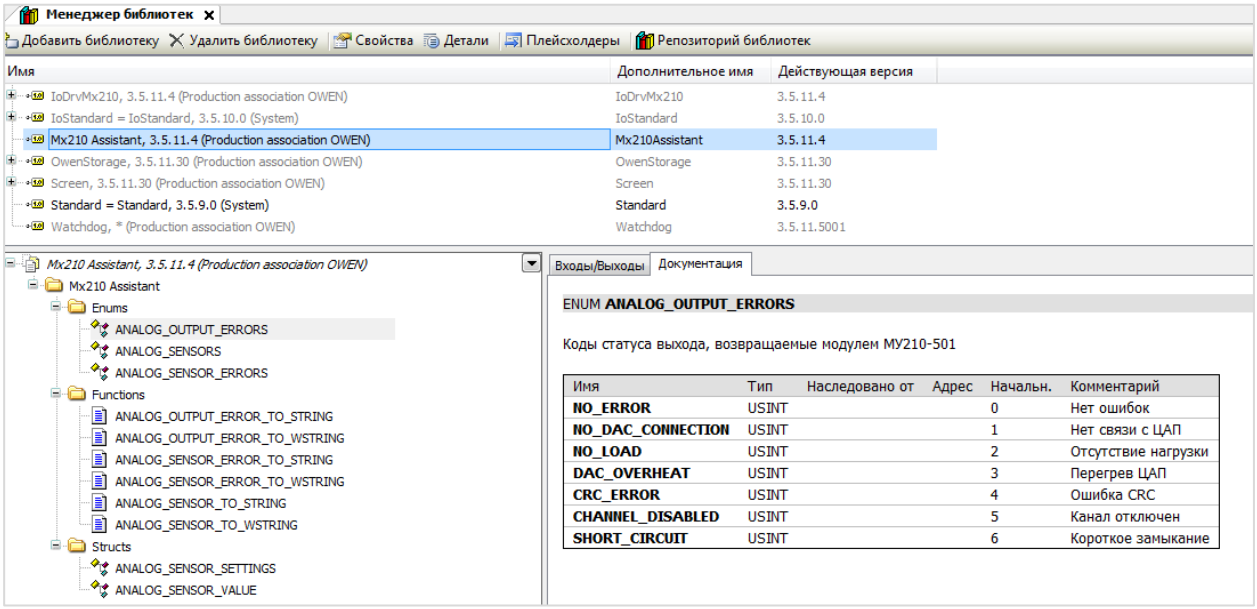


Рисунок 3.5.1 – Библиотека Mx210 Assistant в Менеджере библиотек

Таблица 3.5.1 – Содержимое библиотеки Mx110 Assistant

Название	Тип	Описание
MV_SENSOR_ERROR	Перечисление	Код статуса аналогового входа модуля **MB110-2A/2AC/8A/8AC**. Если зафиксирована ошибка обмена (канал **Флаг ошибки** имеет значение **TRUE**), то будет возвращено значение **MODBUS_ERROR**
MvStatusToString	Функция	Функция преобразует код статуса в строку типа **STRING**
MvStatusToWstring	Функция	Функция преобразует код статуса в строку типа **WSTRING**

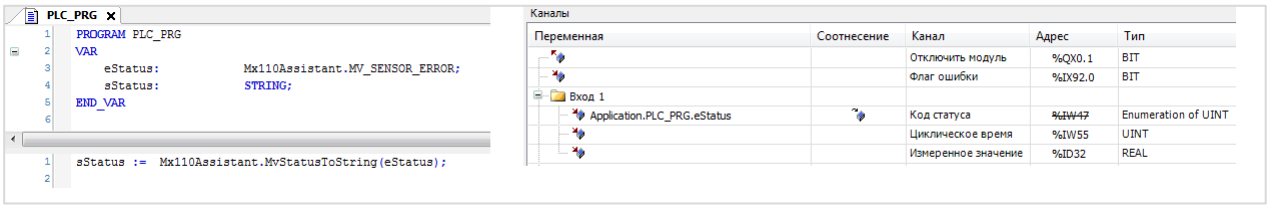


Рисунок 3.5.2 – Пример работы с функциями библиотеки Mx110 Assistant

Таблица 3.5.2 – Содержимое библиотеки Mx210 Assistant

Название	Тип	Описание
ANALOG_SENSOR_VALUE	Структура	Структура параметров аналогового входа модуля **MB210-101**
ANALOG_SENSOR_SETTINGS	Структура	Структура настроек аналогового входа модуля **MB210-101**
ANALOG_SENSORS	Перечисление	Код типа датчика для аналогового входа модуля **MB210-101**
ANALOG_SENSOR_ERRORS	Перечисление	Код статуса аналогового входа модуля **MB210-101**. Если зафиксирована ошибка обмена (канал **Флаг ошибки** имеет значение **TRUE**), то будет возвращено значение **FIELDBUS_ERROR**
ANALOG_OUTPUT_ERRORS	Перечисление	Код статуса аналогового выхода модуля **МУ210-501**
ANALOG_SENSOR_TO_STRING	Функция	Функция преобразует код типа датчика в строку типа **STRING**
ANALOG_SENSOR_TO_WSTRING	Функция	Функция преобразует код типа датчика в строку типа **WSTRING**
ANALOG_SENSOR_ERROR_TO_STRING	Функция	Функция преобразует код статуса аналогового входа в строку типа **STRING**
ANALOG_SENSOR_TO_ERROR_WSTRING	Функция	Функция преобразует код статуса аналогового входа в строку типа **WSTRING**
ANALOG_OUTPUT_ERROR_TO_STRING	Функция	Функция преобразует код статуса аналогового выхода в строку типа **STRING**
ANALOG_OUTPUT_ERROR_TO_WSTRING	Функция	Функция преобразует код статуса аналогового выхода в строку типа **WSTRING**

4 Стандартные средства конфигурирования

4.1 Общая методика конфигурирования интерфейсов

Настройка обмена в **CODESYS** состоит из следующих действий:

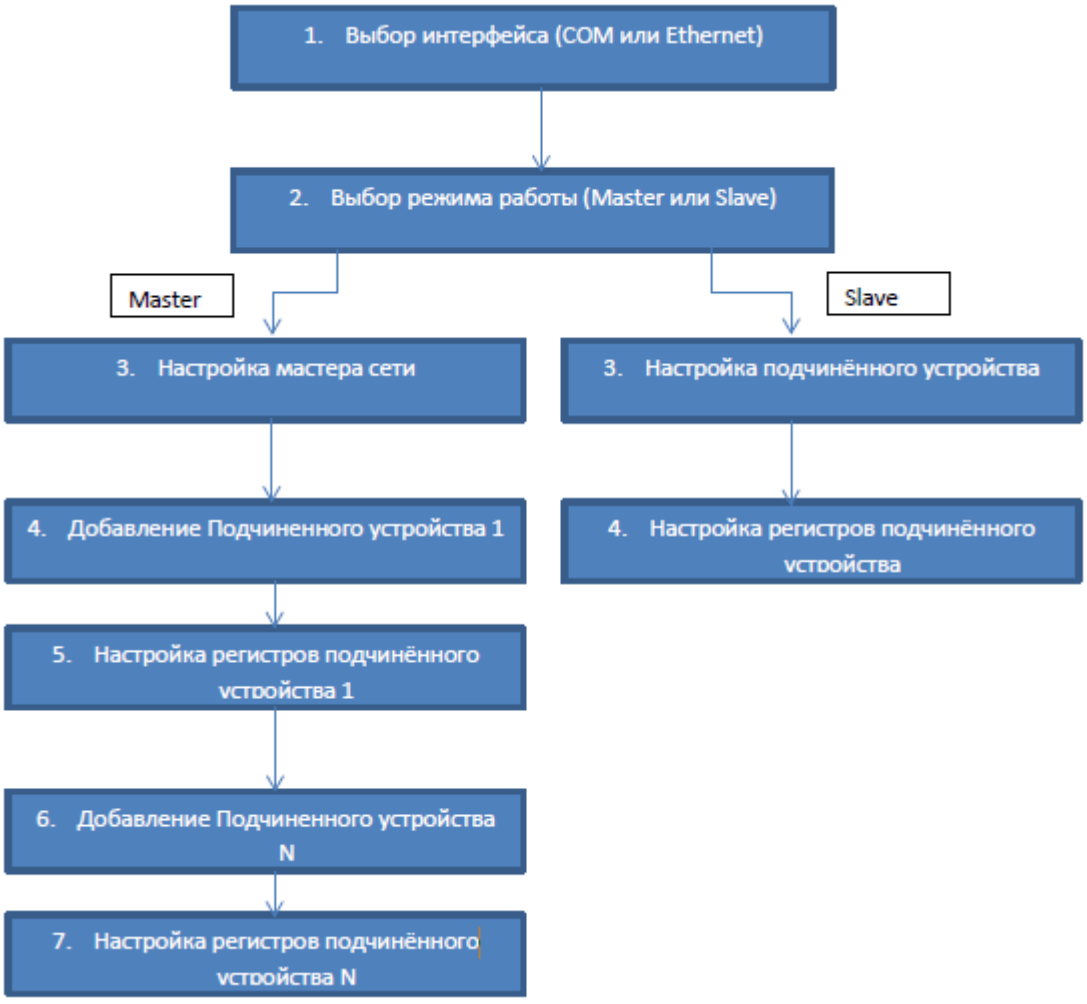


Рисунок 4.1.1 – Последовательность конфигурирования Modbus в CODESYS

Сначала следует добавить и настроить интерфейс. Затем выбрать режим работы интерфейса – **Master** или **Slave** (режим работы представляется отдельным компонентом). Если интерфейс работает в режиме master, то следует добавить все slave-устройства и указать для них адреса и опрашиваемые/записываемые регистры. Если интерфейс работает в режиме slave, то достаточно привязать к его регистрам нужные переменные.

4.2 Настройка контроллера в режиме Modbus Serial Master

Для настройки контроллера в режиме **Modbus Serial Master** следует:

1. Нажать **ПКМ** на компонент **Device** и добавить компонент **Modbus COM**, расположенный во вкладке **Промышленные сети/Modbus/Порт Modbus Serial**.

ПРИМЕЧАНИЕ
Версия компонента должна соответствовать версии таргет-файла. Для отображения всех доступных версий компонента следует установить галочку **Отображать все версии**. См. рекомендации в приложении А.

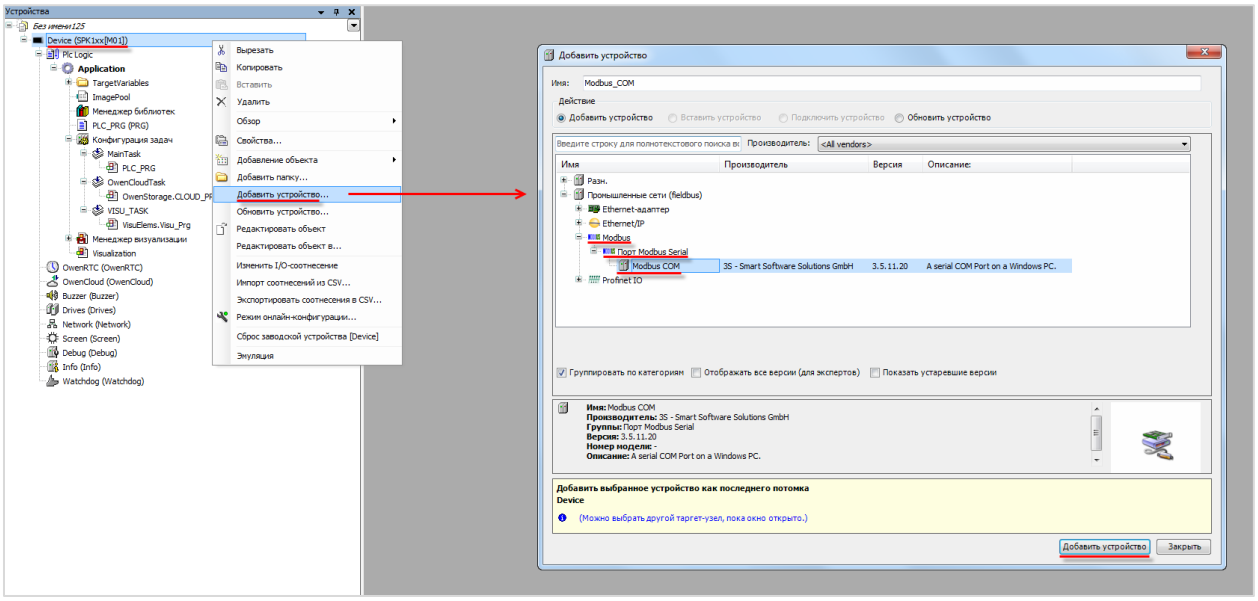


Рисунок 4.2.1 – Добавление компонента Modbus COM

В настройках компонента на вкладке **Общее** следует указать номер COM-порта контроллера и его сетевые настройки.

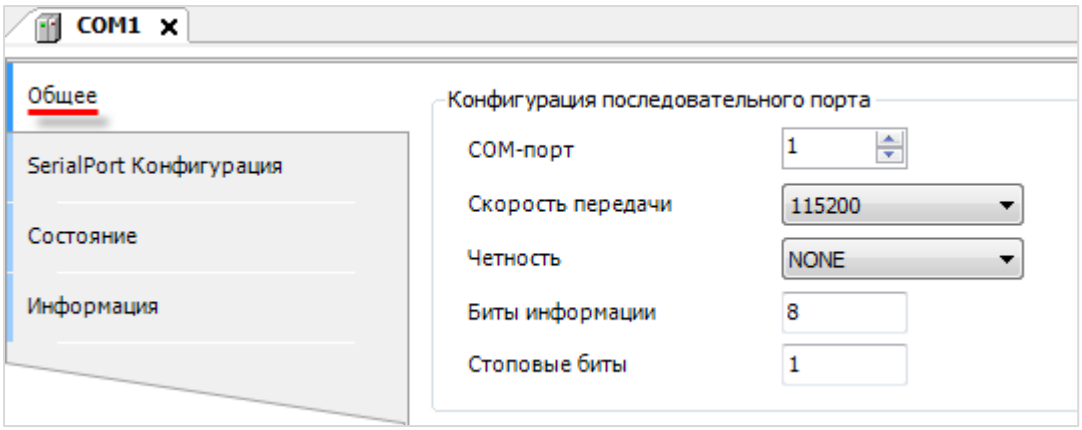


Рисунок 4.2.2 – Настройки компонента Modbus COM

Таблица 4.2.1 – Настройки компонента Modbus COM

Параметр	Описание
COM-порт	Идентификатор используемого COM-порта (см. п. 2.3)
Скорость передачи	Скорость передачи данных в бодах, возможные значения: **1200/2400/4800/9600/19200/38400/57600/115200**
Четность	Режим контроля четности: **NONE** – отсутствует, **EVEN** – проверка на четность, **ODD** – проверка на нечетность
Биты информации	Число бит данных. Возможные значения: **7** или **8**
Стоповые биты	Число стоповых бит. Возможные значения: **1** или **2**

2. Нажать **ПКМ** на компонент **Modbus COM** и добавить компонент **Modbus Master**, расположенный во вкладке **Промышленные сети/Modbus/Мастер Modbus Serial**.

ПРИМЕЧАНИЕ

Версия компонента должна соответствовать версии таргет-файла. Для отображения всех доступных версий компонента следует установить галочку **Отображать все версии**. См. рекомендации в приложении А.

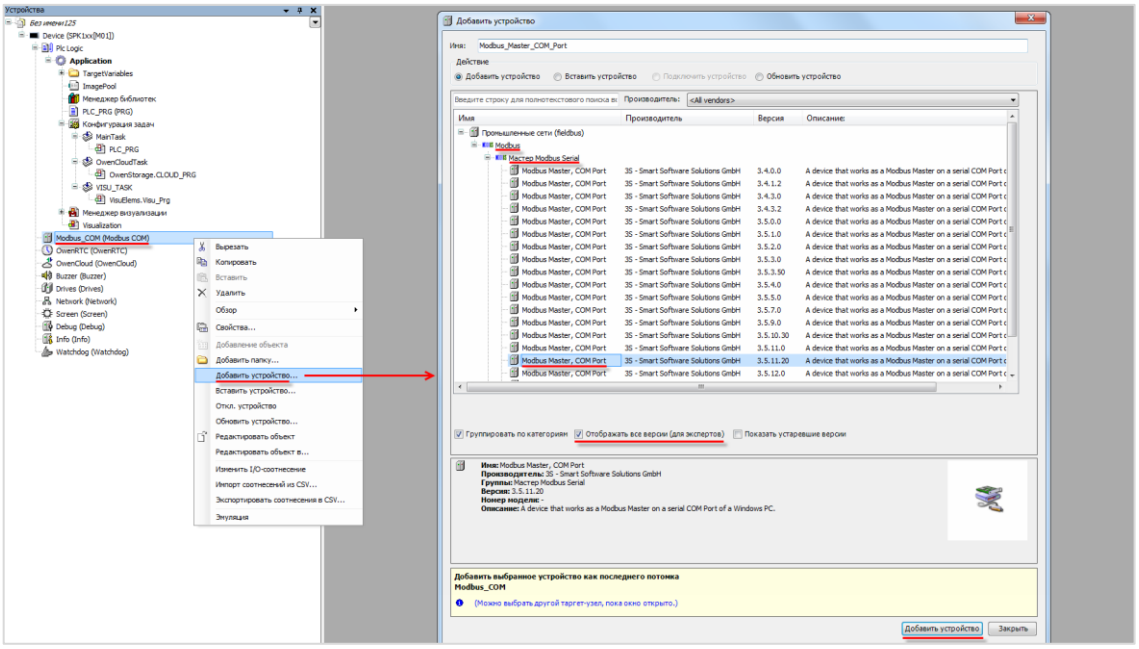


Рисунок 4.2.3 – Добавление компонента Modbus Master

В настройках компонента на вкладке **Общее** следует задать настройки master-устройства.

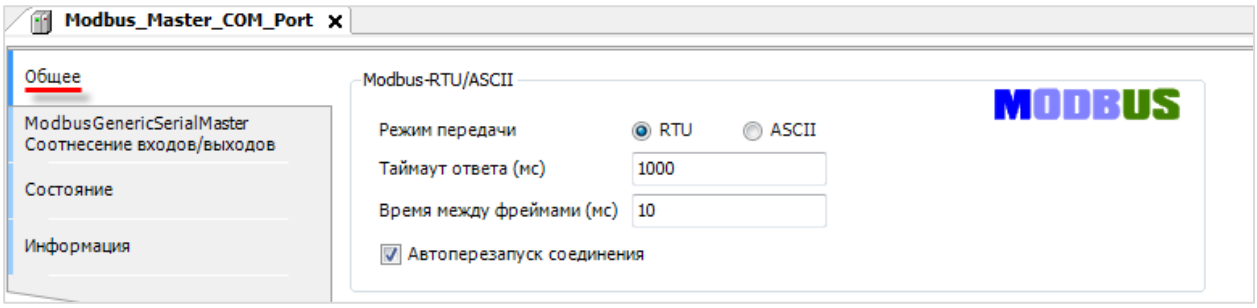


Рисунок 4.2.4 – Настройки компонента Modbus Master

Таблица 4.2.2 – Настройки компонента Modbus Master

Параметр	Описание
Режим передачи	Тип протокола обмена: **Modbus RTU** или **Modbus ASCII**
Таймаут ответа	Время (в мс), в течение которого master ожидает ответа slave-устройства. В случае отсутствия ответа по истечению этого времени master делает паузу на **время между фреймами** и переходит к опросу следующего канала slave-устройства (или следующему slave-устройству). Значение, введенное здесь, будет по умолчанию использоваться для всех slave-устройств. На вкладке **Конфигурация Modbus Slave** (см. рисунок 4.2.6) для каждого устройства можно задать индивидуальный таймаут отклика. Начиная с версии **3.5.17.0** в случае отсутствия ответа отправляется один повторный запрос
Время между фреймами	Время (в мс) между получением ответа от slave-устройства и началом опроса следующего. Чем выше скорость, тем меньшим может быть это значение (на скорости 115200 бит/с – 5–20 мс). Некоторые устройства в течение определенного времени (например, ОВЕН СМИ2 – на 50 мс) удерживают линию связи после ответа, поэтому в данном случае не имеет смысла выставлять время между фреймами меньше, чем это значение. При работе с модулями **Mx110** рекомендуется использовать значение **20 мс**
Автоперезапуск соединения	В случае отсутствия галочки не ответившее slave-устройство исключается из дальнейшего опроса. **Настоятельно рекомендуется** всегда включать эту опцию

3. Нажать **ПКМ** на компонент **Modbus Master** и добавить компонент **Modbus Slave**, расположенный во вкладке **Промышленные сети/Modbus/Слейв Modbus Serial**. Число компонентов должно соответствовать числу slave-устройств, подключенных к COM-порту. Максимальное возможное количество slave-устройств для одного master-устройства – **32**.

ПРИМЕЧАНИЕ

Версия компонента должна соответствовать версии таргет-файла. Для отображения всех доступных версий компонента следует установить галочку **Отображать все версии**. См. рекомендации в приложении А.

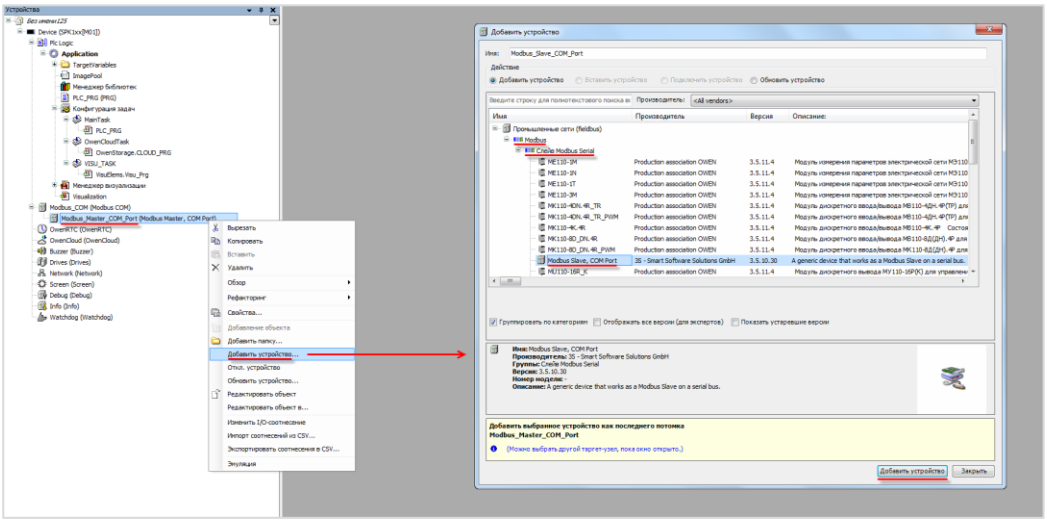


Рисунок 4.2.5 – Добавление компонента Modbus Slave

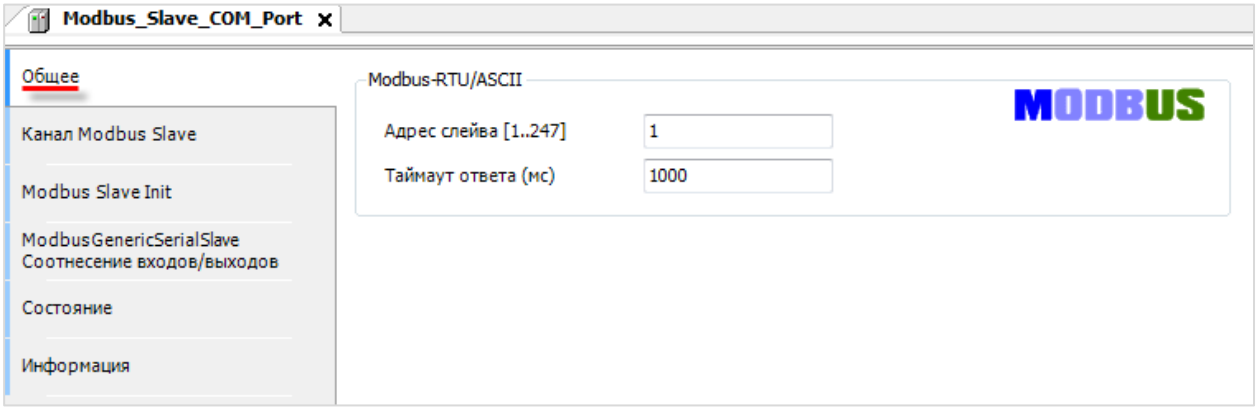


Рисунок 4.2.6 – Настройки компонента Modbus Slave, вкладка Общее

В настройках компонента на вкладке **Общее** следует указать адрес slave-устройства. В случае необходимости можно указать индивидуальный таймаут ответа – он будет иметь приоритет по сравнению с таймаутом, установленным в настройках **Modbus Master** (см. рисунок 4.2.4).

ПРИМЕЧАНИЕ
Диапазон доступных адресов slave-устройств – **1...247**. Широковещательная рассылка через адрес **0** не поддерживается.

На вкладке **Канал Modbus Slave** происходит добавление каналов slave-устройства. Канал является структурной единицей обмена, определяющей тип и число последовательно расположенных регистров slave-устройства и применяемую к ним операцию (чтение/запись). Максимальное число каналов для одного устройства – **100**. Для создания нового канала следует нажать кнопку **Добавить канал**, после чего определить его настройки:

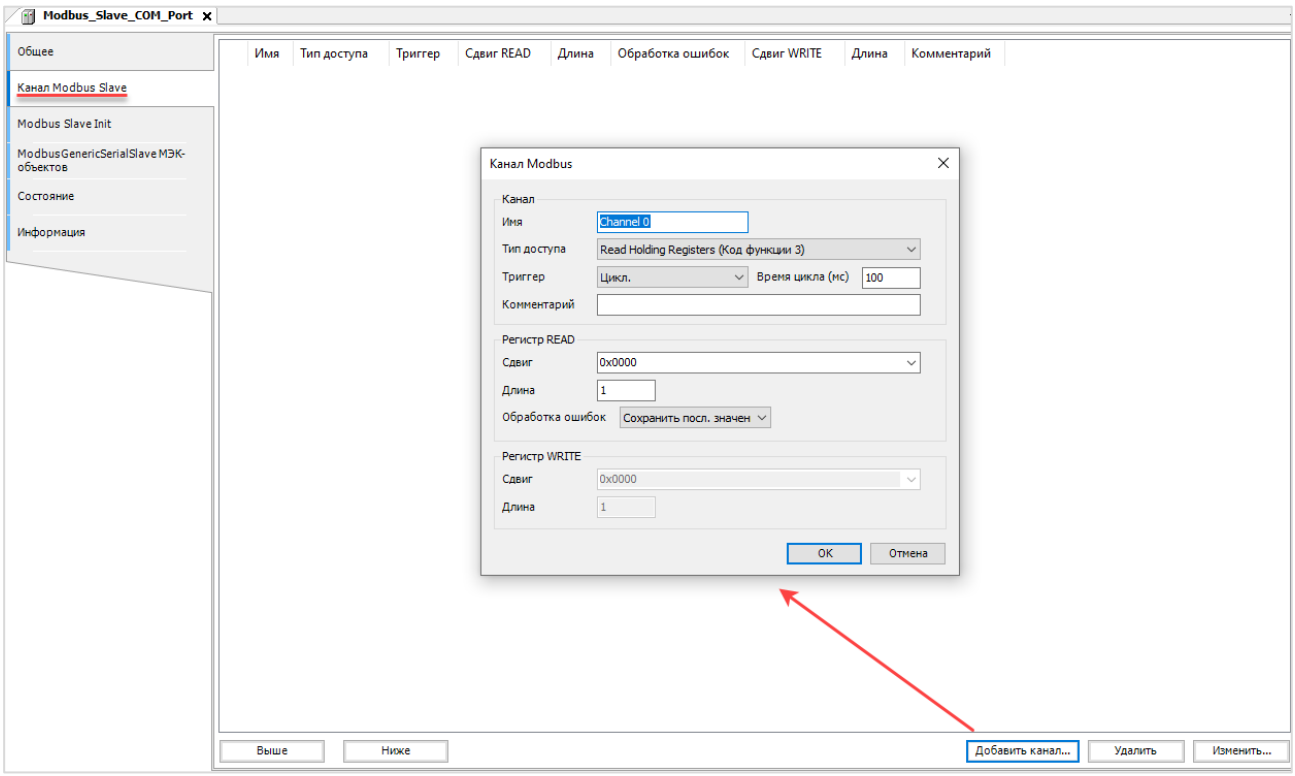


Рисунок 4.2.7 – Добавление канала Modbus Slave

После создания канала его можно отредактировать с помощью кнопки Изменить и переместить в списке добавленных каналов с помощью кнопок **Выше/Ниже**.

Таблица 4.2.3 – Параметры канала Modbus Slave

Параметр	Описание
Имя	Название канала
Тип доступа	Функция, применяемая к регистрам slave-устройства (см. таблицу 2.2);
Триггер	Тип обращения к регистрам slave-устройства: циклически, по переднему фронту заданной логической переменной или из кода приложения
Время цикла	Желательный период опроса канала slave-устройства (для триггера типа **циклический**). Должен быть кратен времени цикла задачи, к которой привязан обмен и быть не меньше, чем интервал ее вызова. Также время цикла должно выбираться в зависимости от опрашиваемого устройства – например, для модулей **MB110-8A** время обновления данных одного канала для термопары типа ТХК составляет 0.4 секунды и, соответственно, разумное время цикла в секундах не должно быть меньше этого значения. Фактический период опроса может быть меньше желаемого (например, из-за наличия в проекте большого числа каналов опроса)
Комментарий	Описание канала
Сдвиг	Номер регистра или первого из последовательности регистров (для операций группового чтения/записи), к которым применяется заданная функция. Можно вводить как в десятичном, так и в шестнадцатеричном виде (например, **0x00FF** или **16#00FF**)
Длина	Количество регистров, к которым применяется заданная функция (для операций группового чтения/записи)
Обработка ошибок	Операция, выполняемая со значениями канала при возникновении ошибки обмена (только для считываемых регистров) – сохранение последнего значения или обнуление

Ниже приведен пример конфигурации двух каналов **Modbus Slave**:

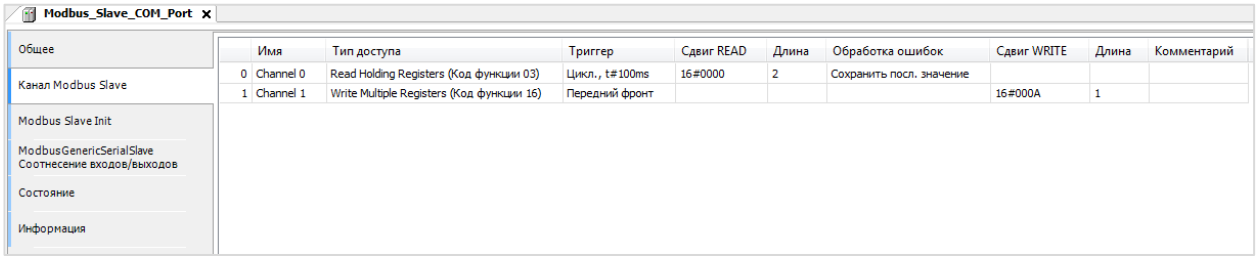


Рисунок 4.2.8 – Пример настройки каналов Modbus Slave

В данном случае master-устройство каждые **100 мс** будет опрашивать нулевой и первый holding регистры slave-устройства и по переднему фронту триггерной переменной записывать значение в десятый (16#000A=10#10) holding регистр slave-устройства.

На вкладке **Modbus Slave Init** можно указать команды записи, однократно выполняемые при запуске проекта.

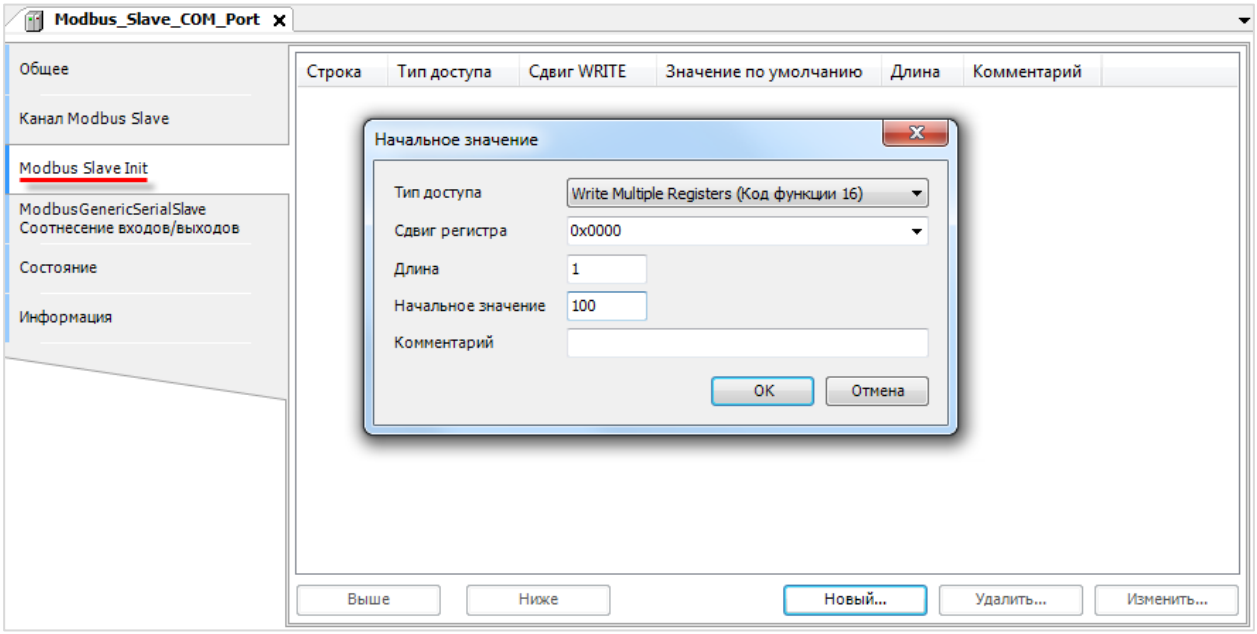


Рисунок 4.2.9 – Настройки вкладки Modbus Slave Init

На вкладке **ModbusGenericSerialSlave Соотнесение входов/выходов** осуществляется привязка переменных программы к каналам Modbus. Стандарт Modbus определяет использование двух типов данных: **BOOL** и **WORD.** Пользователь должен привязать к каждому регистру канала переменную соответствующего типа, либо привязать непосредственно к каналу массив переменных соответствующего типа. К каждому из битов **WORD** переменной можно также привязать **BOOL** переменную (для считываемых переменных эта привязка не исключает привязку WORD переменной, для записываемых – исключает).

ПРИМЕЧАНИЕ
Для корректного обновления данных в компоненте во вкладке **Всегда обновлять переменные** следует установить значение **Включено 2 (Всегда в задаче цикла шины)**.

ПРИМЕЧАНИЕ
Настоятельно рекомендуется не редактировать значения в столбце **Адрес** из-за возможных конфликтов используемой памяти. Более подробная информация приведена по ссылке.

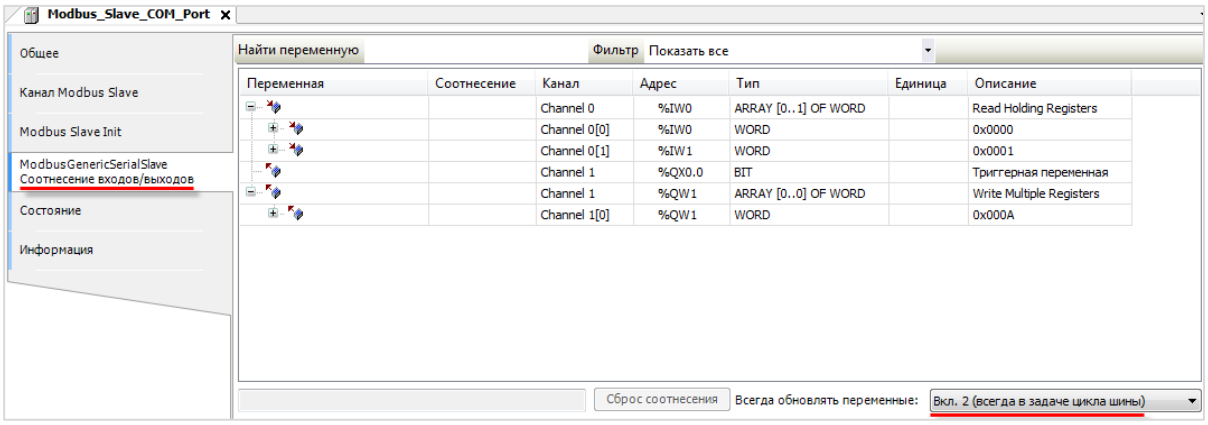


Рисунок 4.2.10 – Настройки вкладки ModbusGenericSerialSlave Соотнесение входов/выходов

Для привязки переменных следует два раза нажать **ЛКМ** на ячейку столбца **Переменная**, после чего выбрать необходимую переменную проекта с помощью **Ассистента ввода** (или ввести имя переменной вручную):

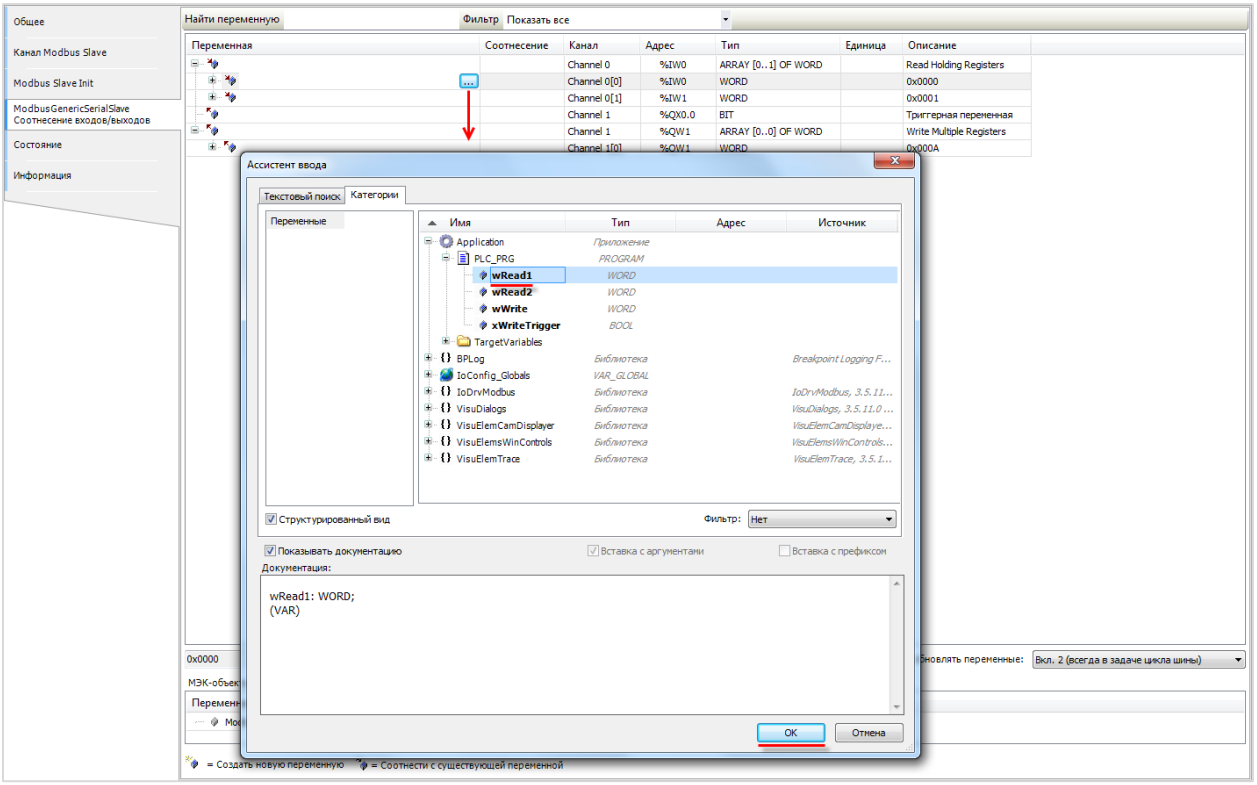


Рисунок 4.2.11 – Привязка переменных программы к каналам Modbus Slave

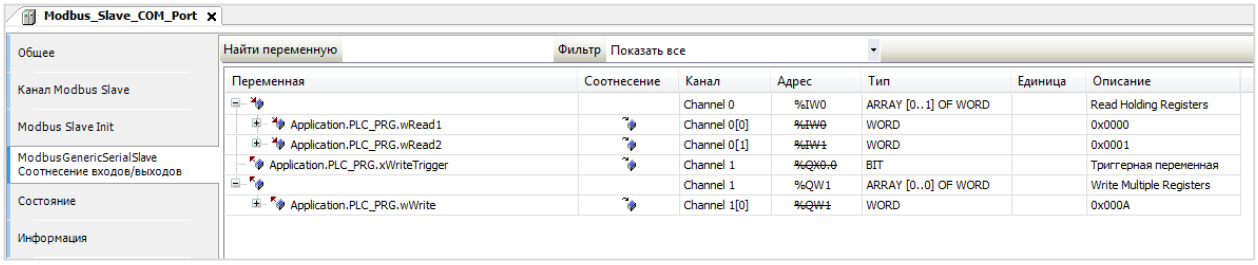


Рисунок 4.2.12 – Привязка переменных программы к каналам Modbus Slave

Пример настройки контроллера в режиме **Modbus Serial Master** для опроса модулей **Mx110** приведен в п. 4.9.

4.3 Настройка контроллера в режиме Modbus RTU Slave

Для настройки контроллера в режиме **Modbus RTU Slave** следует:

1. Нажать **ПКМ** на компонент **Device** и добавить компонент **Modbus COM**, расположенный во вкладке **Промышленные сети/Modbus/Порт Modbus Serial**.

ПРИМЕЧАНИЕ

Версия компонента должна соответствовать версии таргет-файла. Для отображения всех доступных версий компонента следует установить галочку **Отображать все версии**. См. рекомендации в приложении А.

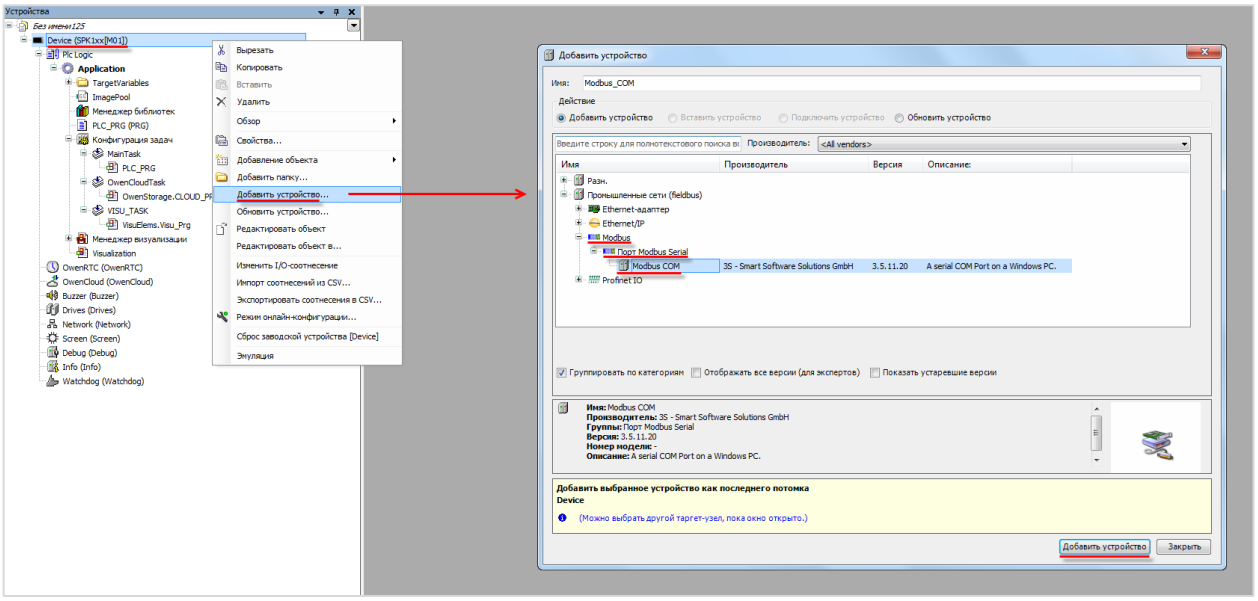


Рисунок 4.3.1 – Добавление компонента Modbus COM

Настройки компонента описаны в п. 4.2 (пп. 1).

2. Нажать **ПКМ** на компонент **Modbus COM** и добавить компонент **Modbus Serial Device**, расположенный во вкладке **Промышленные сети/Modbus/Устройство Modbus Serial**.

ПРИМЕЧАНИЕ

Версия компонента должна соответствовать версии таргет-файла. Для отображения всех доступных версий компонента следует установить галочку **Отображать все версии**. См. рекомендации в приложении А.

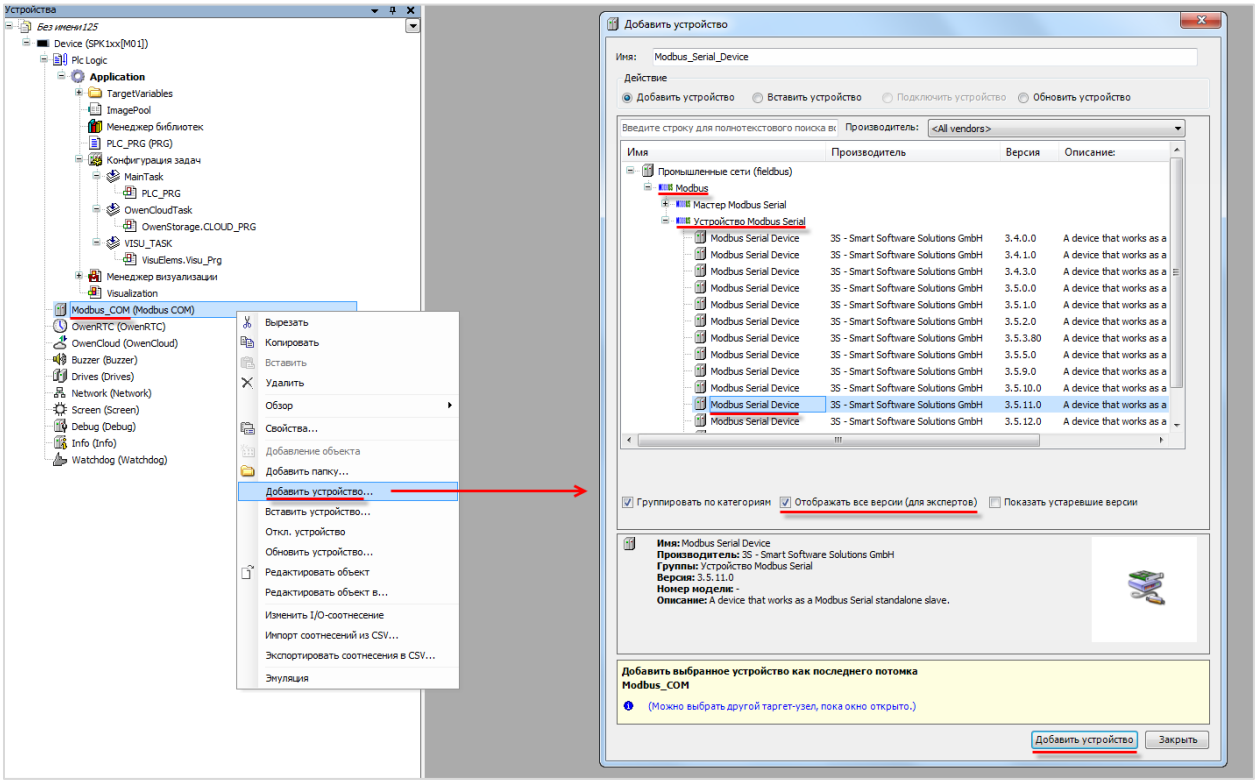


Рисунок 4.3.2 – Добавление компонента Modbus Serial Device

На вкладке **Modbus Serial Device** следует указать настройки slave-устройства:

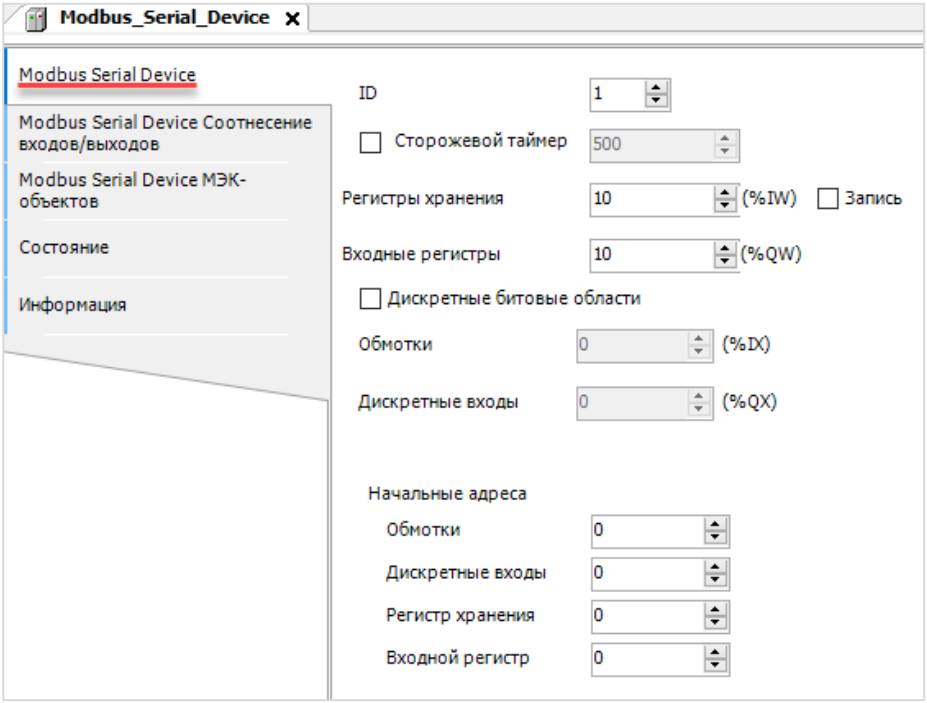


Рисунок 4.3.3 – Настройки компонента Modbus Serial Device

ПРИМЕЧАНИЕ

Компонент не поддерживает протокол **Modbus ASCII**. В случае необходимости работы по этому протоколу следует использовать ФБ MB_SerialSlave из библиотеки **OwenCommunication**.

Таблица 4.3.1 – Настройки компонента Modbus Serial Device

Параметр	Описание
ID	Адрес (**Slave ID**) контроллера в рамках выбранного COM-порта
Сторожевой таймер	Время ожидания (в мс) запроса от master-устройства. Если за это время запроса не приходит, то данные в регистрах обнуляются. В случае отсутствия галочки обнуления данных не происходит
Регистр хранения	Количество **holding** регистров для данного slave-устройства (**2…500**)
Запись	В случае установки галочки значения **coils/holding** регистров slave-устройства можно будет изменять со стороны программы ПЛК. В случае отсутствия галочки изменение значений **coils/holding** регистров возможно только со стороны master-устройства
Входные регистры	Количество **input** регистров для данного slave-устройства (**2…500**)
Дискретные битовые области	В случае установки галочки в slave-устройстве будет использоваться модель данных с 4 независимыми областями памяти (**coils/discrete inputs/input регистры/holding регистры**). В случае отсутствия галочки в slave-устройстве будет использоваться модель данных, в которой области памяти битов и регистров являются общими: область **coils** наложена на область **holding регистров**, а область **discrete inputs** наложена на область **input регистров**. При этом области памяти holding регистров и input регистров являются независимыми
Обмотки	Количество **coils** для данного slave-устройства (**1…65535**)
Дискретные входы	Количество **discrete inputs** для данного slave-устройства (**1…65535**)
Начальные адреса	Начальный адрес для каждой области памяти Modbus. В случае получения запроса к регистру, адрес которого меньше, чем адрес начального регистра, контроллер вернет ошибку **02 (ILLEGAL_DATA_ADDRESS)**

ПРИМЕЧАНИЕ
Holding регистры обозначаются как каналы типа **Входы.**
Input регистры обозначаются как каналы типа **Выходы**.

ПРИМЕЧАНИЕ
В случае установки галочки **Запись** значения переменных, привязанных к каналам **Holding** регистров и **Coils**, нельзя будет изменить из визуализации – потребуется объявить промежуточные переменные, которые будут записываться из визуализации, и генерировать в визуализации команду копирования значений промежуточных переменных в переменные, привязанные к каналам компонента. Это связано с тем, что обработка компонентов Modbus и визуализации выполняется в разных задачах проекта.

ПРИМЕЧАНИЕ
В случае установки галочки **Запись** значения каналов **Holding** регистров и **Coils** нельзя будет изменить в режиме онлайн-отладки в компоненте **Modbus Serial Device** – потребуется изменить значения переменных, привязанных к этим каналам (в POU, где объявлены переменные или в списке просмотра, открываемом через меню **Вид**).

ПРИМЕЧАНИЕ
Адреса регистров могут быть определены по числу в квадратных скобках в столбце **Канал** (в случае использования начальных адресов по умолчанию). **Пример:** Входы[3] – holding регистр с адресом 3, Выходы[4] – input регистр с адресом 4. Адреса бит регистров (в случае отсутствия галочки **Дискретные битовые области**) вычисляются по формуле: *адрес бита = номер регистра · 16 + номер бита в регистре*.

ПРИМЕЧАНИЕ

Начиная с версии **3.5.16.0** изменена адресация бит регистров из-за изменения порядка байт. Пример: бит 0 holding-регистра 0 теперь соответствует coil с адресом 8, бит 1 регистра 0 соответствует coil с адресом 9, бит 8 регистра 0 соответствует coil с адресом 0 и т.д. Это не касается нумерации бит областей **coils/discrete inputs** в случае установки галочки **Дискретные битовые области** – в них нумерация остается «прямой» (бит 0 – coil 0 и т.д.).

ПРИМЕЧАНИЕ

В версиях ниже **3.5.14.0** компонент не поддерживает функцию **05 (Write Single Coil)**.

На вкладке **Modbus Serial Slave Соотнесение входов/выходов** осуществляется привязка переменных программы к регистрам slave-устройства. Стандарт **Modbus** определяет использование двух типов данных: **BOOL** и **WORD**. Пользователь должен привязать к каждому регистру канала переменную соответствующего типа либо привязать непосредственно к каналу массив переменных соответствующего типа. К каждому из битов **WORD** переменной можно также привязать **BOOL** переменную (для holding регистров эта привязка не исключает привязку **WORD** переменной, для input регистров – исключает).

ПРИМЕЧАНИЕ

Для корректного обновления данных во вкладке **Всегда обновлять переменные** следует установить значение **Включено 2 (Всегда в задаче цикла шины)**.

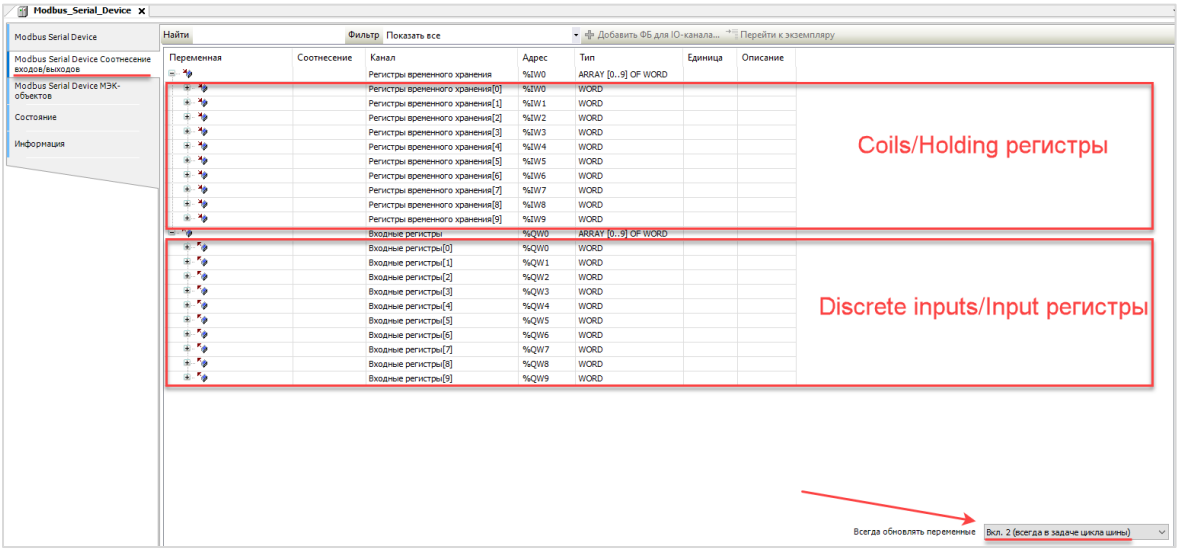


Рисунок 4.3.4 – Настройки вкладки Modbus Serial Slave Соотнесение входов/выходов (без установленной галочки Дискретные битовые области)

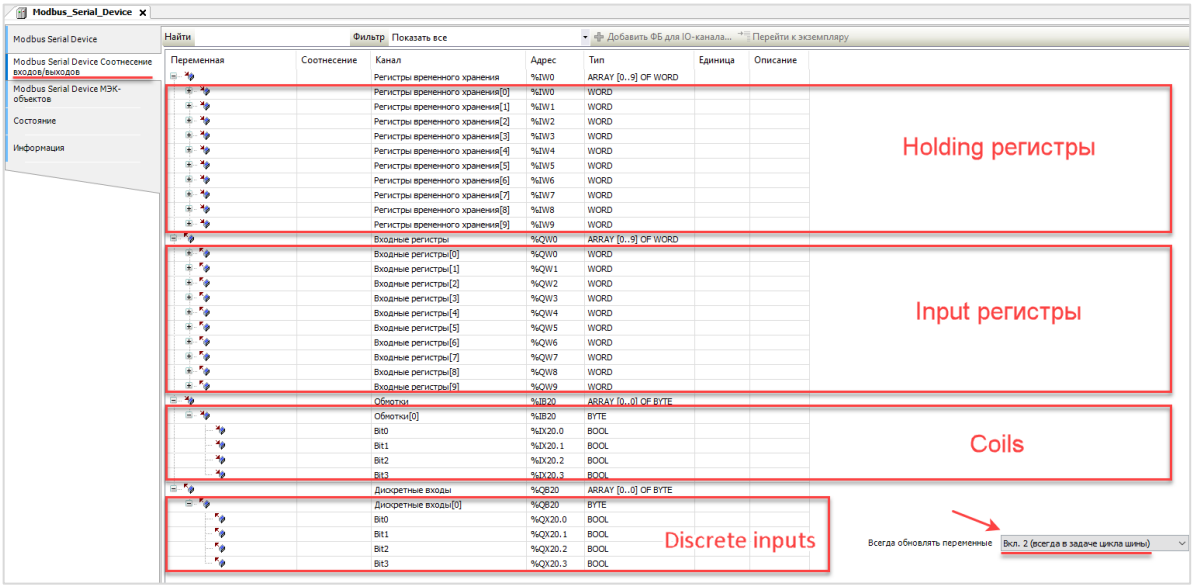
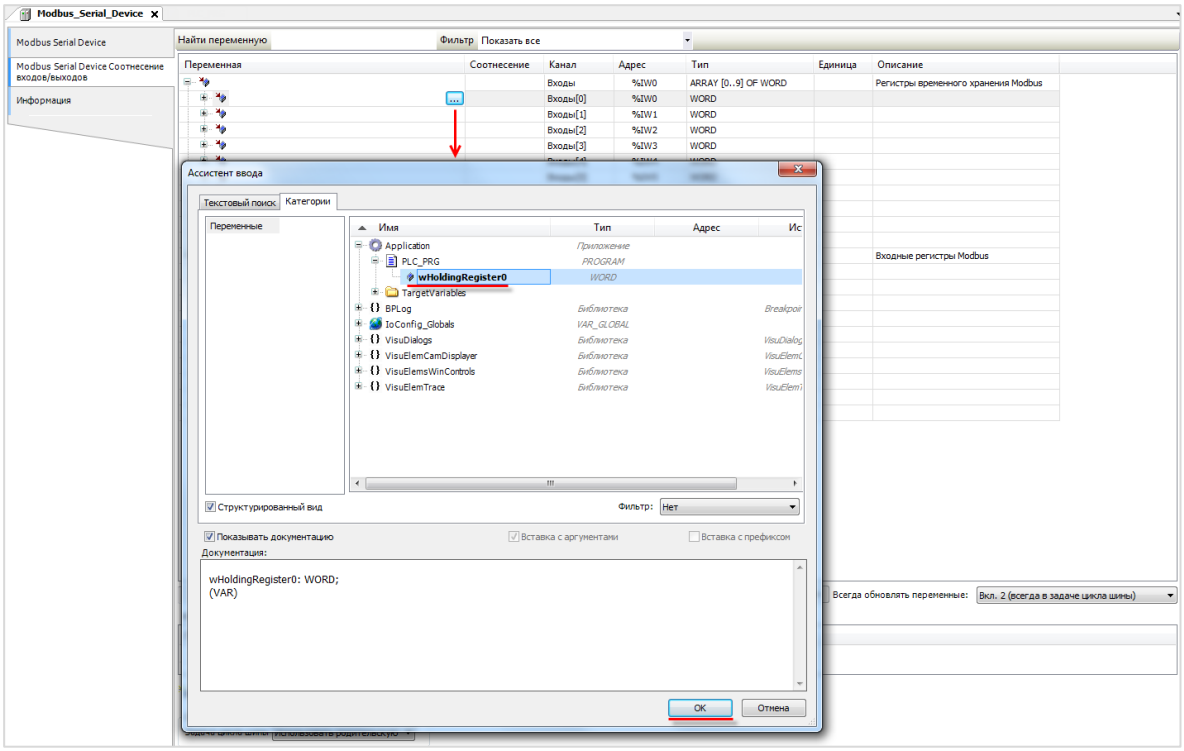


Рисунок 4.3.4 – Настройки вкладки Modbus Serial Slave Соотнесение входов/выходов (с установленной галочкой Дискретные битовые области)

Для привязки переменных следует два раза нажать **ЛКМ** на ячейку столбца **Переменная**, после чего выбрать необходимую переменную проекта с помощью **Ассистента ввода** (или ввести ее имя вручную):



Рисунка 4.3.6 – Привязка переменных программы к регистрам Modbus RTU Slave

Пример настройки контроллера как **Modbus RTU Slave** приведен в п. 4.10.

4.4 Настройка контроллера в режиме Modbus TCP Master

Для настройки контроллера в режиме **Modbus TCP Master** следует:

1. Нажать **ПКМ** на компонент **Device** и добавить компонент **Ethernet**, расположенный во вкладке **Промышленные сети/Ethernet-адаптер**.

ПРИМЕЧАНИЕ

Версия компонента должна соответствовать версии таргет-файла. Для отображения всех доступных версий компонента следует установить галочку **Отображать все версии**. См. рекомендации в приложении А.

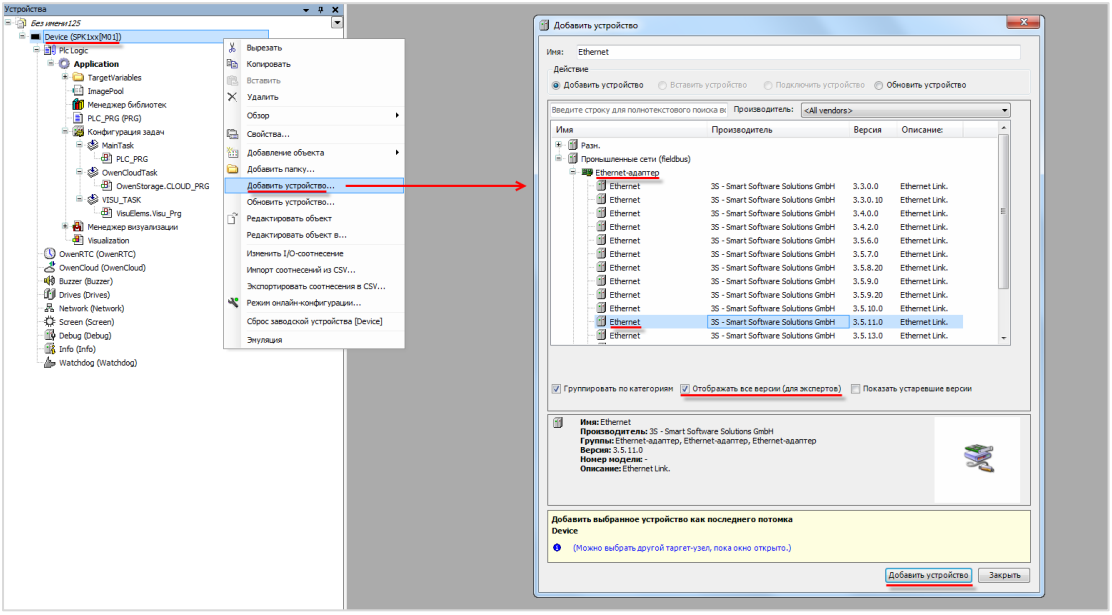


Рисунок 4.4.1 – Добавление компонента Ethernet

Затем следует установить соединение с контроллером, не загружая в него проект (**Device – Установка соединения – Сканировать сеть**) и в компоненте **Ethernet** на вкладке **Конфигурация Ethernet** выбрать нужный интерфейс.

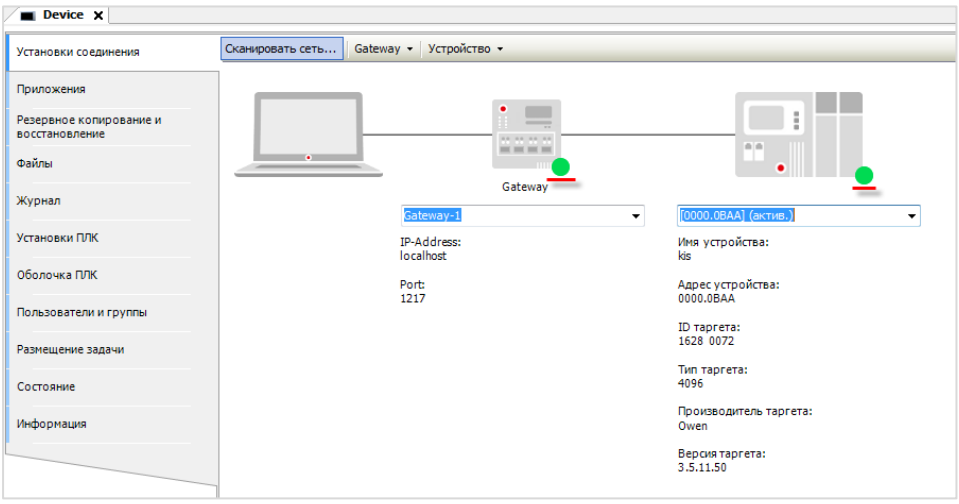


Рисунок 4.4.2 – Подключение к контроллеру

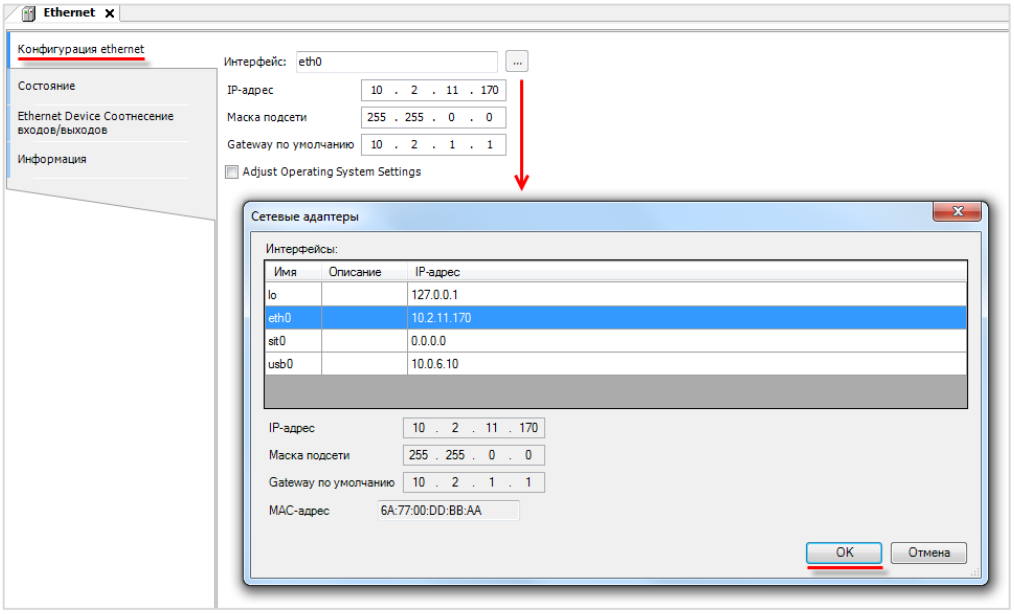


Рисунок 4.4.3 – Выбор используемого интерфейса

ПРИМЕЧАНИЕ
Настройки интерфейса задаются в конфигураторе контроллера (см. документ **CODESYS V3.5. FAQ**).

ПРИМЕЧАНИЕ
В случае установки галочки **Adjust Operating System Settings** пользователь может изменить настройки интерфейса. После загрузки проекта в контроллер эти настройки будут применены в операционной системе контроллера. Контроллеры ОВЕН **не поддерживают** данный функционал.

2. Нажать **ПКМ** на компонент **Ethernet** и добавить компонент **Modbus TCP Master**, расположенный во вкладке **Промышленные сети/Modbus/Мастер Modbus TCP**.

ПРИМЕЧАНИЕ
Версия компонента должна соответствовать версии таргет-файла. Для отображения всех доступных версий компонента следует установить галочку **Отображать все версии**. См. рекомендации в приложении А.

ПРИМЕЧАНИЕ
В компонент **Ethernet** может быть добавлено произвольное число компонентов **Modbus TCP Master**.

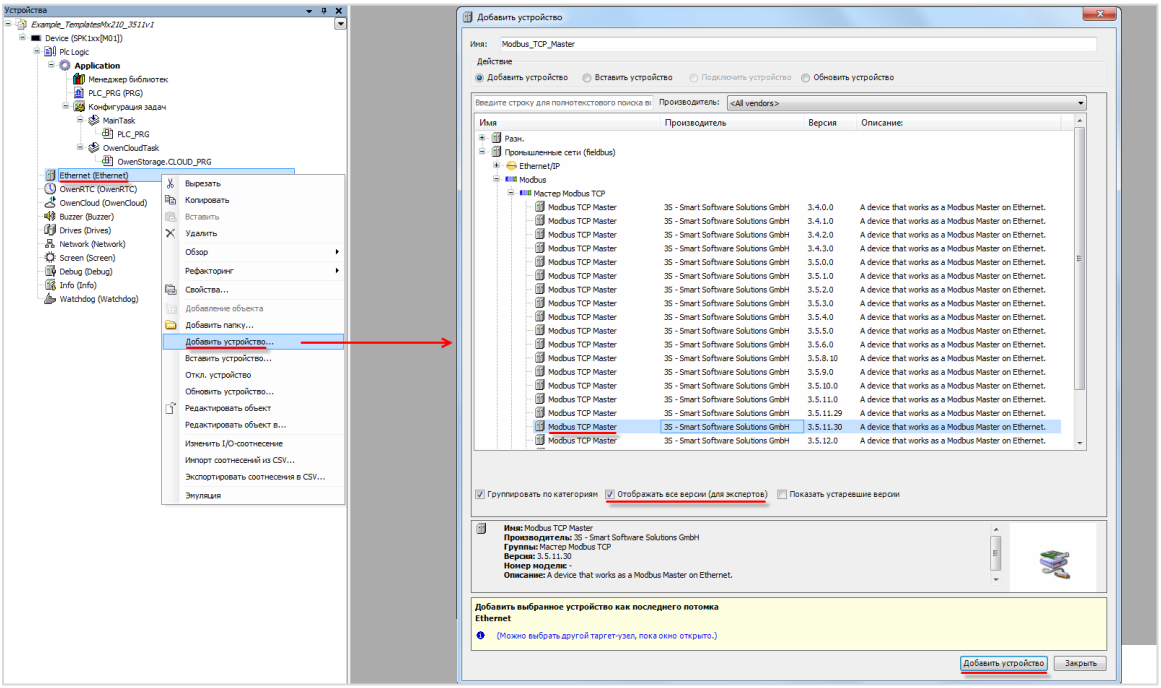


Рисунок 4.4.4 – Добавление компонента Modbus TCP Master

В настройках компонента на вкладке **Общее** следует задать настройки master-устройства.

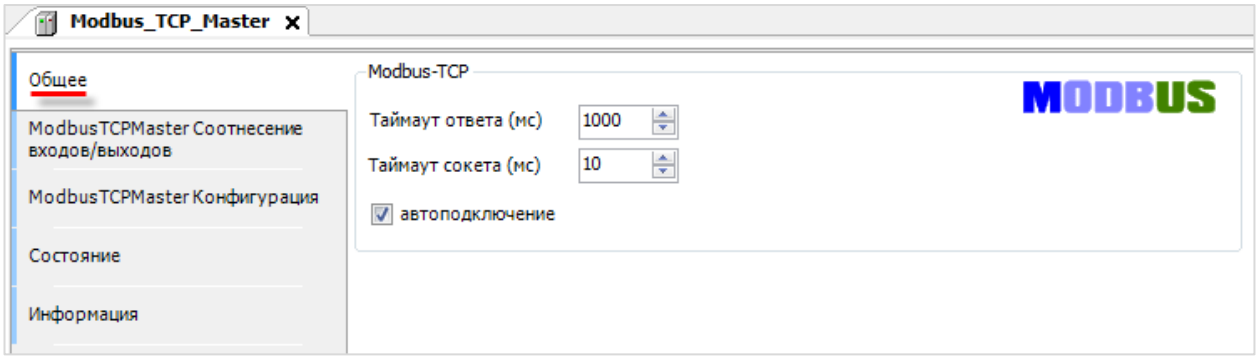


Рисунок 4.4.5 – Настройки компонента Modbus TCP Master

Таблица 4.4.1 – Настройки компонента Modbus TCP Master

Параметр	Описание
Таймаут ответа	Время (в мс), в течение которого master ожидает ответа slave-устройства. В случае отсутствия ответа по истечению этого времени master-устройство делает паузу на **время между фреймами** и переходит к опросу следующего канала slave-устройства (или следующему slave-устройству). Значение, введенное здесь, будет по умолчанию использоваться для всех slave-устройств. На вкладке **Конфигурация Modbus Slave** (см. рисунок 4.4.7) для каждого устройства можно задать индивидуальный таймаут отклика
Таймаут сокета	Время (в мс), в течение которого master ожидает TCP/IP пакеты от slave-устройства. В случае отсутствия пакетов по истечению этого времени соединение с устройством разрывается
Автоподключение	В случае отсутствия галочки не ответившее slave-устройство исключается из дальнейшего опроса. **Настоятельно рекомендуется** всегда включать эту опцию

3. Нажать **ПКМ** на компонент **Modbus TCP Master** и добавить компонент **Modbus TCP Slave**, расположенный во вкладке **Промышленные сети/Modbus/Слейв Modbus TCP**. Число компонентов должно соответствовать числу slave-устройств. Максимальное возможное количество slave-устройств для одного master-устройства – **32** (начиная с версии 3.5.13.0 – **64**).

ПРИМЕЧАНИЕ

Версия компонента должна соответствовать версии таргет-файла. Для отображения всех доступных версий компонента следует установить галочку **Отображать все версии**. См. рекомендации в приложении А.

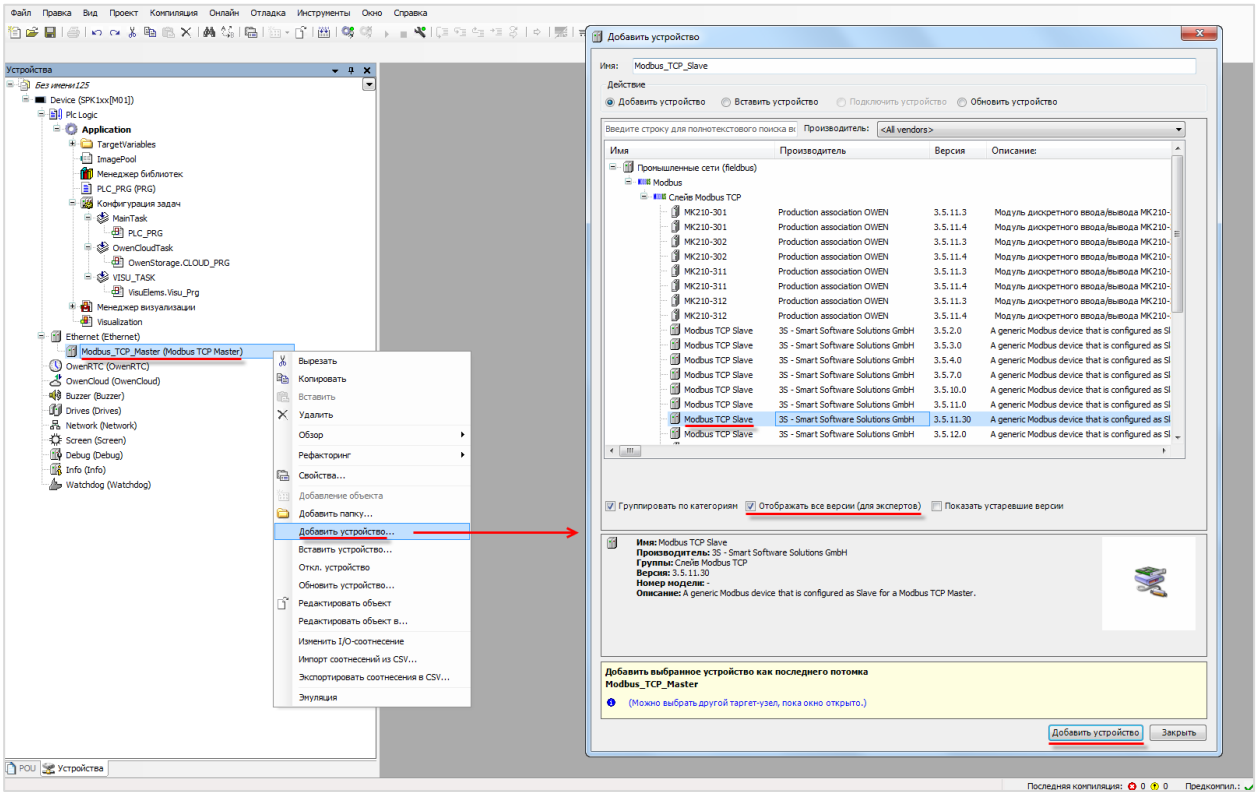


Рисунок 4.4.6 – Добавление компонента Modbus TCP Slave

В настройках компонента на вкладке **Общее** следует указать IP-адрес и порт slave-устройства. В случае необходимости можно указать индивидуальный таймаут ответа – он будет иметь приоритет по сравнению с таймаутом, установленным в настройках **Modbus TCP Master** (см. рисунок 4.4.5).

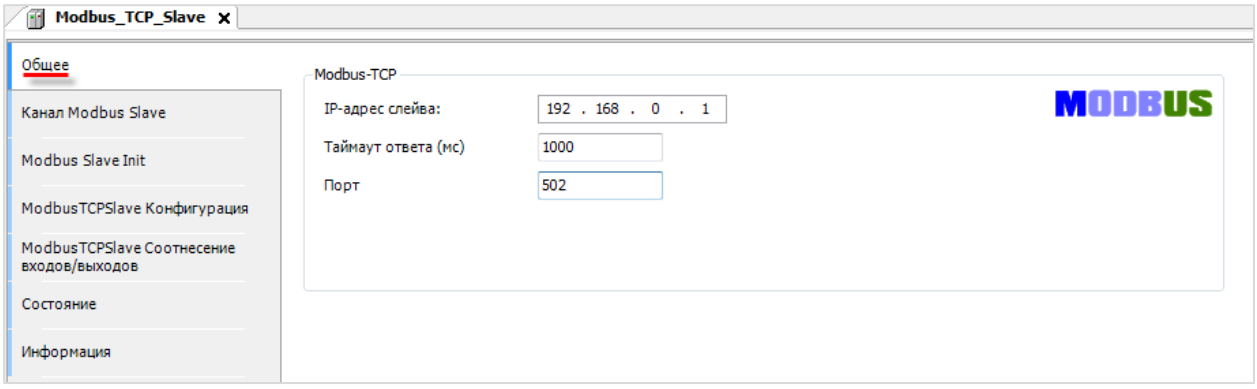


Рисунок 4.4.7 – Настройки компонента Modbus TCP Slave, вкладка Общее

Настройки вкладки **Общее** дублируются на вкладке **ModbusTCPSlave Конфигурация.** На данной вкладке также можно задать адрес (**Unit ID**) slave-устройства – это требуется в тех случаях, когда производится опрос устройства через **шлюз Modbus TCP/Modbus Serial** или если устройство не отвечает на запросы, в которых **Unit ID** имеет значение, предусмотренное спецификацией Modbus по умолчанию (**16#FF**).

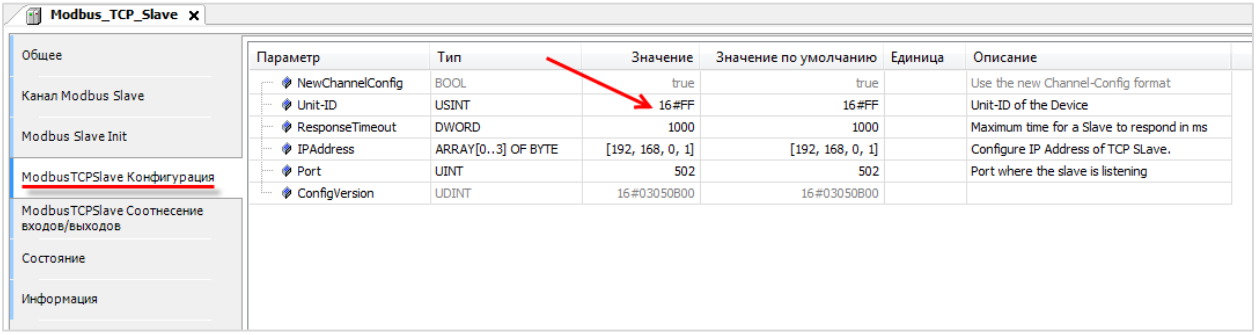


Рисунок 4.4.8 – Настройки компонента Modbus TCP Slave, вкладка ModbusTCPSlave Конфигурация

Настройки вкладок **Канал Modbus Slave** и **Modbus Slave Init** идентичны настройкам одноименных вкладок компонента **Modbus Slave** и описаны в п. 4.2 (пп. 3).

4.5 Настройка контроллера в режиме Modbus TCP Slave

Для настройки контроллера в режиме **Modbus TCP Slave** следует:

1. Нажать **ПКМ** на компонент **Device** и добавить компонент **Ethernet**, расположенный во вкладке **Промышленные сети/Ethernet-адаптер**.

ПРИМЕЧАНИЕ
Версия компонента должна соответствовать версии таргет-файла. Для отображения всех доступных версий компонента следует установить галочку **Отображать все версии**. См. рекомендации в приложении А.

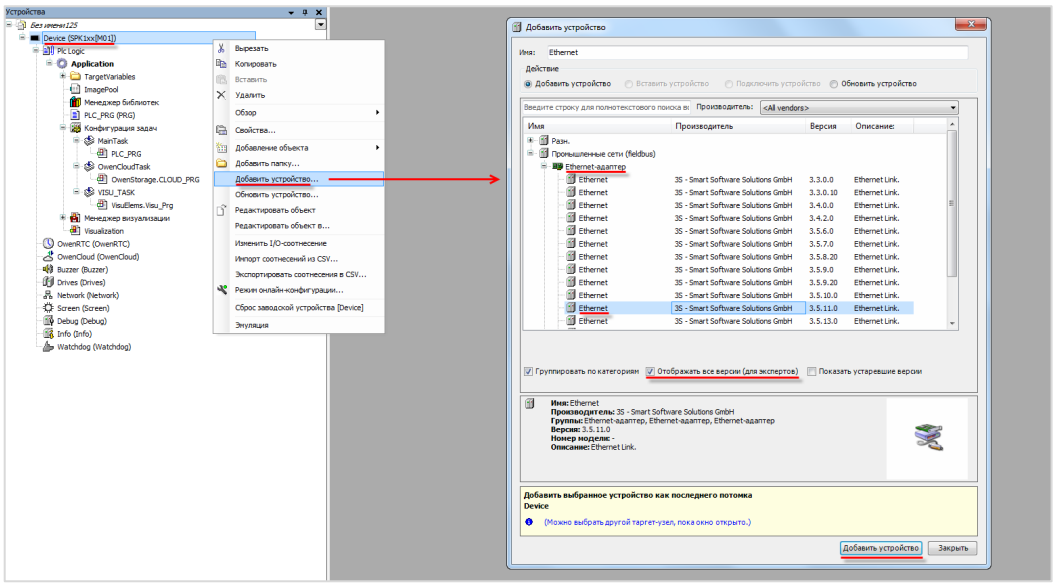


Рисунок 4.5.1 – Добавление компонента Ethernet

Настройки компонента описаны в п. 4.4 (пп. 1).

2. Нажать **ПКМ** на компонент **Ethernet** и добавить компонент **Modbus TCP Slave Device**, расположенный во вкладке **Промышленные сети/Modbus/Слейв-устройство Modbus TCP**.

ПРИМЕЧАНИЕ
Версия компонента должна соответствовать версии таргет-файла. Для отображения всех доступных версий компонента следует установить галочку **Отображать все версии**. См. рекомендации в приложении А.

ПРИМЕЧАНИЕ
В компонент **Ethernet** может быть добавлено произвольное число компонентов **Modbus TCP Slave Device**.

ПРИМЕЧАНИЕ
Компонент поддерживает одновременное подключение до 64 клиентов (в версии **3.5.16.0** и выше).

ПРИМЕЧАНИЕ
В версии **3.5.16.x** компонент отвечает только на запросы с **Unit ID = 0** и **255**. В более ранних и поздних (**3.5.17.0** и выше) версиях компонент отвечает на запросы с любыми **Unit ID**.

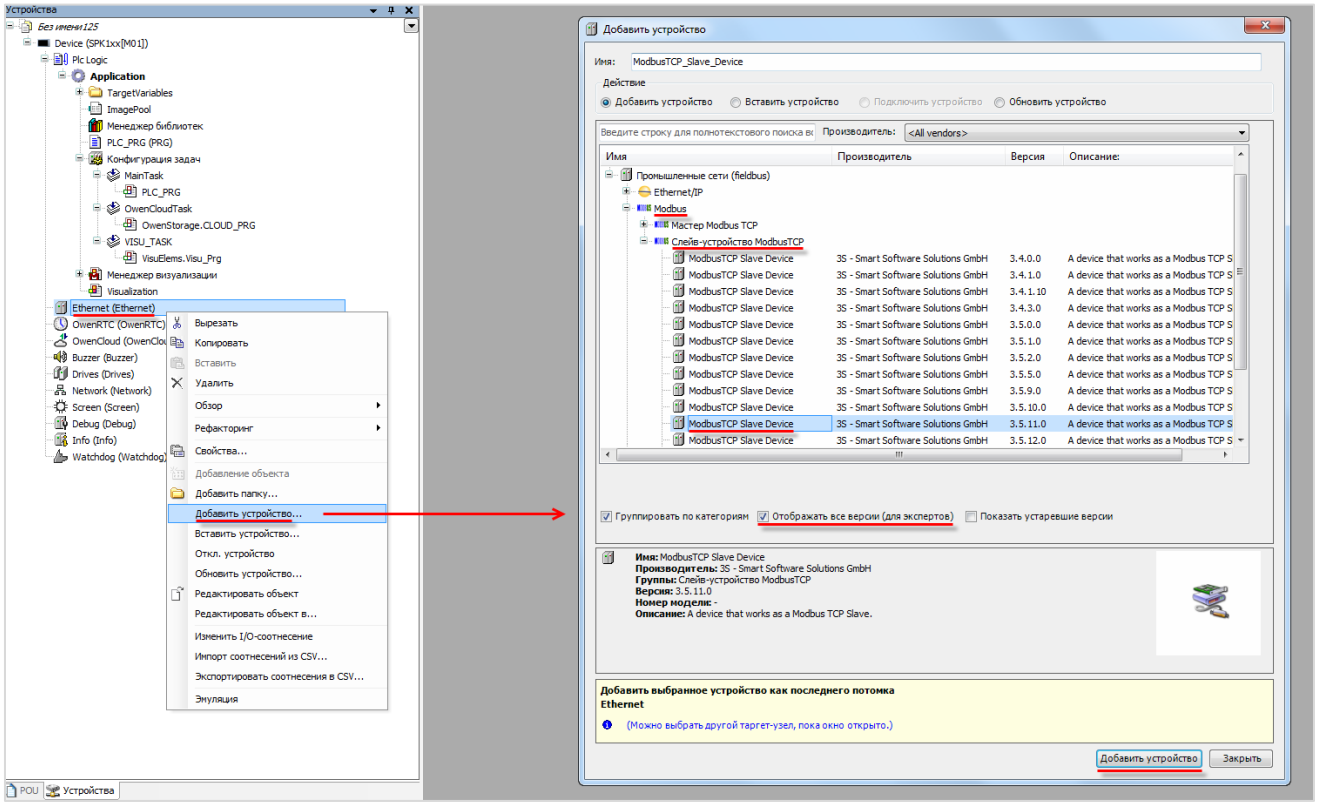


Рисунок 4.5.2 – Добавление компонента Modbus TCP Slave Device

На вкладке **Страница конфигурации** следует указать настройки slave-устройства:

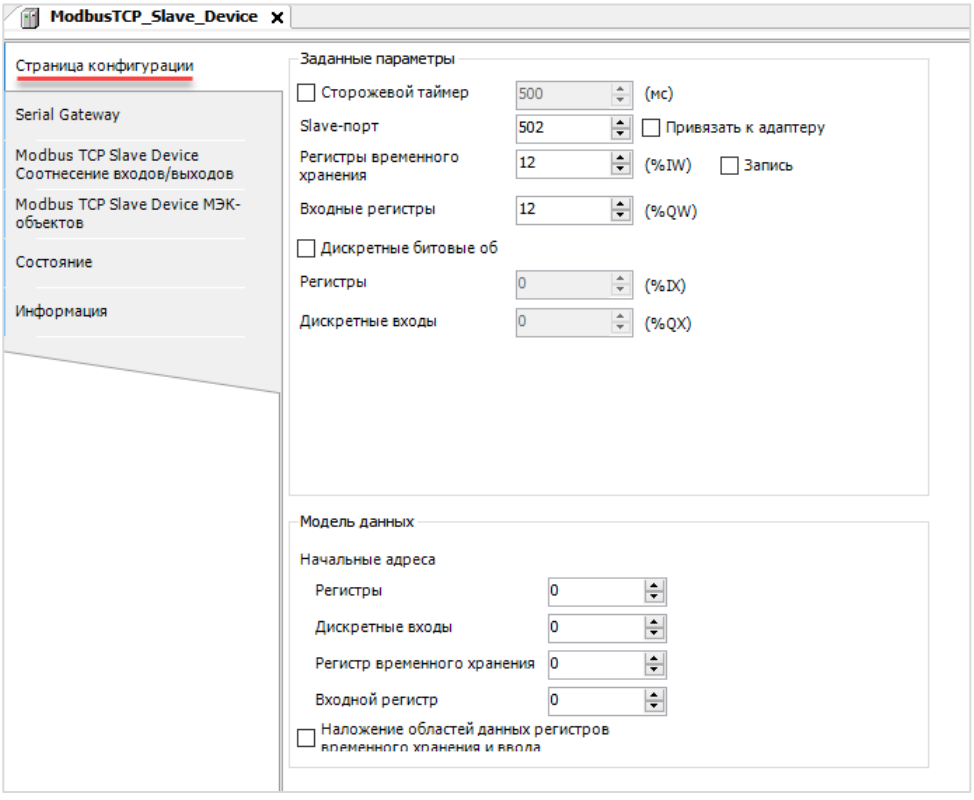


Рисунок 4.5.3 – Настройки компонента Modbus TCP Slave Device

Таблица 4.5.1 – Настройки компонента Modbus TCP Slave Device

Параметр	Описание
Сторожевой таймер	Время ожидания (в мс) запроса от master-устройства. Если за это время запроса не приходит, то данные в **holding** регистрах обнуляются. В случае отсутствия галочки обнуления данных не происходит
Slave-порт	Порт, используемый для обмена (по умолчанию – **502**)
Привязать к адаптеру	В случае установки галочки slave будет доступен только по сетевому интерфейсу, выбранному в компоненте Ethernet. В случае отсутствия галочки slave будет доступен по всем интерфейсам устройства
Регистр хранения	Количество **holding регистров** для данного slave-устройства (**2…4096**)
Запись	В случае установки галочки значения **coils/holding** регистров slave-устройства можно будет изменять со стороны программы ПЛК. В случае отсутствия галочки изменение значений **coils/holding** регистров возможно только со стороны master-устройства
Входные регистры	Количество **input регистров** для данного slave-устройства (**2…4096**)
Дискретные битовые области	В случае установки галочки в slave-устройстве будет использоваться модель данных с 4 независимыми областями памяти (**coils/discrete inputs/input регистры/holding регистры**). В случае отсутствия галочки в slave-устройстве будет использоваться модель данных, в которой области памяти битов и регистров являются общими: область **coils** наложена на область **holding регистров**, а область **discrete inputs** наложена на область **input регистров**. При этом области памяти holding регистров и input регистров являются независимыми
Регистры	Количество **coils** для данного slave-устройства (**1…65535**)
Дискретные входы	Количество **discrete inputs** для данного slave-устройства (**1…65535**)
Начальные адреса	Начальный адрес для каждой области памяти Modbus. В случае получения запроса к регистру, адрес которого меньше, чем адрес начального регистра, контроллер вернет ошибку **02 (ILLEGAL_DATA_ADDRESS)**. В случае установки галочки **Наложение областей данных регистров временного хранения и ввода** при считывании master-устройством holding регистров контроллера будут возвращаться значения соответствующих (совпадающих по номерам) input регистров

На вкладке **Serial Gateway** можно настроить шлюз протоколов **Modbus TCP/Modbus RTU**. Если установлена галочка **Serial Gateway Active**, то запросы от **Modbus TCP Master**, опрашивающего контроллер, будут преобразованы в запросы **Modbus RTU** и отправлены в выбранный на вкладке COM-порт на заданной скорости (этот COM-порт не должен использоваться в других компонентах проекта). Поддерживается только режим настроек **8-N-1**. Ответы от **Modbus RTU Slave-устройств**, подключенных к COM-порту, будут преобразованы в **Modbus TCP** и отправлены обратно master-устройству.

ПРИМЕЧАНИЕ
Holding регистры обозначаются как каналы типа **Входы.** Input регистры обозначаются как каналы типа **Выходы**.

ПРИМЕЧАНИЕ
В случае установки галочки **Запись** значения каналов **Holding** регистров и **Coils** нельзя будет изменить в режиме онлайн-отладки в компоненте **Modbus TCP Slave Device** – потребуется изменить значения переменных, привязанных к этим каналам (в POU, где объявлены переменные или в списке просмотра, открываемом через меню **Вид – Просмотр**).

ПРИМЕЧАНИЕ
В случае установки галочки **Запись** значения переменных, привязанных к каналам **Holding** регистров и **Coils**, нельзя будет изменить из визуализации – потребуется объявить промежуточные переменные, которые будут записываться из визуализации, и генерировать в визуализации команду копирования значений промежуточных переменных в переменные, привязанные к каналам компонента. Это связано с тем, что обработка компонентов Modbus и визуализации выполняется в разных задачах проекта.

ПРИМЕЧАНИЕ
Адреса регистров могут быть определены по числу в квадратных скобках в столбце **Канал** (в случае использования начальных адресов по умолчанию). **Пример:** Входы[3] – holding регистр с адресом 3, Выходы[4] – input регистр с адресом 4. Адреса бит регистров (в случае отсутствия галочки **Дискретные битовые области**) вычисляются по формуле: *адрес бита = номер регистра · 16 + номер бита в регистре*.

ПРИМЕЧАНИЕ
Начиная с версии **3.5.16.0** изменена адресация бит регистров из-за изменения порядка байт. Пример: бит 0 holding-регистра 0 теперь соответствует coil с адресом 8, бит 1 регистра 0 соответствует coil с адресом 9, бит 8 регистра 0 соответствует coil с адресом 0 и т.д. Это не касается нумерации бит областей **coils/discrete inputs** в случае установки галочки **Дискретные битовые области** – в них нумерация остается «прямой» (бит 0 – coil 0 и т.д.).

Настройки вкладки **Modbus TCP Slave Device Соотнесение входов/выходов** идентичны настройкам одноименной вкладки компонента **Modbus Serial Device** и описаны в п. 4.3 (пп. 2).

4.6 Диагностика и управление обменом

В случае необходимости контролировать процесс обмена данными можно воспользоваться системными переменными компонентов Modbus. В нужном месте программы следует ввести имя компонента из дерева проекта, поставить точку и из выпадающего списка выбрать нужную переменную:

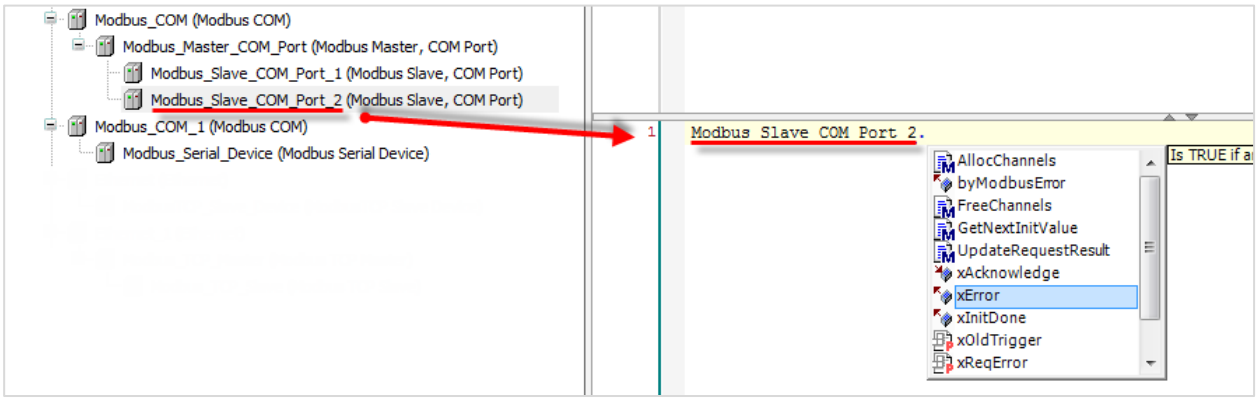


Рисунок 4.6.1 – Использование переменных диагностики в программе

Таблица 4.6.1 – Системные переменные компонентов Modbus

Переменная	Тип	Описание
Компонент **Modbus Master COM Port**		
Входы		
xResetComPort	BOOL	По переднему фронту выполняется переинициализация компонента **Modbus COM**, в который добавлен данный компонент
xStop	BOOL	Если вход имеет значение **TRUE**, то опрос всех slave-устройств данного компонента прекращается
Выходы		
uiNumberOfCom-municatingSlaves	UINT	Число slave-устройств данного компонента, от которых во время последнего сеанса опроса были получены корректные ответы (без кодов ошибок)
xAllSlavesOk	BOOL	**TRUE** – во время последнего сеанса опроса были получены корректные ответы (без кодов ошибок) от всех slave-устройств данного компонента
Компонент **Modbus Slave COM Port**		
Входы		
xAcknowledge	BOOL	По переднему фронту выполняется переинициализация компонента без обнуления выходов **xError** и **byModbusError**

Переменная	Тип	Описание
xDoInit	BOOL	Если вход имеет значение **TRUE**, то по переднему фронту входа **xReset** также происходит повторная отправка команд инициализации, заданных на вкладке **Modbus Slave Init**
xReset	BOOL	По переднему фронту выполняется переинициализация компонента с обнулением выходов **xError** и **byModbusError**
xTrigger	BOOL	По переднему фронту выполняется опрос всех каналов компонента, у которых параметр **Триггер** имеет значение **Передний фронт**
Выходы		
byModbusError	IoDrvModbus. MB_ErrorCodes	Код ошибки обмена со slave-устройством
iChannelIndex	INT	Номер текущего опрашиваемого канала (нумерация с нуля)
xBusy	BOOL	**TRUE** – выполняется опрос канала
xDone	BOOL	**TRUE** – опрос текущего канала успешно завершен
xError	BOOL	Бит ошибки обмена со slave-устройством
xInitDone	BOOL	**TRUE** – отправлены все команды инициализации, заданные на вкладке **Modbus Slave Init**
Компонент Modbus Serial Device		
Входы		
xEnable	BOOL	**TRUE** – включить компонент, **FALSE** – отключить
Выходы		
ErrorCode	UDINT	Код ошибки
xBusy	BOOL	**TRUE** – компонент находится в работе
xInternalError	BOOL	**TRUE** – внутренняя ошибка компонента (например, ошибка выделения памяти)
xOnline	BOOL	**TRUE** – компонент получает запросы от master-устройства
Компонент **Modbus TCP Master**		
Входы		
xStop	BOOL	Если вход имеет значение **TRUE**, то опрос всех slave-устройств данного компонента прекращается
Выходы		
uiNumberOfCom-municatingSlaves	UINT	Число slave-устройств данного компонента, от которых во время последнего сеанса опроса были получены корректные ответы (без кодов ошибок)
xSlaveError	BOOL	**TRUE** – во время последнего сеанса опроса произошла ошибка обмена как минимум с

Переменная	Тип	Описание
		одним из slave-устройств данного компонента (был получен ответ с кодом ошибки или устройство не ответило)
Компонент **Modbus TCP Slave**		
Входы		
xConfirmError	BOOL	По переднему фронту выполняется переинициализация компонента
xDoInit	BOOL	Если вход имеет значение **TRUE**, то по переднему фронту входа **xReset** также происходит повторная отправка команд инициализации, заданных на вкладке **Modbus Slave Init**
Выходы		
byModbusError	IoDrvModbus. MB_ErrorCodes	Код ошибки обмена со slave-устройством
ComSettings	IoDrvModbusTcp. ModbusTCPComSettings	Структура сетевых настроек опрашиваемого slave-устройства (IP-адрес и порт)
ComState	IoDrvModbusTcp. ModbusTCPComState	Статус обмена со slave-устройством
iChannelIndex	INT	Номер текущего опрашиваемого канала (нумерация с нуля)
xBusy	BOOL	**TRUE** – выполняется опрос канала
xDone	BOOL	**TRUE** – опрос текущего канала успешно завершен
xError	BOOL	Бит ошибки обмена со slave-устройством
xInitDone	BOOL	**TRUE** – отправлены все команды инициализации, заданные на вкладке **Modbus Slave Init**
Компонент **Modbus TCP Slave Device**		
Входы		
xEnable	BOOL	**TRUE** – включить компонент, **FALSE** – отключить
Выходы		
Port	UINT	Порт, используемый компонентом
uiClientConnections	UINT	Число активных TCP-соединений компонента
ErrorCode	UDINT	Код ошибки
xBusy	BOOL	**TRUE** – компонент находится в работе
xInternalError	BOOL	**TRUE** – внутренняя ошибка компонента (например, ошибка выделения памяти)
xOnline	BOOL	**TRUE** – компонент получает запросы от master-устройства
xTimeout	BOOL	**TRUE** – сработал таймаут ожидания запросов от master-устройств

Таблица 4.6.2 – Описание элементов перечисления MB_ErrorCodes

Название	Значение	Описание
RESPONSE_SUCCESS	16#0	Отсутствие ошибок обмена
ILLEGAL_ FUNCTION	16#1	Slave-устройство не поддерживает функцию Modbus, указанную в запросе
ILLEGAL_DATA_ ADDRESS	16#2	Slave-устройство не содержит одного или нескольких регистров, указанных в запросе
ILLEGAL_DATA_VALUE	16#3	Данная команда записи является некорректной с точки зрения протокола Modbus
SLAVE_ DEVICE_FAILURE	16#4	Во время выполнения запроса в slave-устройстве произошла внутренняя ошибка
ACKNOWLEDGE	16#5	Slave-устройство приняло запрос и обрабатывает его, но это потребует некоторого времени. Этот ответ предохраняет master-устройство от генерации ошибки таймаута
SLAVE_DEVICE_BUSY	16#6	Slave-устройство занято обработкой другой команды. Master-устройство должно повторить запрос позже, когда slave-устройство освободится
MEMORY_ PARITY_ ERROR	16#8	Произошла ошибка во время использования функции Modbus 20 или 21 (см. более подробную информацию в спецификации протокола)
GATEWAY_PATH_UNAVALIABLE	16#A	Ошибка конфигурации сетевого шлюза Modbus TCP/Modbus RTU – маршрут к slave-устройству не может быть построен
GATEWAY_DEVICE_FAILED_TO_ RESPONSE	16#B	Устройство, расположенное за сетевым шлюзом Modbus TCP/Modbus RTU, не ответило на запрос
RESPONSE_TIMEOUT	16#A1	В течение времен таймаута не был получен ответ от slave-устройства
RESPONSE_CRC_FAIL	16#A2	Контрольная сумма ответа некорректна
RESPONSE_WRONG_SLAVE	16#A3	Получен ответ от другого slave-устройства (не от того, которому был отправлен запрос)
RESPONSE_WRONG_FUNCTION_ CODE	16#A4	Получен ответ с неверным кодом функции
REQUEST_FAILED_TO_SEND	16#A5	Ошибка COM-порта master-устройства. Запрос не был отправлен
RESPONSE_INVALID_DATA	16#A6	Ответ содержит данные, некорректные с точки зрения протокола Modbus (например, размер полученных данных не соответствует ожидаемому)
RESPONSE_INVALID_PROTOCOL	16#A7	Protocol ID в MBAP Header отличается от **0** (т е. используемый протокол не является Modbus TCP)
RESPONSE_INVALID_HEADER	16#A8	MBAP Header является некорректным (для протокола Modbus TCP)
UNDEFINED	16#FF	Код ошибки не может быть определен (например, еще не было отправлено ни одного запроса)

Таблица 4.6.3 – Описание элементов перечисления ModbusTCPComState

Название	Значение	Описание
OFF	16#0	Никаких действий не выполняется (например, slave-устройство исключено из опроса)
CONNECTING	16#1	Устанавливается соединение со slave-устройством
CONNECTED	16#2	Соединение установлено
DISCONNECTING	16#3	Соединение разорвано (со стороны контроллера)
SOCKET_ERROR	16#4	Ошибка операций чтения/записи через сокет (например, из-за разрыва линии связи)

Ниже приведен пример использования переменных диагностики. В случае возникновения ошибки обмена с одним из slave-устройств запускается таймер, который каждую секунду подает импульс в переменную диагностики **xResetComPort**, что приводит к переинициализации COM-порта, соответствующего данному компоненту **Modbus Master**. Это может быть полезным, когда на линию связи действуют помехи (например, от преобразователей частоты), что может привести к остановке работы COM-порта контроллера.

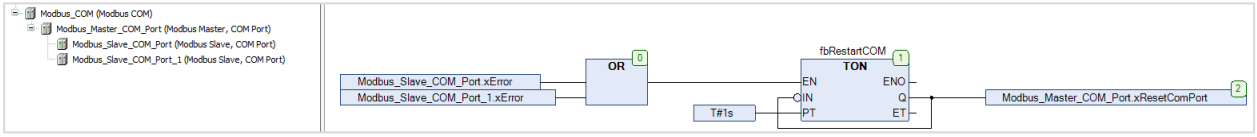


Рисунок 4.6.2 – Переинициализация COM-порта

Если контроллер работает в режиме **Modbus Serial Master** или **Modbus TCP Master**, то опросом slave-устройств можно управлять из кода программы. Для этого следует:

1. В компоненте **Device** на вкладке **Установки ПЛК** установить галочку **Включить диагностику для устройств**.

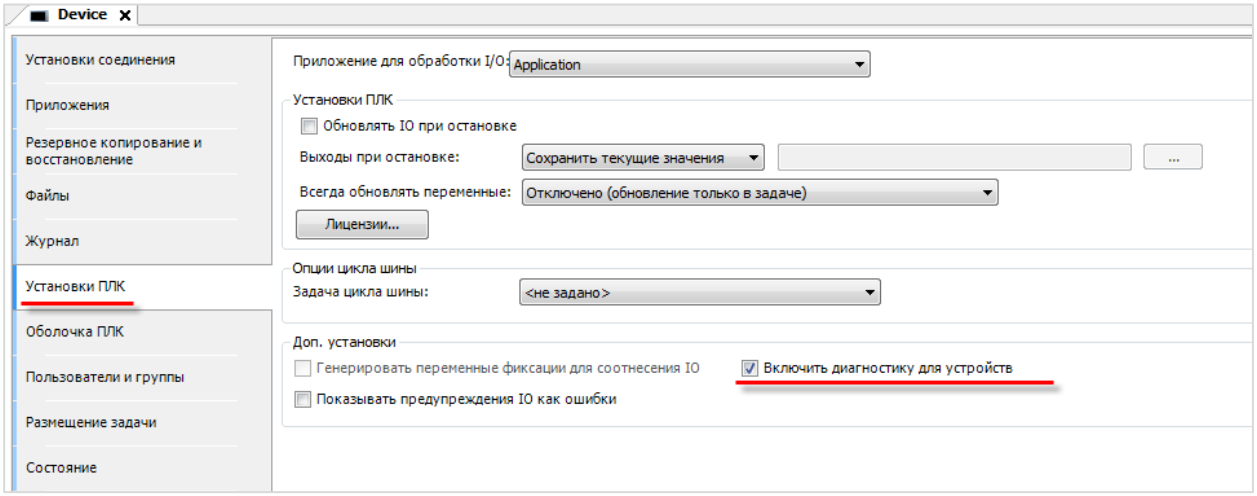


Рисунок 4.6.3 – Включение диагностики устройств

2. Для отключения устройства из опроса в коде программы присвоить переменной **<имя_устройства_из_дерева_проекта>.Enable** значение **FALSE**.

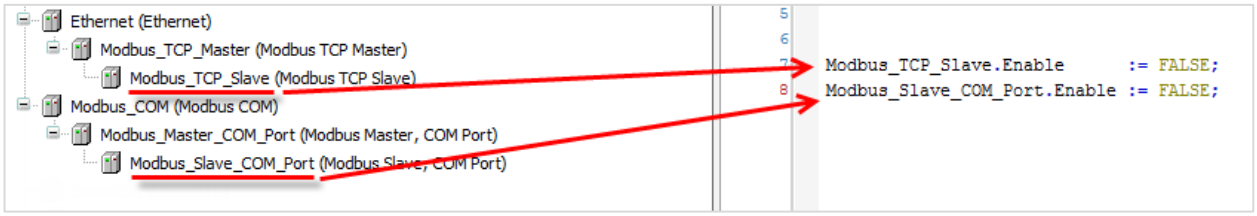


Рисунок 4.6.4 – Исключение slave-устройств из опроса

3. Для возобновления опроса присвоить переменной **<имя_устройства_из_дерева_проекта>.Enable** значение **TRUE**.

Если контроллер работает в режиме **Modbus Serial Master**, то можно изменить заданный адрес опрашиваемого slave-устройства из кода программы через свойство **SlaveAddress**.

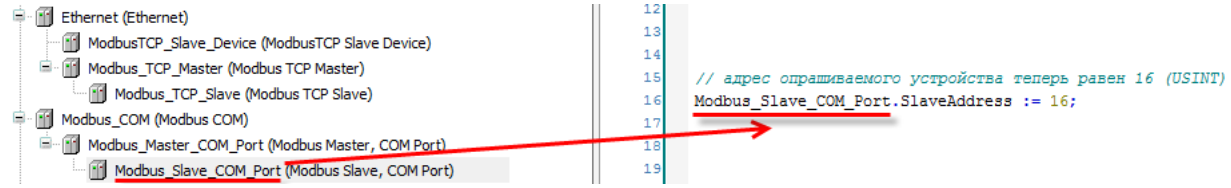


Рисунок 4.6.5 – Изменение заданного адреса опрашиваемого slave-устройства

Перед изменением потребуется остановить работу компонента (с помощью свойства **Enable**), а после изменения – запустить его заново. После перезагрузки контроллера адрес опрашиваемого slave-устройства будет инициализирован значением, заданным в настройках компонента (вкладка **Общее**), поэтому процедуру потребуется провести заново.

Если контроллер работает в режиме **Modbus Serial Master**, то можно изменить настройки COM-порта, используемого компонентом, с помощью метода **UpdateComPortSettings**:

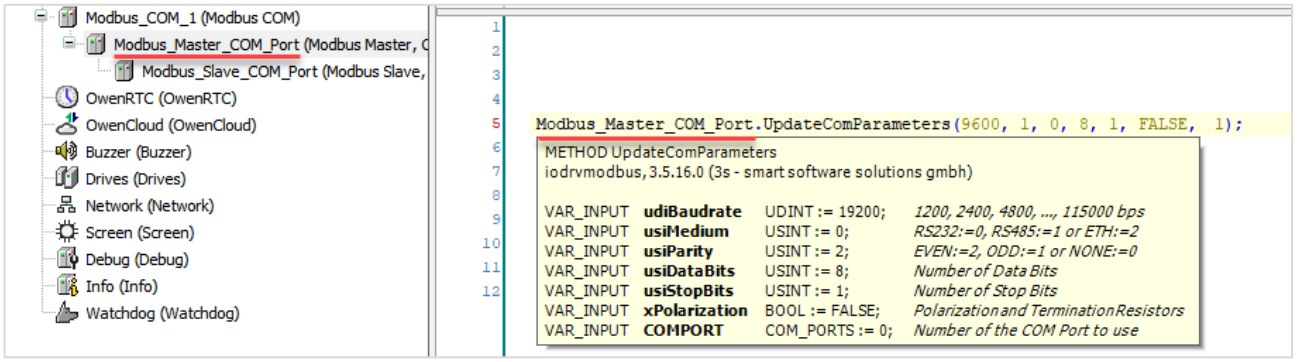


Рисунок 4.6.6 – Изменение настроек COM-порта

Перед изменением потребуется остановить работу компонента (с помощью входа **xStop**), а после изменения – запустить его заново. После перезагрузки контроллера параметры будут инициализированы значениями, заданными в настройках компонента (вкладка **Общее**), поэтому процедуру потребуется провести заново.

Если контроллер работает в режиме **Modbus TCP Master**, то можно изменить заданные сетевые настройки опрашиваемого slave-устройства с помощью метода **UpdateCommunicationSettings**:

PLC_PRG

```
PROGRAM PLC_PRG
VAR
    // новый IP-адрес для слэйва
    abyNewSlaveIp:          ARRAY [0..3] OF BYTE := [10, 2, 11, 177];
    // бит обновления настроек слэйва
    xUpdate:        BOOL;
END_VAR
```

```
// Modbus_TCP_Slave - имя нужного компонента из дерева проекта

Modbus_TCP_Slave.xConfirmError := FALSE;

IF xUpdate THEN

        // тормозим опрос слэйва
        Modbus_TCP_Slave.Enable := FALSE;

        // проверяем, что слэйв остановлен
        IF Modbus_TCP_Slave.ComState = 0 THEN

            // задаем новые настройки (адрес порта тоже можно изменить)
            Modbus_TCP_Slave.UpdateCommunicationSettings(ipAddress := abyNewSlaveIp, uiPort := 502);

            // включаем слэйв в работу
            Modbus_TCP_Slave.xConfirmError := TRUE;
            Modbus_TCP_Slave.Enable := TRUE;
            xUpdate := FALSE;
        END_IF
END_IF
```

Рисунок 4.6.7 – Изменение заданных сетевых настроек опрашиваемого slave-устройства

После перезагрузки контроллера настройки slave-устройства будут инициализированы значениями, заданными настройках компонента (вкладка **Общее**), поэтому процедуру смены настроек потребуется провести заново.

Если контроллер работает в режиме **Modbus TCP Slave**, то можно изменить номер сетевого порта, используемого компонентом, с помощью метода **UpdateCommunicationSetting**:

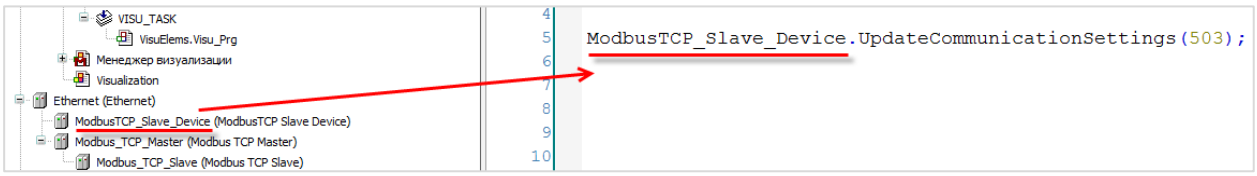


Рисунок 4.6.8 – Изменение номера порта в режиме Modbus TCP Slave

Перед изменением потребуется остановить работу компонента (с помощью входа **xEnable**), а после изменения – запустить его заново. После перезагрузки контроллера номер порта будет инициализирован значением, заданным в настройках компонента (вкладка **Общее**), поэтому процедуру потребуется провести заново.

Если контроллер работает в режиме **Modbus RTU Slave**, то можно изменить номер, скорость, режим контроля четности COM-порта и адрес устройства (Slave ID), используемого компонентом, с помощью методов **UpdateComPortSettings** и **UpdateUnitId**:

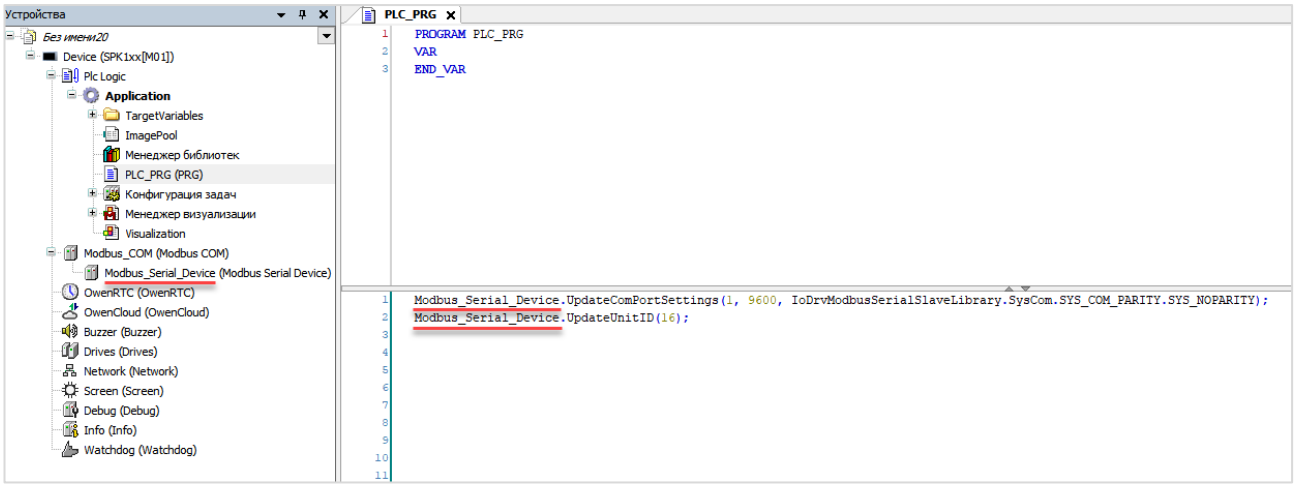


Рисунок 4.6.9 – Изменение настроек COM-порта в режиме Modbus RTU Slave

Перед изменением потребуется остановить работу компонента (с помощью входа **xEnable**), а после изменения – запустить его заново. После перезагрузки контроллера параметры будут инициализированы значениями, заданными в настройках компонента (вкладка **Общее**), поэтому процедуру потребуется провести заново.

Начиная с версии **3.5.16.0** при онлайн-подключении к контроллеру доступен ряд средств диагностики для обмена по Modbus.

В компоненте **Modbus TCP Master** на вкладке **Журнал** отображается лог подключений/потерь связи со slave-устройствами.

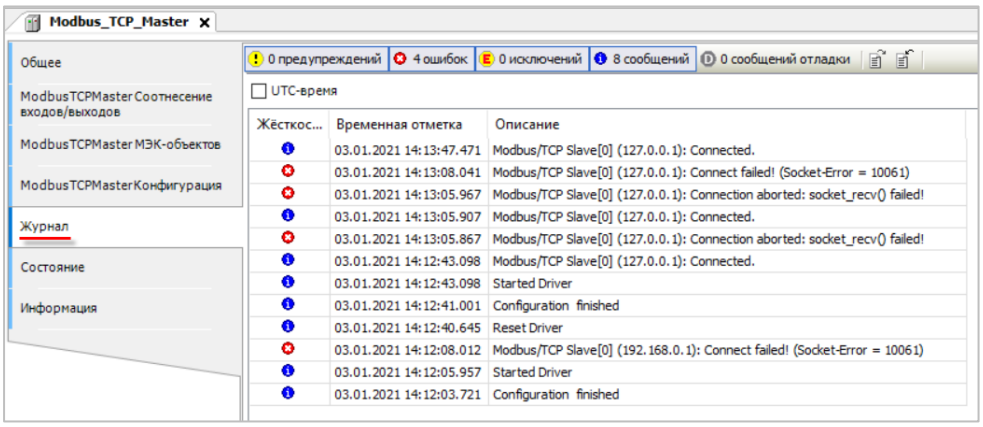


Рисунок 4.6.10 – Вкладка Журнал в компоненте Modbus TCP Master

В компонентах **Modbus TCP Slave** и **Modbus Slave COM Port** на вкладке **Состояние** отображается информация диагностики:

- состояние соединения (для **Modbus TCP Slave**);
- счетчик запросов, отправленных slave-устройству;
- счетчик ответов с кодом ошибки Modbus, полученных от slave-устройства;
- информация о последней ошибке – метка времени, индекс канала запроса (если ошибка не связана с каналом, то отображается **-1**) и код ошибки (например, ILLEGAL FUNCTION).

Кроме того, при возникновении и последующем исчезновении ошибки на иконке компонента продолжает отображаться бледный восклицательный знак. Для его исчезновения следует нажать кнопку **Подтвердить**.

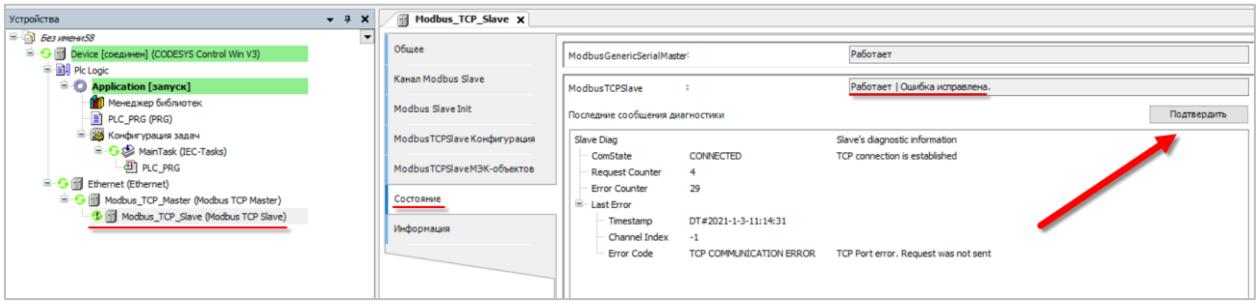


Рисунок 4.6.11 – Вкладка Состояние в компоненте Modbus TCP Slave

В компоненте **Modbus TCP Slave Device** на вкладке **Состояние** отображается информация диагностики:

- число подключенных клиентов;
- статус TCP-порта;
- счетчик запросов, полученных компонентом;
- счетчик ответов с кодом ошибки Modbus, которые компонент отправил master-устройству;
- статус, счетчик запросов и счетчик ошибок для режима шлюза (**Serial Gateway**).

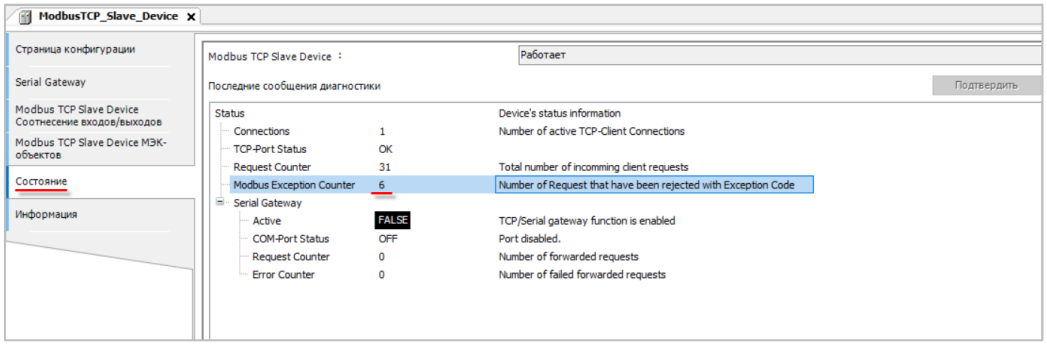


Рисунок 4.6.12 – Вкладка Состояние в компоненте Modbus TCP Slave Device

4.7 Компоненты Modbus и конфигурация задач

По умолчанию компоненты Modbus выполняются в задаче с наименьшим временем цикла. Для корректной работы компонентов в проекте должна присутствовать хотя бы одна задача с временем цикла **10…20 мс**. Более подробная информация по этому поводу приведена в справке CODESYS. Информация об обработке задач в CODESYS V3 приведена в этой статье.

При отсутствии ответа от slave-устройств время цикла задачи, в которой происходит выполнение компонентов Modbus, может значительно возрастать из-за ожидания таймаута ответа.

4.8 Преобразование данных для передачи по Modbus

Стандарт **Modbus** описывает только два типа данных – **BOOL** и **WORD**. Достаточно часто возникает потребность передать данные других типов (например, **REAL** и **STRING**). В данном случае на устройстве, которое отправляет данные, следует преобразовать их в последовательность переменных типа **WORD**. Соответственно, на устройстве, получающем данные, должно быть выполнено обратное преобразование.

В **CODESYS V3.5** есть два базовых способа для подобных преобразований: объединения и указатели. Кроме того, можно воспользоваться функциями конвертации из библиотеки **OwenCommunication**.

4.8.1 Использование объединений (UNION)

Объединение (UNION) представляет собой пользовательский тип данных, все переменные которого расположены в одной области памяти. Таким образом, переменные различных типов будут представлять различную интерпретацию одних и тех же данных. Для конвертации достаточно записать значение в одну из переменных объединения и считать его из другой.

Для конвертации значения с плавающей точкой из двух переменных типа **WORD** в переменную типа **REAL** следует:

1. Нажать **ПКМ** на приложение **Application** и добавить объект **DUT** типа **объединение** с названием **Real_Word**:

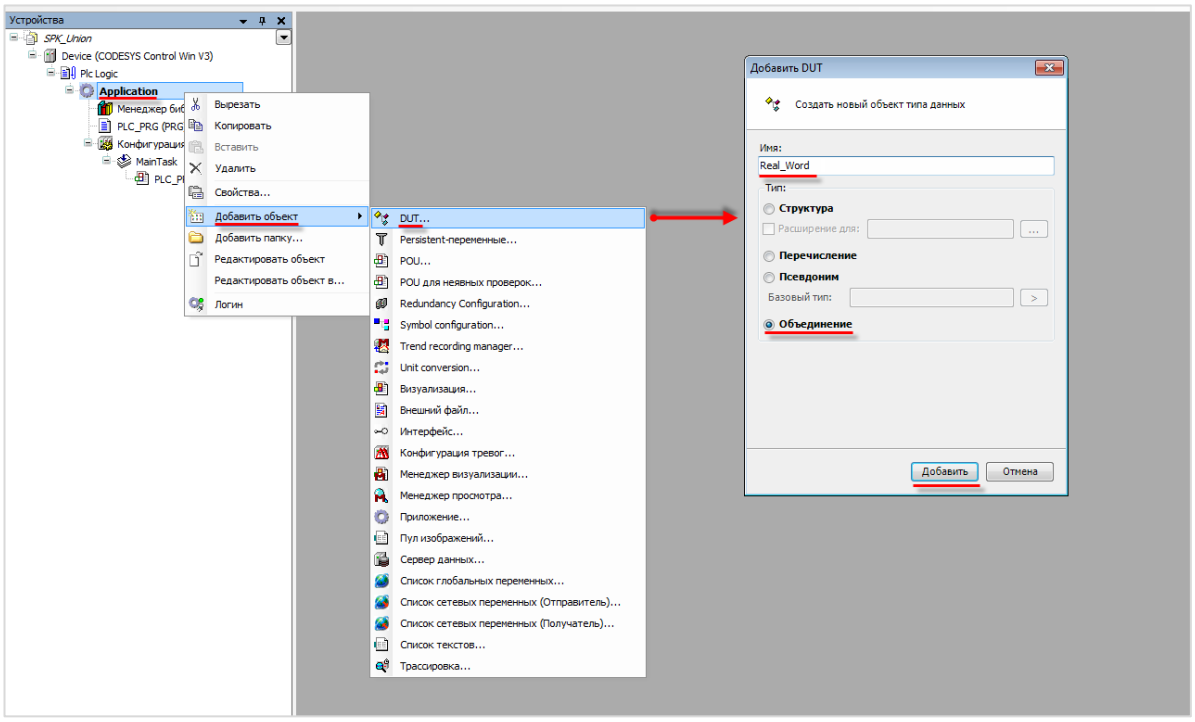


Рисунок 4.8.1 – Добавление объединения в проект CODESYS

2. В объединении объявить переменную **rRealValue** типа **REAL** и массив **awModbusReal** типа **WORD**, содержащий два элемента:

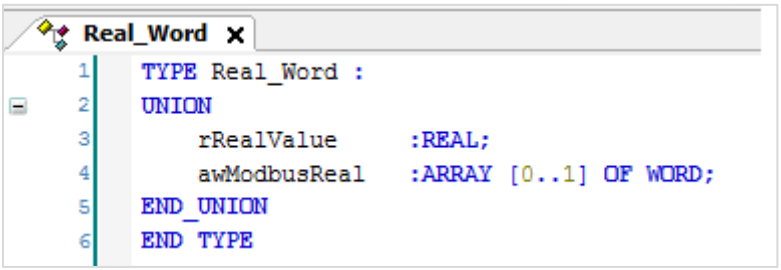


Рисунок 4.8.2 – Объявление переменных объединения

3. В программе объявить экземпляр объединения **Real_Word** с названием **_2WORD_TO_REAL**:

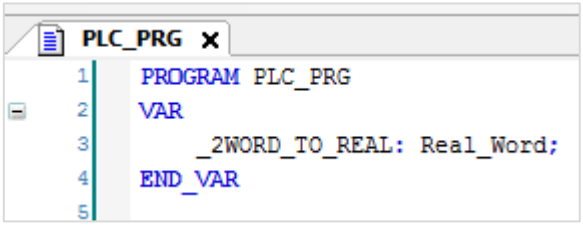


Рисунок 4.8.3 – Объявление экземпляра объединения в программе

Для использования переменных объединения в нужном месте программы следует ввести имя экземпляра объединения и нажать точку, после чего выбрать из списка нужную переменную:

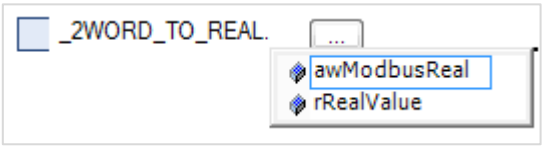


Рисунок 4.8.4 – Работа с переменными объединения в программе

4. Переменные массива **awModbusReal** будут привязаны к регистрам во время настройки **Modbus**, а переменная **rRealValue** будет использоваться в программе.

Ниже приведен пример: контроллер является master-устройством и считывает значение типа **REAL** из 10 и 11 holding регистров slave-устройства в переменную **rRealValue** объединения **_2WORD_TO_REAL**:

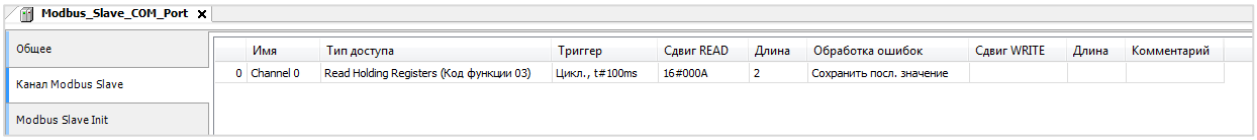


Рисунок 4.8.5 – Настройка канала для считывания значения с плавающей точкой

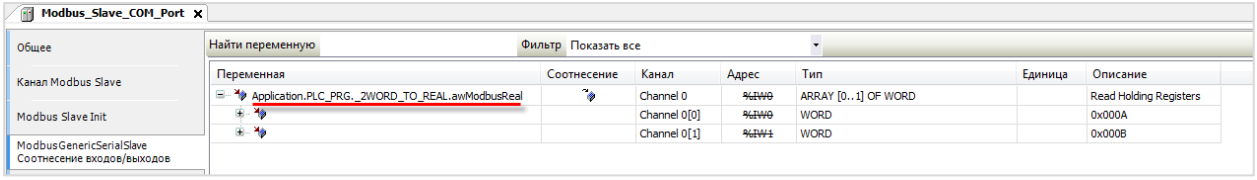


Рисунок 4.8.6 – Привязка переменной объединения к каналу

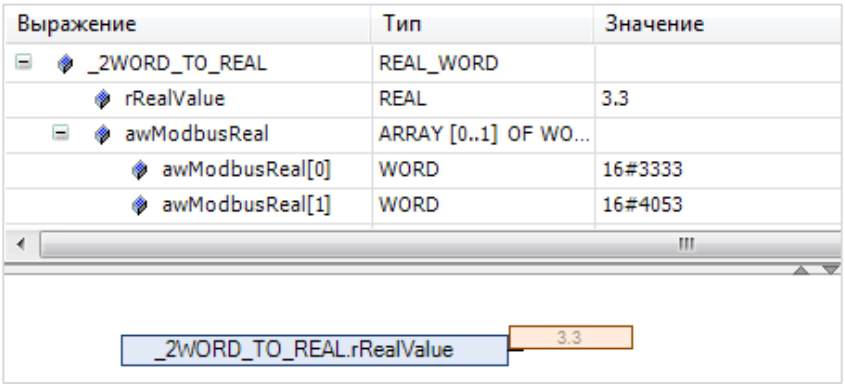


Рисунок 4.8.7 – Использование REAL переменной объединения в программе

Обратное преобразование (**REAL** в два **WORD**) выполняется аналогичным способом: пользователь записывает значение в **REAL** переменную объединения, после чего работает с массивом из двух **WORD**.

Работа с **DWORD, STRING** и другими типами данных происходит аналогично – в приведенном выше примере достаточно изменить тип переменной объединения (вместо **rRealValue** использовать **dwDwordValue** типа **DWORD**, **sStringValue** типа **STRING** и так далее).

5. Передача **REAL** по протоколу **Modbus** не стандартизирована – значение с плавающей точкой передаются в виде двух регистров (переменных типа **WORD**), но порядок этих **WORD** переменных (или даже их байт) может отличаться. В данном случае следует привести их к нужному для конкретного устройства виду.

Порядок **WORD** можно менять на этапе привязки переменных к регистрам – например, можно сравнить рисунки 4.8.6 и 4.8.8:

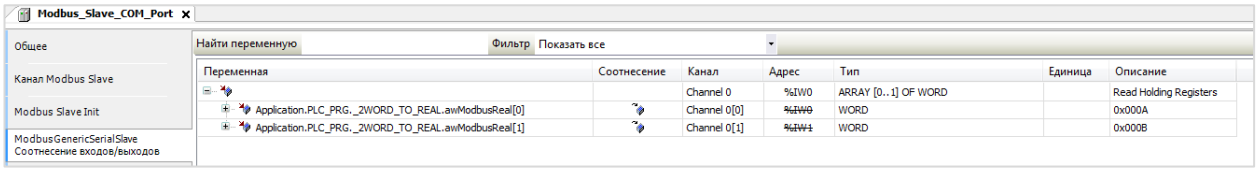


Рисунок 4.8.8 – Привязка элементов массива объединения к регистрам канала

В случае необходимости изменения порядка байтов можно создать два объединения – в первом будет происходить конвертация полученных по **Modbus** значений **WORD** в массив байтов, а во втором – конвертация нового массива байтов (переставленных в нужном порядке) в переменную типа **REAL**. Ниже приведен пример конвертации 2 **WORD** в **REAL** с перестановкой байт (0–1–2–3 в 3–2–1–0):

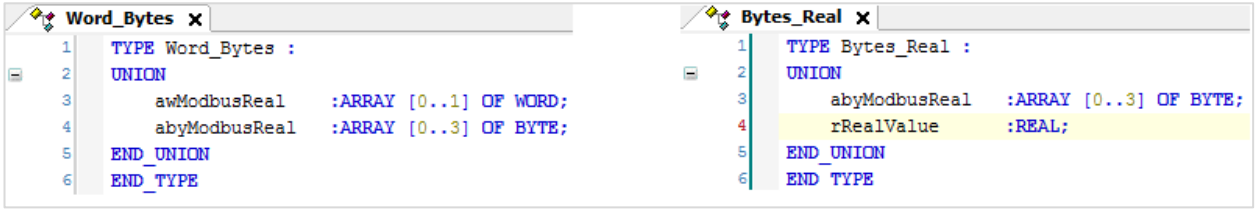


Рисунок 4.8.9 – Объявление двух объединений

Выражение	Тип	Значение	Подготовленное ...	Адрес	Комментарий
_2WORD_TO_4BYTES	Word_Bytes				
awModbusReal	ARRAY [0..1] OF WO...				
awModbusReal[0]	WORD	16#5340			
awModbusReal[1]	WORD	16#3333			
abyModbusReal	ARRAY [0..3] OF BYTE				
abyModbusReal[0]	BYTE	16#40			
abyModbusReal[1]	BYTE	16#53			
abyModbusReal[2]	BYTE	16#33			
abyModbusReal[3]	BYTE	16#33			
_4BYTES_TO_REAL	Bytes_Real				
abyModbusReal	ARRAY [0..3] OF BYTE				
abyModbusReal[0]	BYTE	16#33			
abyModbusReal[1]	BYTE	16#33			
abyModbusReal[2]	BYTE	16#53			
abyModbusReal[3]	BYTE	16#40			
rRealValue	REAL	3.3			

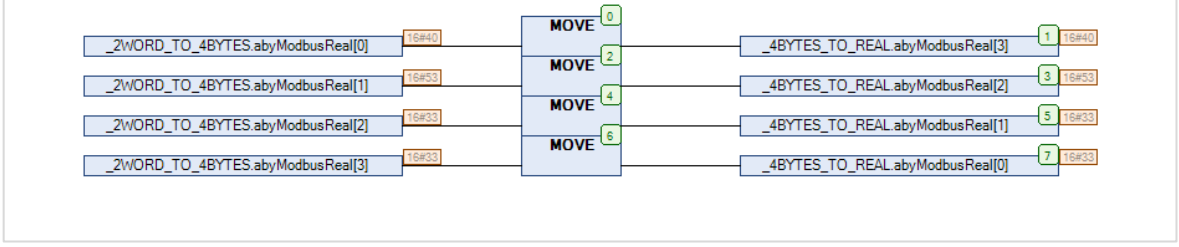


Рисунок 4.8.10 – Пример работы с объединениями на языке CFC. Перестановка байтов

На основе приведенных примеров пользователь может создать свои функции и функциональные блоки для удобной конвертации данных.

4.8.2 Использование указателей

Указатели содержат адреса переменных. Обращаясь к переменной по указателю, пользователь работает непосредственно с областью памяти, в которой хранится эта переменная, что позволяет производить любую обработку находящихся в ней данных.

ПРИМЕЧАНИЕ
Использование указателей подразумевает соответствующую квалификацию программиста. Некорректное использование указателей может привести к «зависанию» программы и контроллера.

Для конвертации значения с плавающей точкой из двух переменных типа **WORD** в переменную типа **REAL** следует:

1. Объявить в программе нужные переменные и указатель на переменную того типа, в который производится конвертация:

```
2  VAR
3      rRealValue        :REAL;
4      awModbusReal      :ARRAY [0..1] OF WORD;
5      prModbusReal      :POINTER TO REAL;
6  END_VAR
```

Рисунок 4.8.11 – Объявление указателя

2. Переменные массива **awModbusReal** будут привязаны к регистрам при настройке **Modbus**.

Ниже приведен пример: контроллер является master-устройством и считывает значение типа **REAL** из 10 и 11 holding регистров slave-устройства в переменную **rRealValue**:

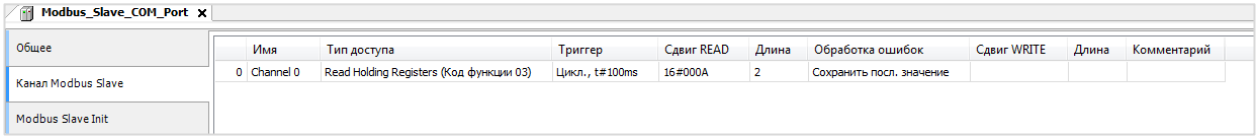


Рисунок 4.8.12 – Настройка канала для считывания значения с плавающей точкой

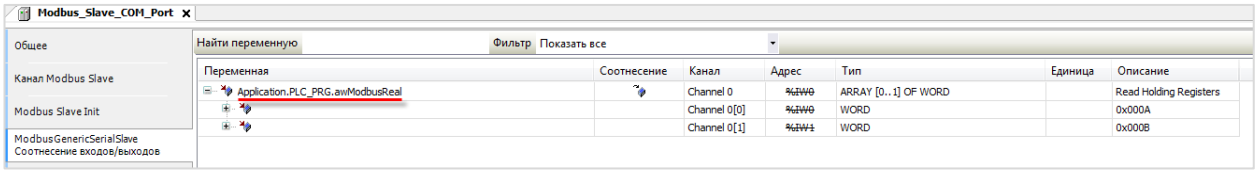


Рисунок 4.8.13 – Привязка переменной к каналу

3. В программе с помощью оператора **ADR** записать в указатель адрес массива **awModbusReal**, после чего присвоить переменной **rRealValue** значение, хранящееся по указателю:

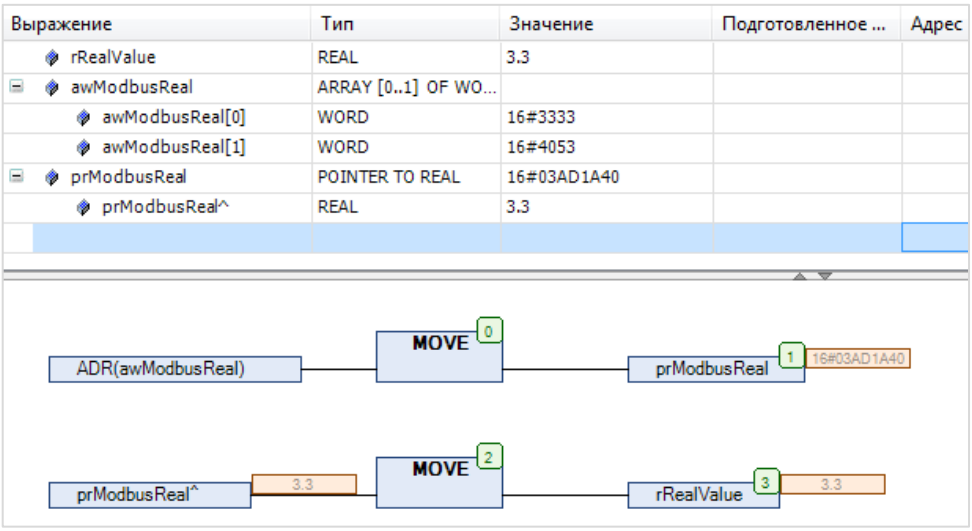


Рисунок 4.8.14 – Пример работы с указателями на языке CFC (конвертация 2 WORD в REAL)

Обратное преобразование (**REAL** в два **WORD**) выполняется аналогичным способом:

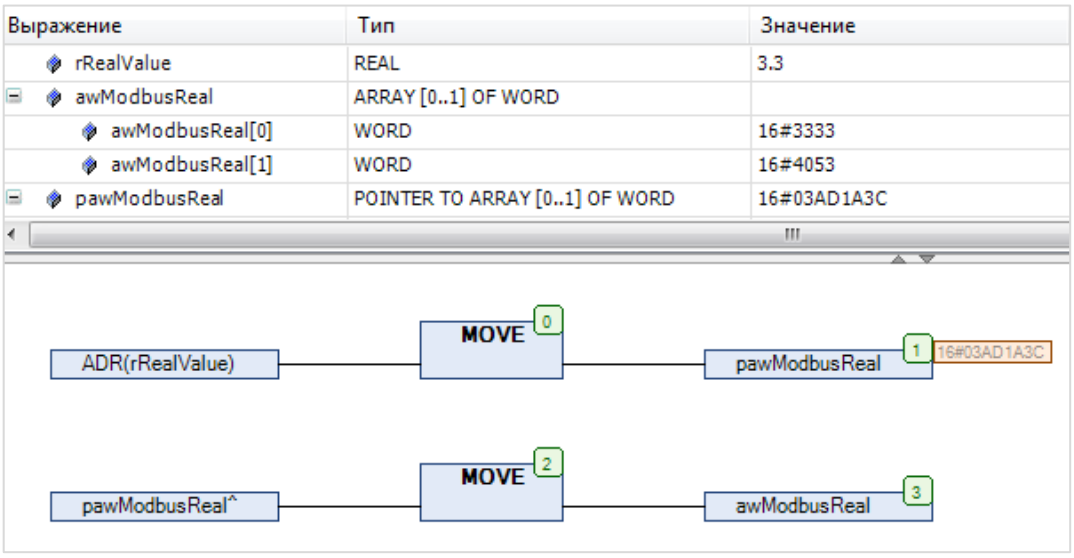


Рисунок 4.8.15 – Пример работы с указателями на языке CFC (конвертация REAL в 2 WORD)

Работа с **DWORD**, **STRING** и другими типами данных происходит аналогично – в приведенном выше примере достаточно изменить тип переменной объединения и ее указателя (например, использовать **dwDwordValue** типа **DWORD** и **pdwModbusDword** типа **POINTER TO DWORD**).

4. Передача **REAL** по протоколу **Modbus** не стандартизирована – значение с плавающей точкой передаются в виде двух регистров (переменных типа **WORD**), но порядок этих **WORD** переменных (или даже их байт) может отличаться. В данном случае следует привести их к нужному для конкретного устройства виду.

Порядок **WORD** можно менять на этапе привязки переменных к регистрам – например, можно сравнить рисунки 4.8.14 и 4.8.16:

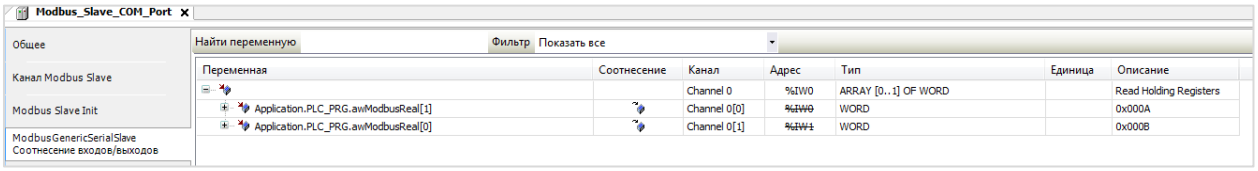


Рисунок 4.8.16 – Привязка элементов массива к регистрам канала

В случае необходимости изменения порядка байтов следует вместо массива из двух **WORD** использовать массив из четырех байт и указатель на него. Ниже приведен пример конвертации 2 **WORD** в **REAL** с перестановкой байт (0–1–2–3 в 3–2–1–0):

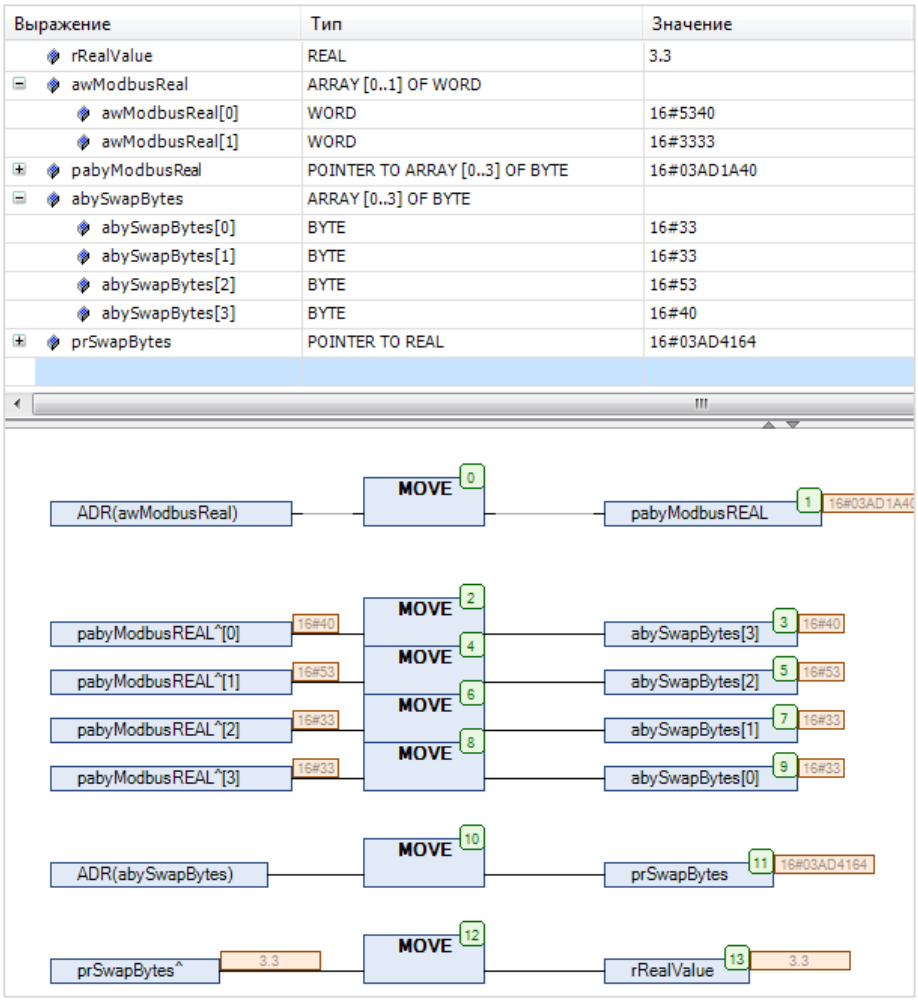


Рисунок 4.8.17 – Пример работы с указателями на языке CFC. Перестановка байт

4.9 Пример: СПК1xx [M01] (Modbus RTU Master) + модули Mx110

В качестве примера будет рассмотрена настройка обмена с модулями Mx110 (МВ110-8А, МВ110-16Д, МУ110-16Р) с использованием **стандартных средств конфигурации**.

Реализуемый алгоритм: если значение первого аналогового входа модуля **МВ110-8А** превышает **30** и при этом первый дискретный вход модуля **МВ110-16Д** имеет значение **TRUE** (замкнут), то первому дискретному выходу модуля **МУ110-16Р** присваивается значение **TRUE** (замкнут). Во всех остальных случаях дискретному выходу присваивается значение **FALSE** (разомкнут).

Структурная схема примера приведена на рисунке ниже:

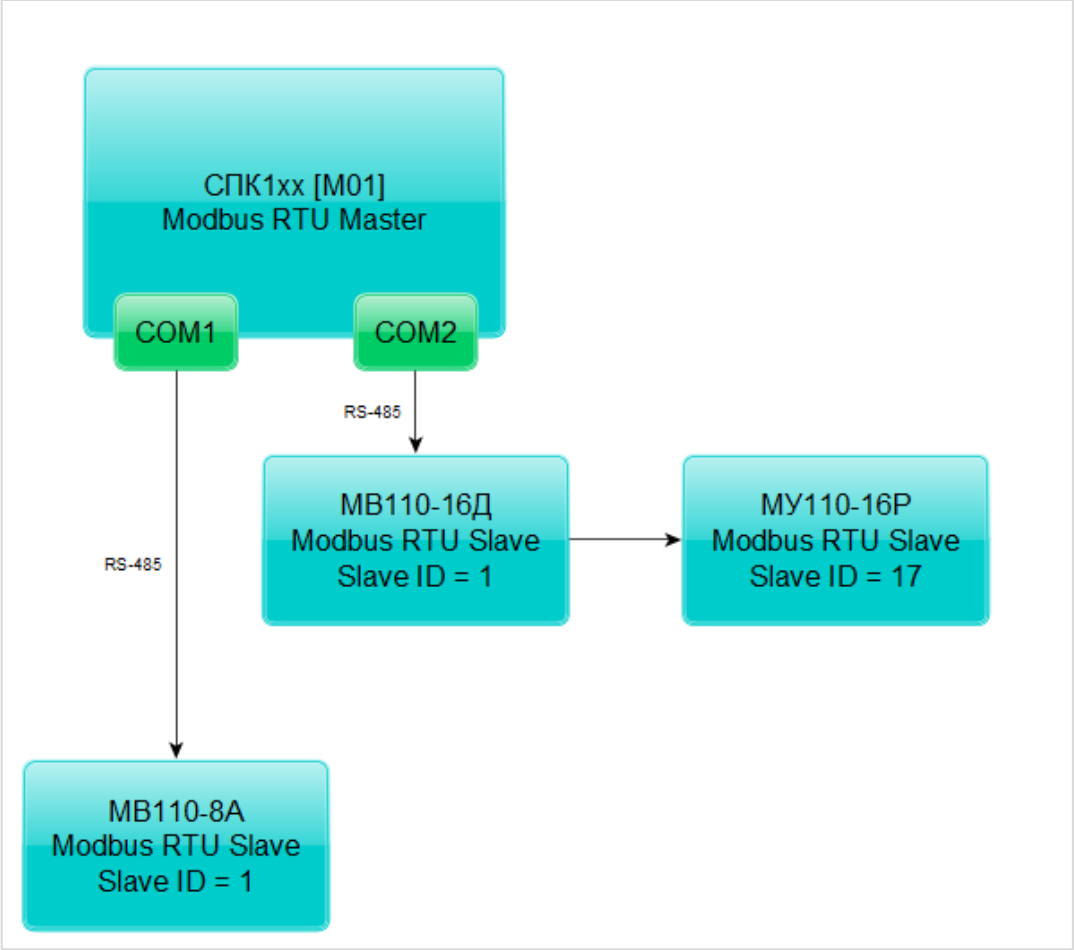


Рисунок 4.9.1 – Структурная схема примера

Пример создан в среде **CODESYS V3.5 SP17 Patch 3** и подразумевает запуск на **СПК1xx [M01]** с таргет-файлом **3.5.17.31.** 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_CodesysModbusRtuMasterMx110_3517v1.projectarchive

Сетевые параметры модулей приведены в таблице ниже:

Таблица 4.9.1 – Сетевые параметры модулей Mx110

Параметр	МВ110-8А	МВ110-16Д	МУ110-16Р
COM-порт контроллера, к которому подключен модуль	COM1	COM2	
ID COM-порта	1	2	
Адрес модуля	1	1	17
Скорость обмена	115200		
Количество бит данных	8		
Контроль четности	Отсутствует		
Количество стоп-бит	1		

Переменные примера описаны в таблице ниже:

Таблица 4.9.2 – Список переменных примера

Модуль	Имя переменной	Тип	Описание
МВ110-8А	awModbusReal	ARRAY [0..1] OF WORD	Значение температуры в виде двух **WORD**, считываемое с модуля
	rRealValue	REAL	Значение температуры в виде числа с плавающей точкой для использования в программе
МВ110-16Д	wDI	WORD	Значение дискретных входов в виде битовой маски. При обращении к отдельным входам указывается их номер, начиная с 0: **wDI.0** – состояние первого входа (TRUE/FALSE) **wDI.1** – состояние второго входа …
МУ110-16Р	wDO	WORD	Значение дискретных выходов в виде битовой маски. При обращении к отдельным выходам указывается их номер, начиная с 0: **wDO.0** – состояние первого выхода (TRUE/FALSE) **wDO.1** – состояние второго выхода …
-	wPrevDO	WORD	Значение дискретных выходов в виде битовой маски из предыдущего цикла программы. Используется для отправки команды записи только в случае изменения значений выходов (иначе будет производиться циклическая запись последнего значения)
-	xTrigger	BOOL	Триггерная переменная, управляющая функцией записи дискретного выхода (запись происходит по переднему фронту переменной)

Для настройки обмена следует:

1. Настроить модули **Mx110** с помощью программы **Конфигуратор Mx110** в соответствии с таблицей 4.9.1. Подключить модули к COM-портам контроллера в соответствии с рисунком 4.9.1.

2. Создать новый проект **CODESYS** с программой **PLC_PRG** на языке **CFC**:

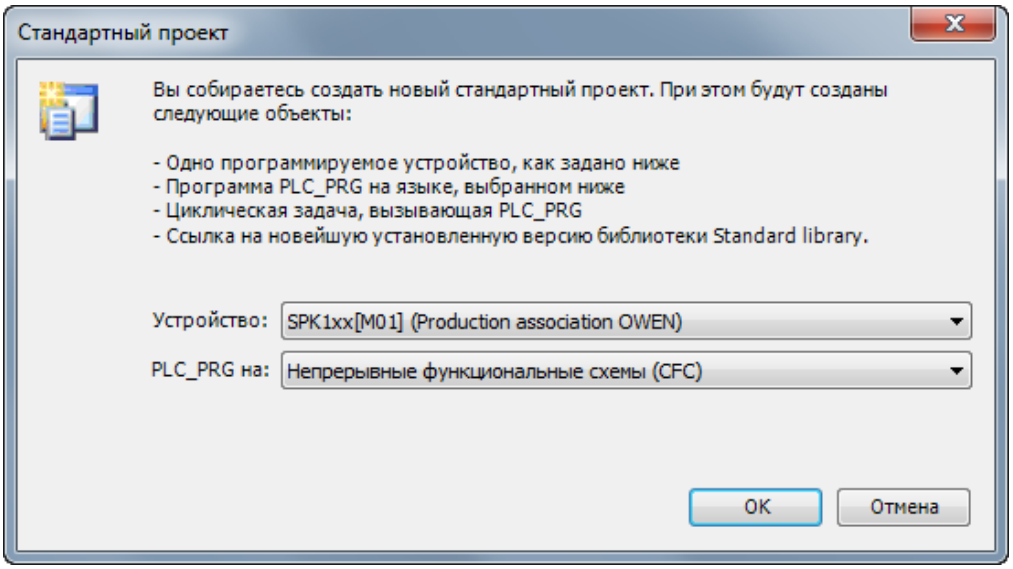


Рисунок 4.9.2 – Создание проекта CODESYS

3. Добавить в проект объединение с именем **Real_Word**:

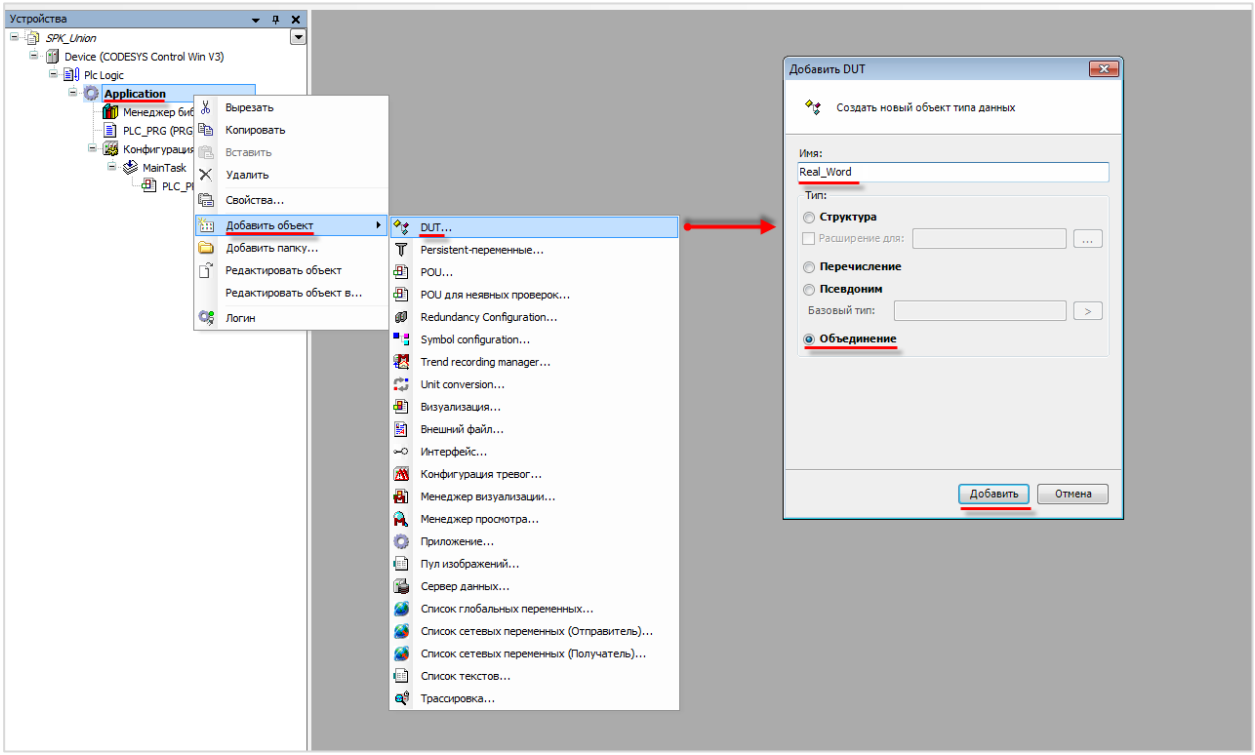


Рисунок 4.9.3 – Добавление в проект объединения

4. В объединении объявить переменную **rRealValue** типа **REAL** и массив **awModbusReal** типа **WORD**, содержащий два элемента:

```
Real_Word
1  TYPE Real_Word :
2  UNION
3      rRealValue      :REAL;
4      awModbusReal    :ARRAY [0..1] OF WORD;
5  END_UNION
6  END_TYPE
```

Рисунок 4.9.4 – Объявление переменных объединения

5. В программе **PLC_PRG** объявить экземпляр объединения **Real_Word** с названием **_2WORD_TO_REAL,** переменные **wDI, wDO** и **wPrevDO** типа **WORD** и переменную **xTrigger** типа **BOOL**. Описание переменных приведено в таблице 4.9.2.

```
PLC_PRG
1  PROGRAM PLC_PRG
2  VAR
3      _2WORD_TO_REAL:     Real_Word;  // значение 1-го входа МВ110-8А
4      wDI:                WORD;       // битовая маска входов МВ110-16Д
5      wDO:                WORD;       // битовая маска выходов МУ110-16Р
6      wPrevDO:            WORD;       // битовая маска предыдущей записи выходов МУ110-16Р
7      xTrigger:           BOOL;       // триггер записи выходов
8  END_VAR
```

Рисунок 4.9.5 – Объявление переменных программы

Код программы будет выглядеть следующим образом:

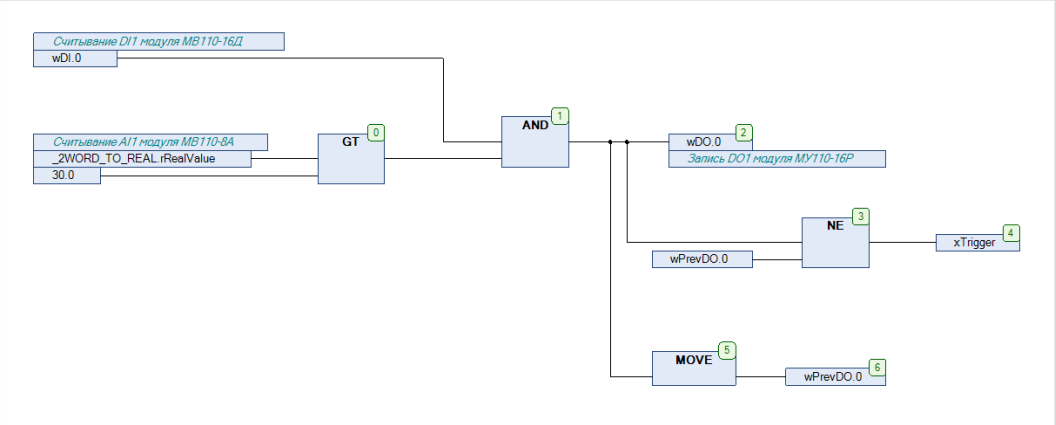


Рисунок 4.9.6 – Код программы PLC_PRG

Программа работает следующим образом: если значение переменной **rRealValue** (связанной с первым аналоговым входом модуля **МВ110-8А)** превышает 30 и при этом значение нулевого бита переменной **wDI** (связанной с первым дискретным входом модуля **МВ110-16Д**) имеет значение **TRUE**, то нулевому биту переменной **wDO** присваивается значение **TRUE**. Если на предыдущем цикле значение нулевого бита **wDO** отличалось от текущего, то переменная **xTrigger** принимает значение **TRUE**, что приводит к однократной записи текущего значения бита в первый дискретный выход модуля **МУ110-16Р**.

6. Добавить в проект два компонента **Modbus COM** с названиями **COM1** и **COM2**.

ПРИМЕЧАНИЕ

Версия компонента должна соответствовать версии таргет-файла. Для отображения всех доступных версий компонента следует установить галочку **Отображать все версии**. См. рекомендации в приложении А.

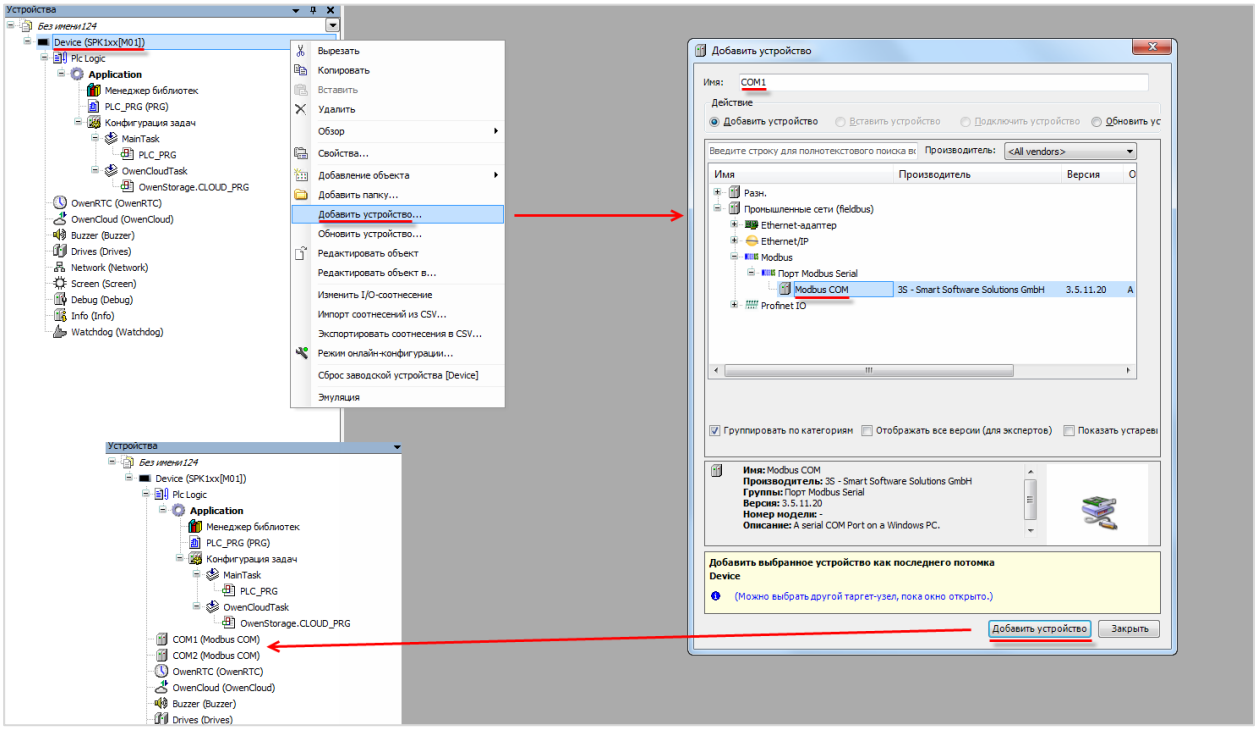


Рисунок 4.9.7 – Добавление компонента Modbus COM

В конфигурации COM-портов следует указать номера COM-портов и сетевые настройки в соответствии с таблицей 4.9.1:

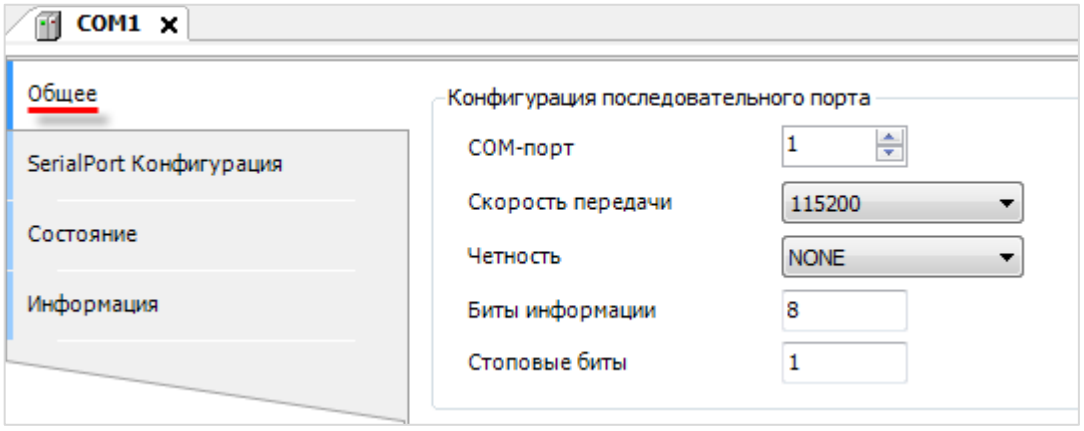


Рисунок 4.9.8 – Настройки COM-порта COM1

7. В каждый из COM-портов добавить компонент **Modbus Master**.

ПРИМЕЧАНИЕ

Версия компонента должна соответствовать версии таргет-файла. Для отображения всех доступных версий компонента следует установить галочку **Отображать все версии**. См. рекомендации в приложении А.

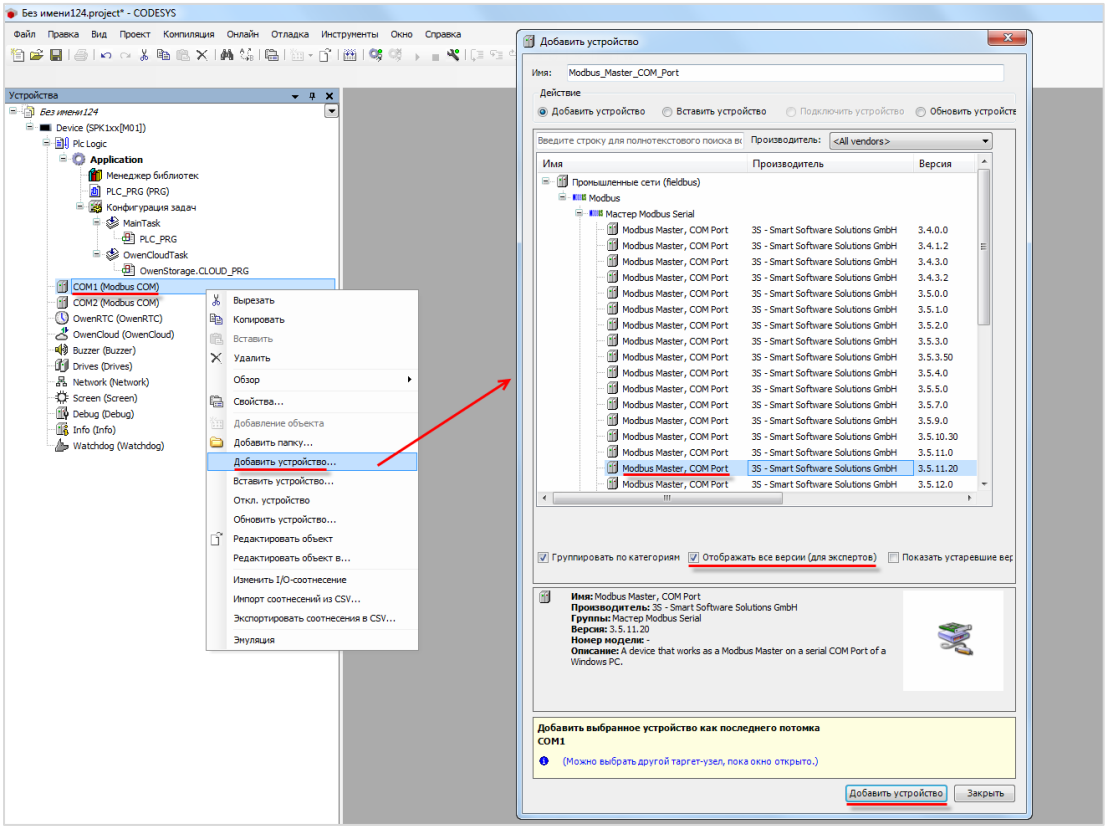


Рисунок 4.9.9 – Добавление компонента Modbus Master

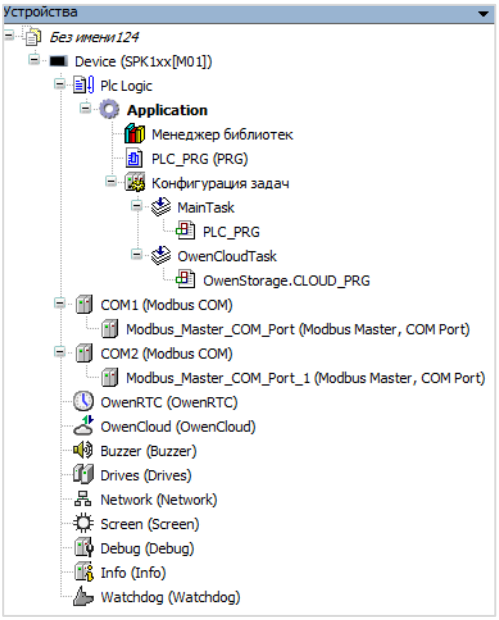


Рисунок 4.9.10 – Внешний вид дерева проекта после добавления Modbus Master

В настройках компонентов на вкладке **Общее** следует установить галочку **Автоперезапуск соединения**. В параметре **Время между фреймами** установить значение **20 мс**.

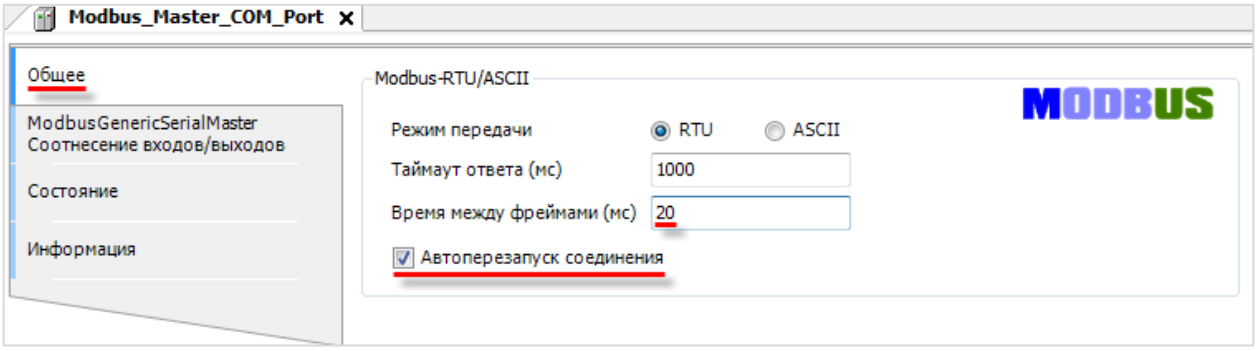


Рисунок 4.9.11 – Настройки компонентов Modbus Master

8. В компонент **Modbus Master** порта **COM1** следует компонент **Modbus Slave** с именем **MV110_8A,** а в компонент **Modbus Master** порта **COM2** – компоненты **Modbus Slave** с именами **MV110_16D** и **MU110_16R**.

ПРИМЕЧАНИЕ

Версия компонента должна соответствовать версии таргет-файла. Для отображения всех доступных версий компонента следует установить галочку **Отображать все версии**. См. рекомендации в приложении А.

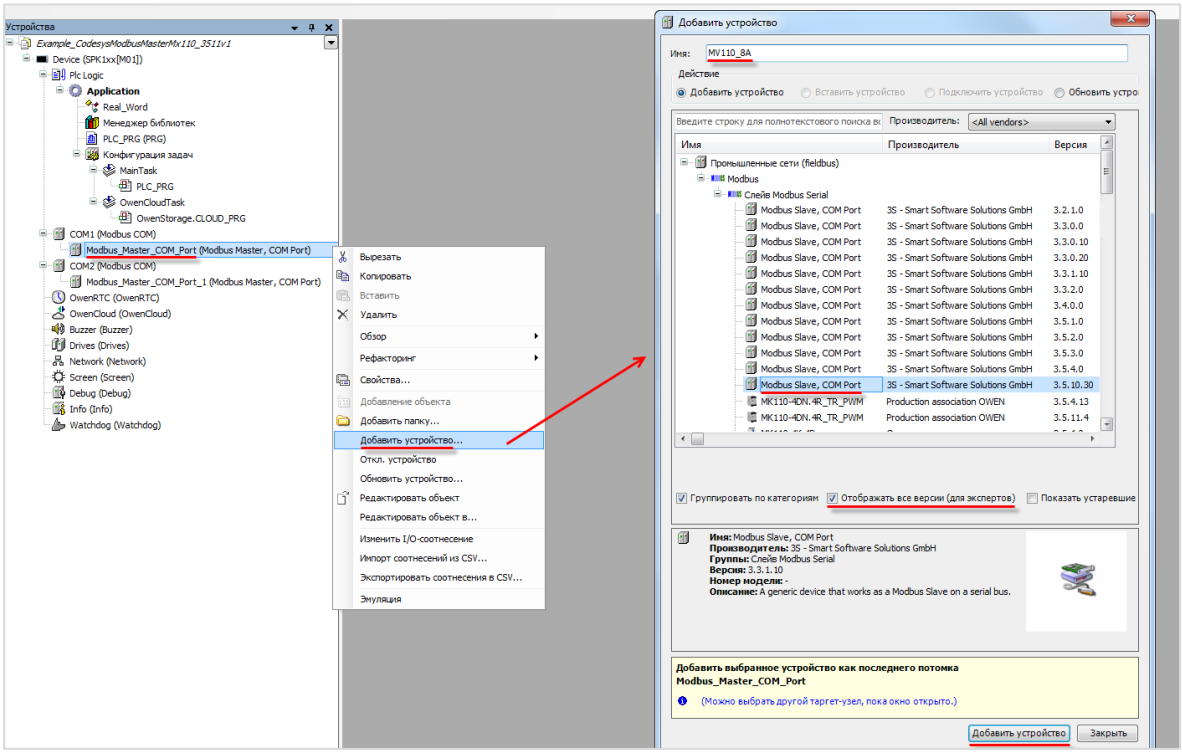


Рисунок 4.9.12 – Добавление slave-устройств в проект CODESYS

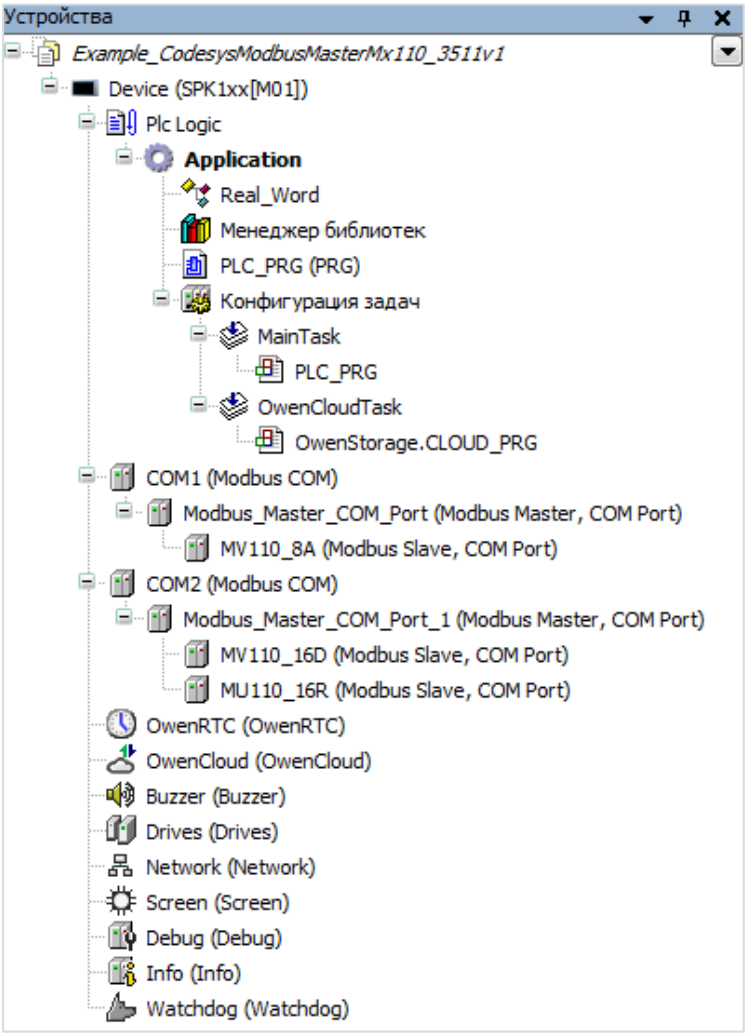


Рисунок 4.9.13 – Внешний вид дерева проекта после добавления slave-устройств

В настройках slave-устройств следует указать их адреса согласно таблице 4.9.1 (МВ110-8А – адрес **1**, МВ110-16Д – адрес **1**, МУ110-16Р – адрес **17**):

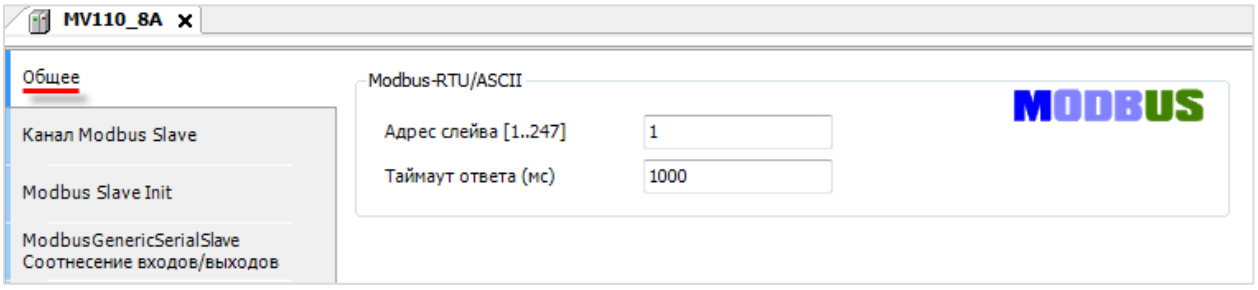


Рисунок 4.9.14 – Настройка slave-устройства MV110_8A

9. В настройках компонента **MV110_8A** на вкладке **Канал Modbus Slave** следует добавить канал, в котором с помощью функции **Read Holding Registers** будет считываться значение **четвертого** и **пятого** регистров модуля. В данных регистрах содержится значение входа 1 в представлении с плавающей точкой. Таблица регистров модуля и поддерживаемые функции приведены в руководстве по эксплуатации.

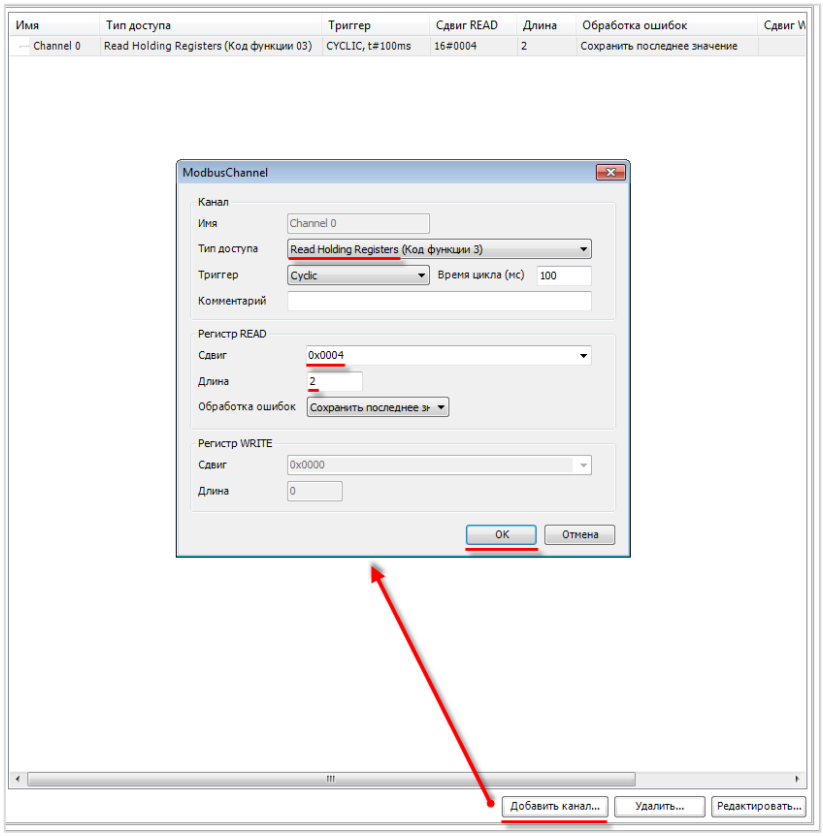


Рисунок 4.9.15 – Добавление канала в конфигурацию slave-устройства MV110_8A

На вкладке **ModbusGenericSerialSlave Соотнесение входов/выходов** следует привязать к каналу элементы объединения **_2WORD_TO_REAL**. Первый считываемый регистр присваивается первому элементу массива, а второй – нулевому. Это связано с тем, что порядок **WORD** в **REAL** у модуля и контроллера отличается.

Для параметра **Всегда обновлять переменные** следует установить значение **Включено 2**.

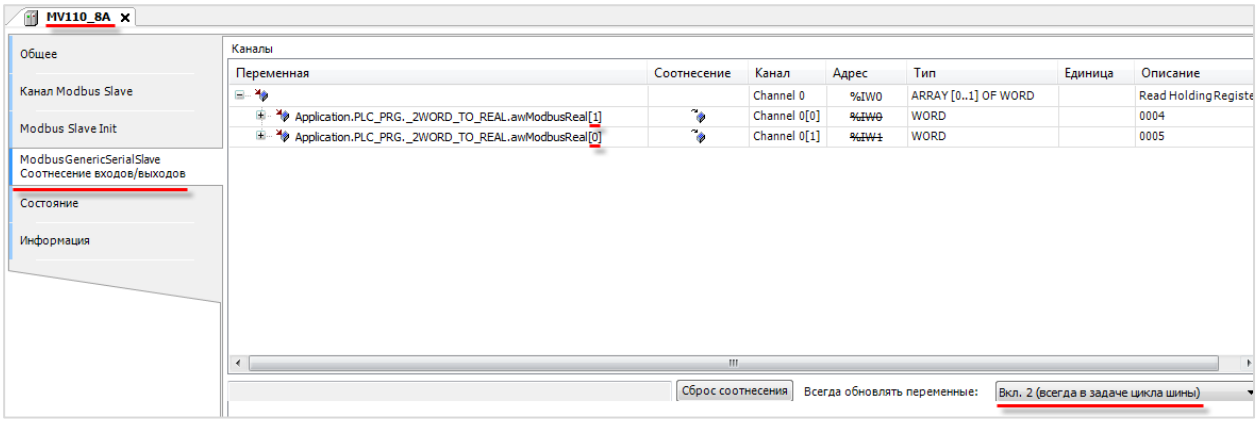


Рисунок 4.9.16 – Привязка переменных к каналу

10. В настройках компонента **MV110_16D** на вкладке **Канал Modbus Slave** следует добавить канал, в котором с помощью функции **Read Holding Registers** будет считываться значение регистра **0x0033**. В данном регистре содержится битовая маска состояний дискретных входов.

Таблица регистров модуля и поддерживаемые функции приведены в руководстве по эксплуатации.

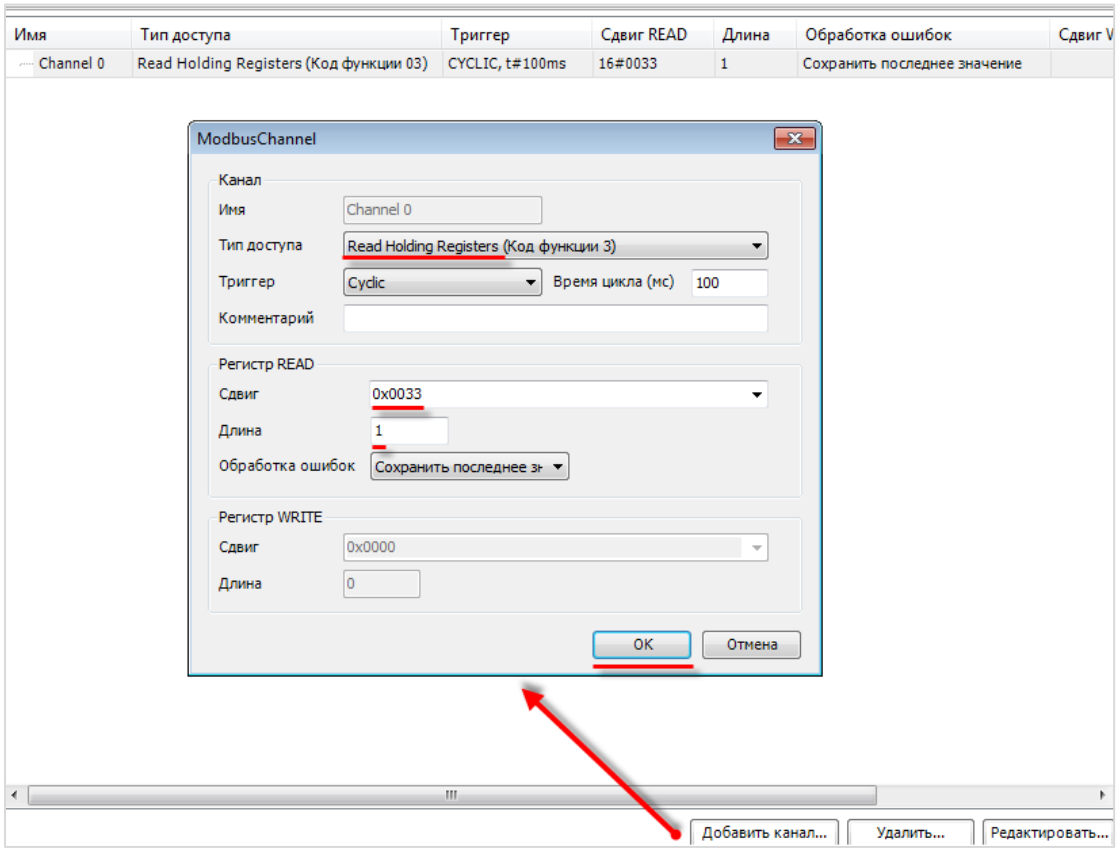


Рисунок 4.9.17 – Добавление канала в конфигурацию slave-устройства MV110_16D

На вкладке **ModbusGenericSerialSlave Соотнесение входов/выходов** следует к каналу переменную **wDI**. Для параметра **Всегда обновлять переменные** следует установить значение **Включено 2**.

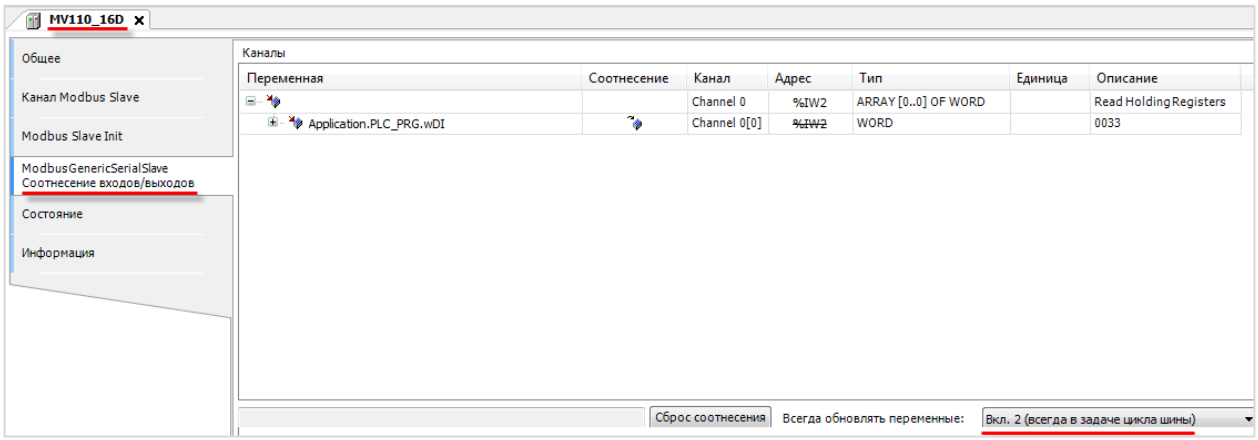


Рисунок 4.9.18 – Привязка переменных к каналу

11. В настройках компонента **MU110_16R** на вкладке **Канал Modbus Slave** следует добавить канал, в котором с помощью функции **Write Multiple Registers** будет записываться значение в регистр **0x0032**. В данном регистре содержатся значения выходов модуля в виде битовой маски. У параметра **Триггер** следует установить значение **Передний фронт**, чтобы иметь возможность управлять записью в модуль с помощью логической переменной.

Таблица регистров модуля и поддерживаемые функции приведены в руководстве по эксплуатации.

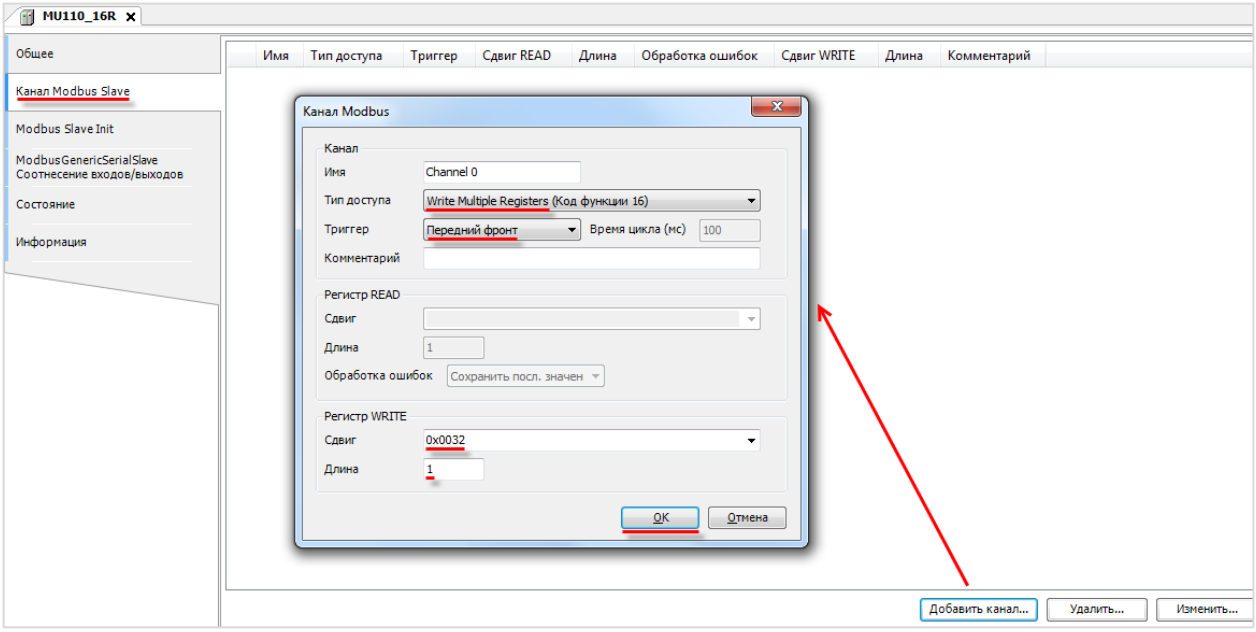


Рисунок 4.9.19 – Добавление канала в конфигурацию slave-устройства MU110_16R

На вкладке **ModbusGenericSerialSlave Соотнесение входов/выходов** следует привязать к каналу переменную **wDO** и триггерную переменную **xTrigger**. Для параметра **Всегда обновлять переменные** следует установить значение **Включено 2**.

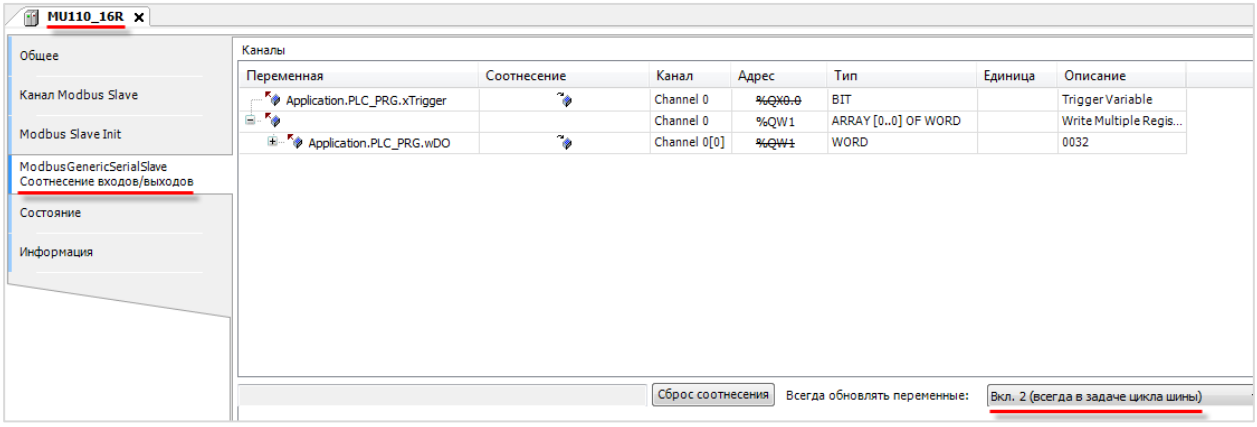


Рисунок 4.9.20 – Привязка переменных к каналу

12. Загрузить проект в контроллер и запустить его.

ПРИМЕЧАНИЕ

В дереве проекта рядом с модулем **MU110_16R** будет отображаться пиктограмма «ожидание соединения». Это связано с тем, что запись в модуль производится по триггеру, и в данный момент обмен с модулем отсутствует. После первой записи в модуль статус связи изменится на «связь установлена».

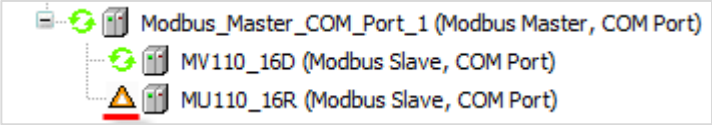


Рисунок 4.9.21 – Пиктограммы статусов связи

В переменной **_2WORD_TO_REAL.rRealValue** будет отображаться текущее значение первого аналогового входа модуля **MB110-8A**. В нулевом бите переменной **wDI (wDI.0)** будет отображаться текущее значение первого дискретного входа модуля **MB110-16Д**.

Если значение **_2WORD_TO_REAL.rRealValue** превысит **30** и при этом значение **wDI.0** будет равно **TRUE**, то в нулевой бит переменной **wDO (wDO.0)** будет однократно (по триггеру) записано значение **TRUE,** что приведет к замыканию первого дискретного выхода модуля **МУ110-16Р**. Если одно из условий перестанет выполняться, то выход будет разомкнут.

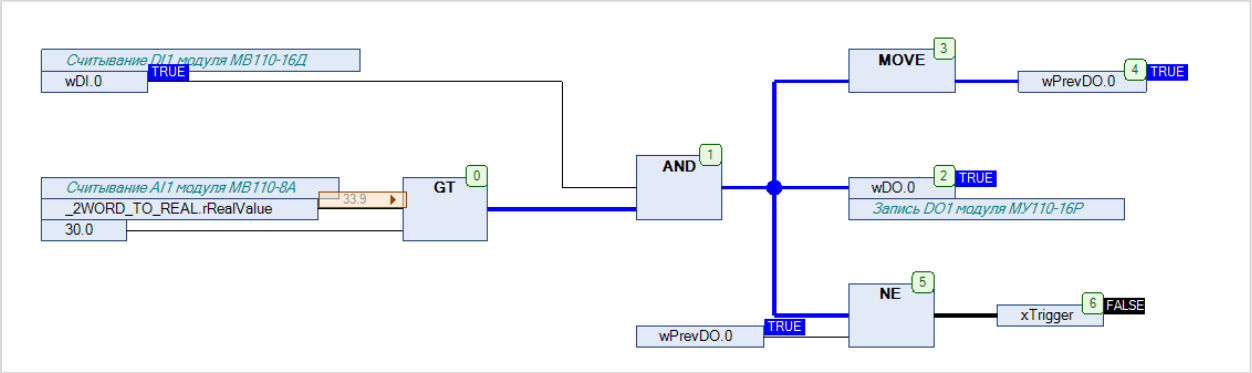


Рисунок 4.9.22 – Выполнение программы в режиме Online

4.10 Пример: СПК1xx [M01] (Modbus RTU Slave) + MasterOPC Universal Modbus Server

В качестве примера будет рассмотрена настройка обмена с OPC-сервером Insat MasterOPC Universal Modbus Server, который будет использоваться в режиме **Modbus RTU Master**.

Структурная схема примера приведена на рисунке ниже:

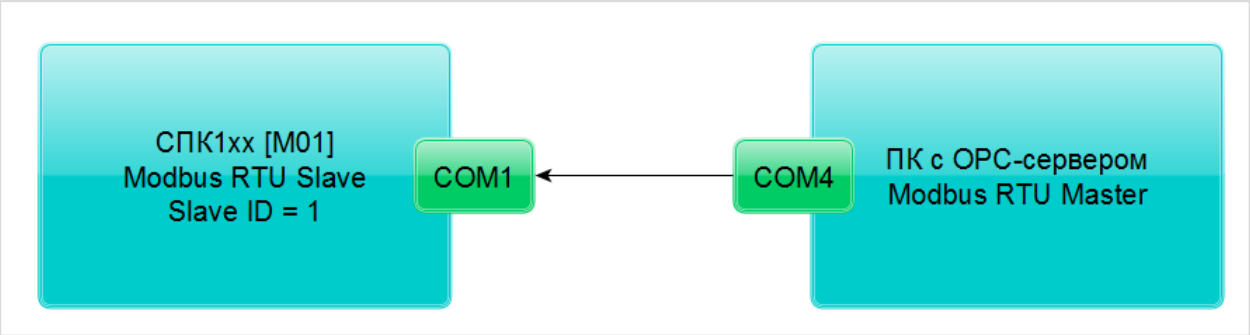


Рисунок 4.10.1 – Структурная схема примера

Пример создан в среде **CODESYS V3.5 SP17 Patch 3** и подразумевает запуск на **СПК1xx [M01]** с таргет-файлом **3.5.17.31.** 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_CodesysModbusRtuSlave_3517v1.zip

Сетевые параметры устройств приведены в таблице ниже:

Таблица 4.10.1 – Сетевые параметры устройств

Параметр	СПК1xx [M01]	ПК с OPC-сервером
COM-порт контроллера, к которому подключен модуль	COM1	COM4
ID COM-порта	1	4
Режим работы	slave	master
Slave ID	1	-
Скорость обмена	115200	
Количество бит данных	8	
Контроль четности	Отсутствует	
Количество стоп-бит	1	

Переменные примера описаны в таблице ниже:

Таблица 4.10.2 – Список переменных примера

Имя	Тип	Область памяти Modbus	Адрес регистра/бита
Переменные Modbus Slave			
xVar_Modbus	BOOL	Coils	0/0
wVar_Modbus	WORD	Holding регистры	0
rVar_Modbus	REAL		1–2
sVar_Modbus	STRING(16)		3–10
Переменные программы для записи в Modbus Slave			
xVar_Prg	BOOL		
wVar_Prg	WORD		
rVar_Prg	REAL		
sVar_Prg	STRING(16)		

В рамках примера области **Coils** и **Holding** регистров являются независимыми (это определяется в настройках компонента **Modbus Serial Device**).

Для настройки обмена следует:

1. Подключить контроллер к ПК (например, с помощью конвертера ОВЕН АС4).
2. Создать новый проект **CODESYS** с программой **PLC_PRG** на языке **CFC**:

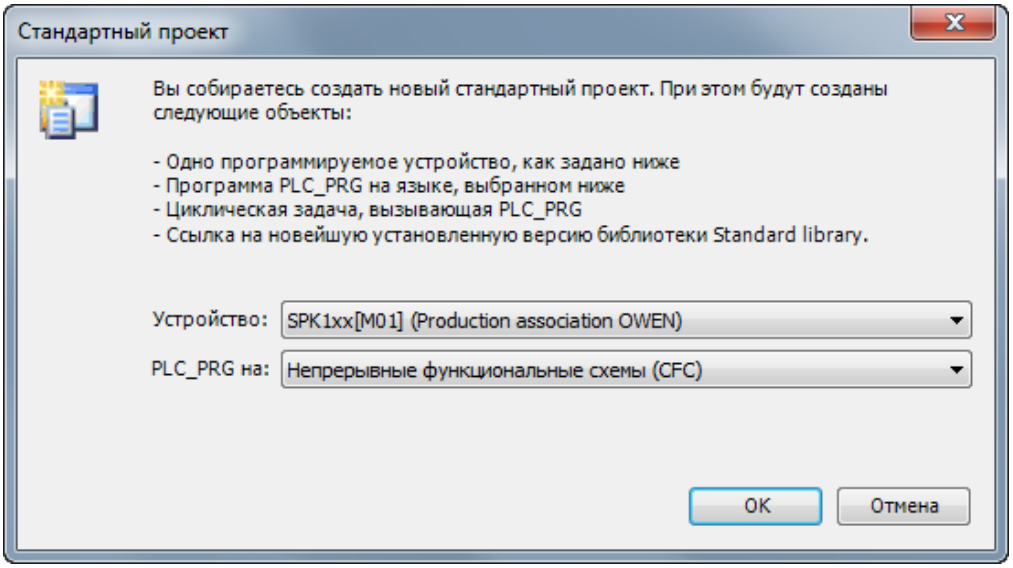


Рисунок 4.10.2 – Создание проекта CODESYS

3. Добавить в проект объединения с именами **Real_Word** и **String_Word**:

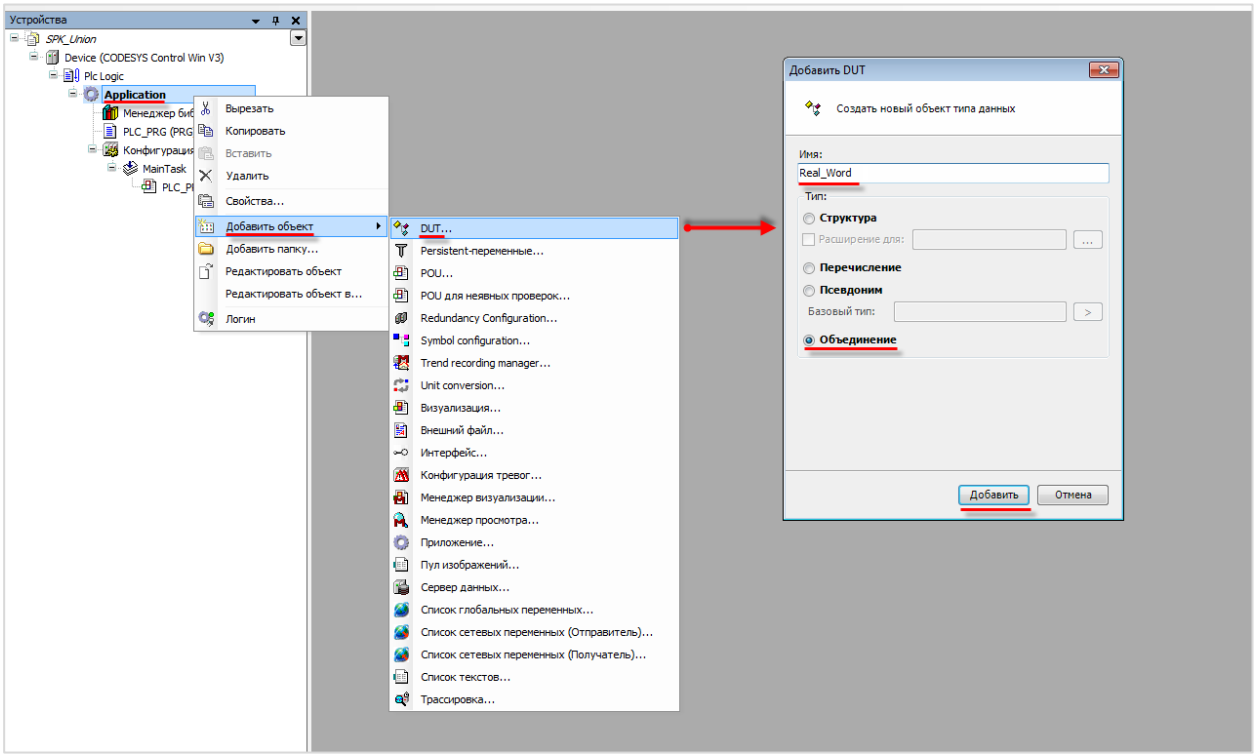


Рисунок 4.10.3 – Добавление в проект объединения Real_Word

Объединения потребуются для преобразования переменных типов **REAL** и **STRING** в набор переменных типа **WORD** для привязки к компоненту **Modbus Serial Device**.

4. В объединениях объявить следующие переменные:

Real_Word

```
TYPE Real_Word :
UNION
    rRealValue      :REAL;
    awModbusReal    :ARRAY [0..1] OF WORD;
END_UNION
END_TYPE
```

Рисунок 4.10.4 – Объявление переменных объединения Real_Word

String_Word

```
TYPE String_Word :
UNION
    awModbusString: ARRAY [0..7] OF WORD;
    sStringValue:   STRING(16);
END_UNION
END_TYPE
```

Рисунок 4.10.5 – Объявление переменных объединения String_Word

5. В **менеджере библиотек** добавить библиотеку OwenCommunication.

6. В программе **PLC_PRG** объявить переменные в соответствии с таблицей 4.10.2.

PLC_PRG

```
PROGRAM PLC_PRG
VAR
    xVar_Modbus:            BOOL;
    wVar_Modbus:            WORD;
    rVar_Modbus:            REAL;
    sVar_Modbus:            STRING(15);
    uRealWord_Modbus:       Real_Word;
    uStringWord_Modbus:     String_Word;


    xWriteToSlave:          BOOL;
    fbWriteToSlave:         R_TRIG;

    xVar_Prg:               BOOL;
    wVar_Prg:               WORD;
    rVar_Prg:               REAL;
    sVar_Prg:               STRING(15);
    uRealWord_Prg:          Real_Word;
    uStringWord_Prg:        String_Word;
END_VAR
```

Рисунок 4.10.6 – Объявление переменных программы

Код программы будет выглядеть следующим образом:

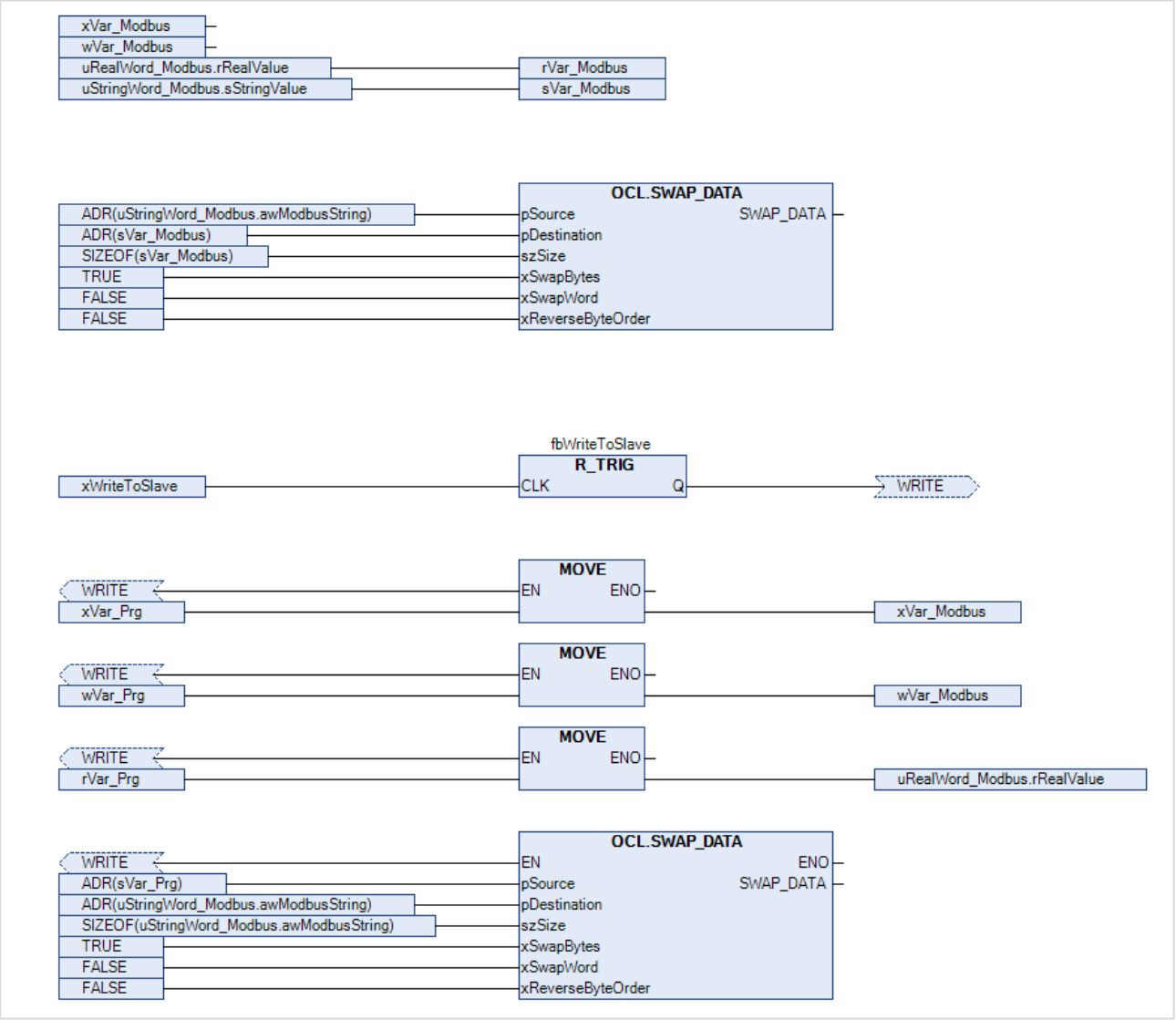


Рисунок 4.10.7 – Код программы PLC_PRG

Функция **SWAP_DATA** из библиотеки **OwenCommunication** используется для изменения порядка байтов в переменной типа **STRING** для соответствия порядку байтов в OPC-сервере (функционал перестановки байт в OPC-сервере не распространяется на тип STRING).

7. Добавить компоненты **Modbus COM** и **Modbus Serial Device** в соответствии с п. 4.3. Настроить компоненты в соответствии с таблицей 4.10.1.

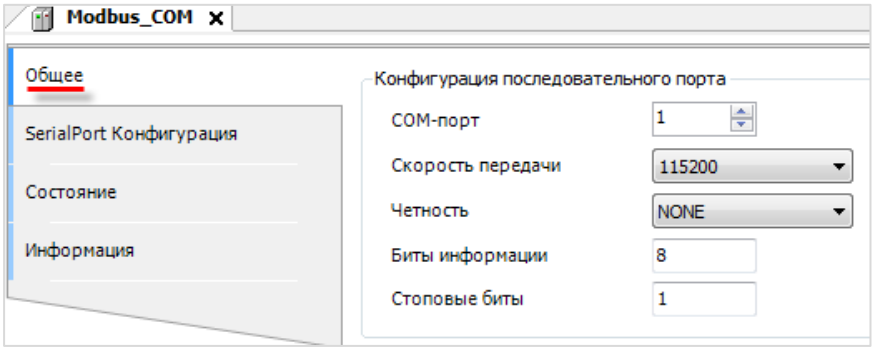


Рисунок 4.10.8 – Настройки компонента Modbus COM

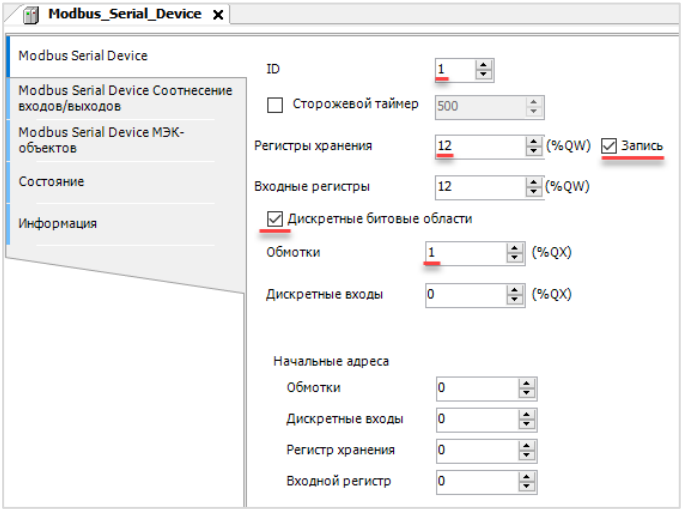


Рисунок 4.10.9 – Настройки компонента Modbus Serial Device

8. Привязать к каналам компонента **Modbus Serial Device** переменные программы в соответствии с таблицей 4.10.2. Установить галочку **Вкл. 2 (Всегда в задаче цикла шины)**.

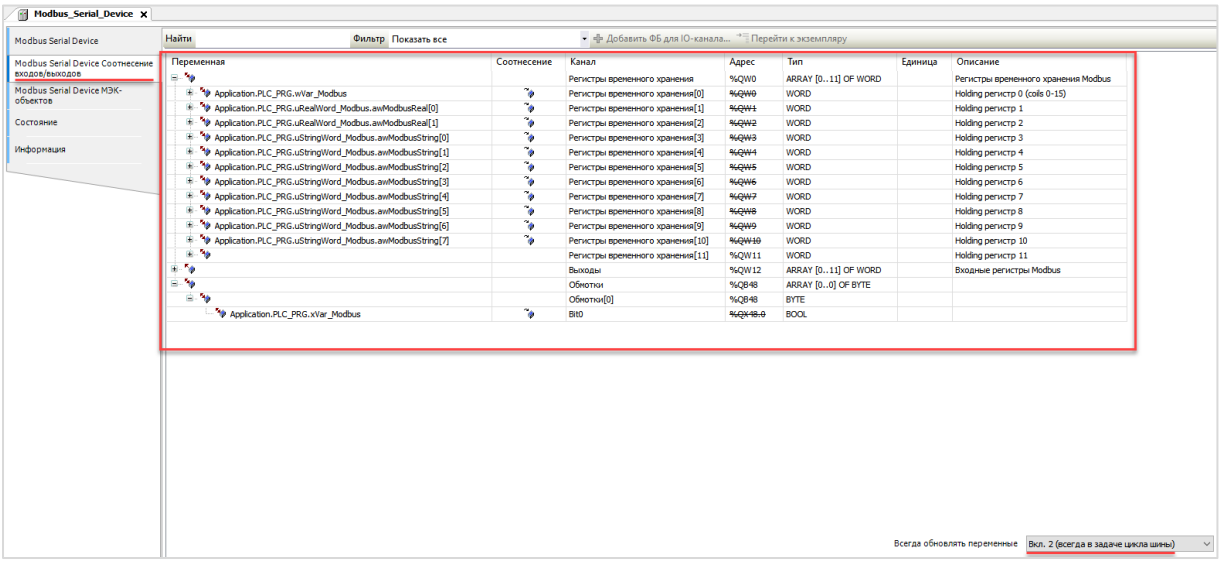


Рисунок 4.10.10 – Привязка переменных к компоненту Modbus Serial Device

9. Установить и запустить MasterOPC Universal Modbus Server.

10. Нажать **ПКМ** на узел **Server** и добавить коммуникационный узел типа **COM**. В узле следует указать сетевые настройки в соответствии с таблицей 4.10.1. Для работы OPC-сервера в режиме **Modbus RTU Master** параметры **Использовать режим ASCII** и **Slave подключение** должны иметь значение **FALSE**.

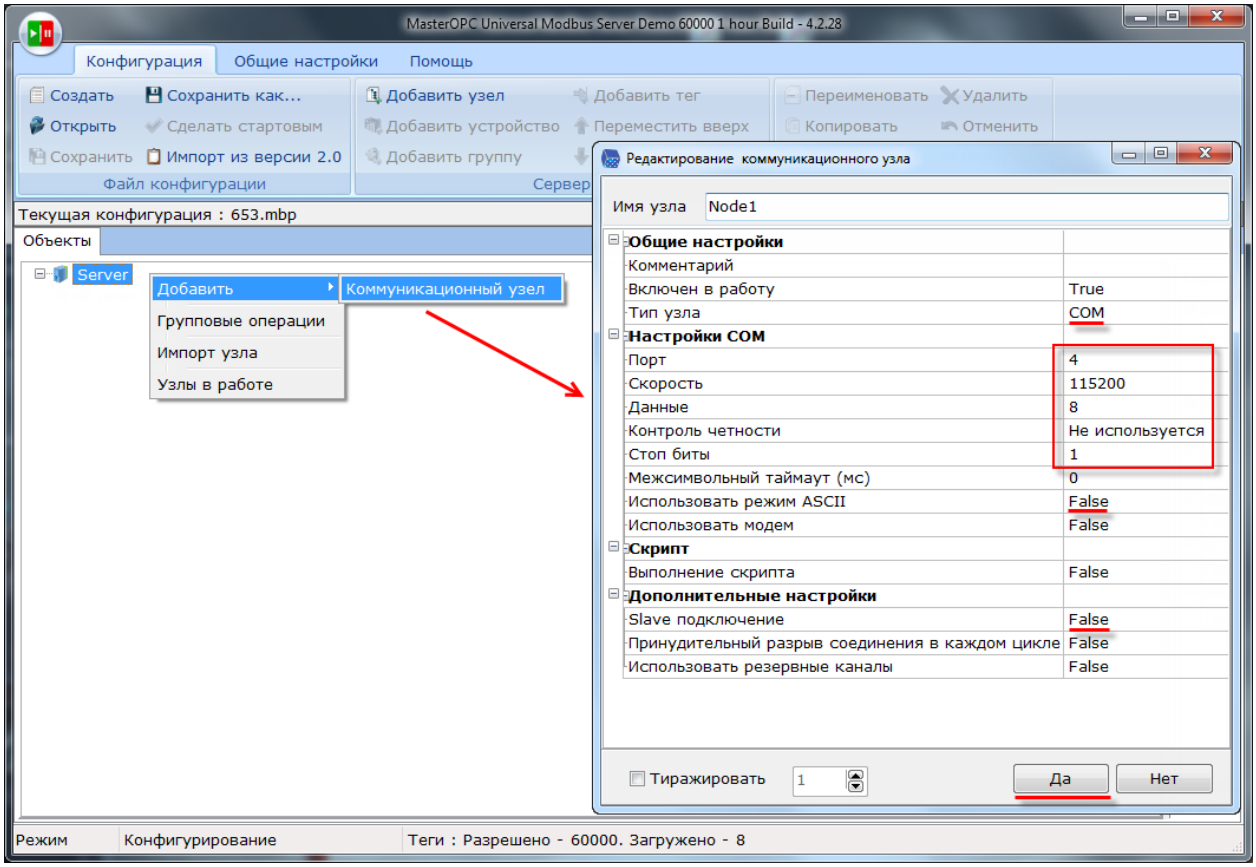


Рисунок 4.10.11 – Добавление коммуникационного узла

11. Нажать **ПКМ** на коммуникационный узел и добавить устройство с настройками по умолчанию (Slave ID = **1** в соответствии с [таблицей 4.10.1](#)).

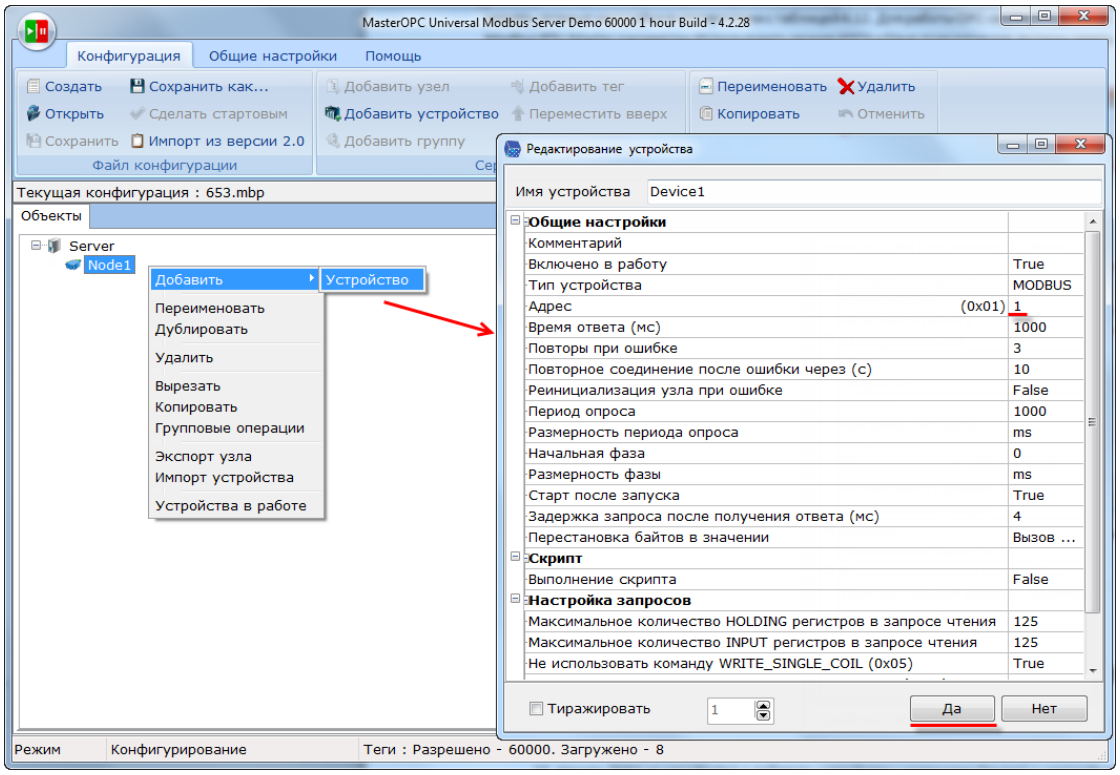


Рисунок 4.10.12 – Добавление устройства

12. Нажать **ПКМ** на устройство и добавить 4 тега. Число тегов соответствует числу переменных, считываемых/записываемых OPC-сервером. Настройки тегов приведены ниже.

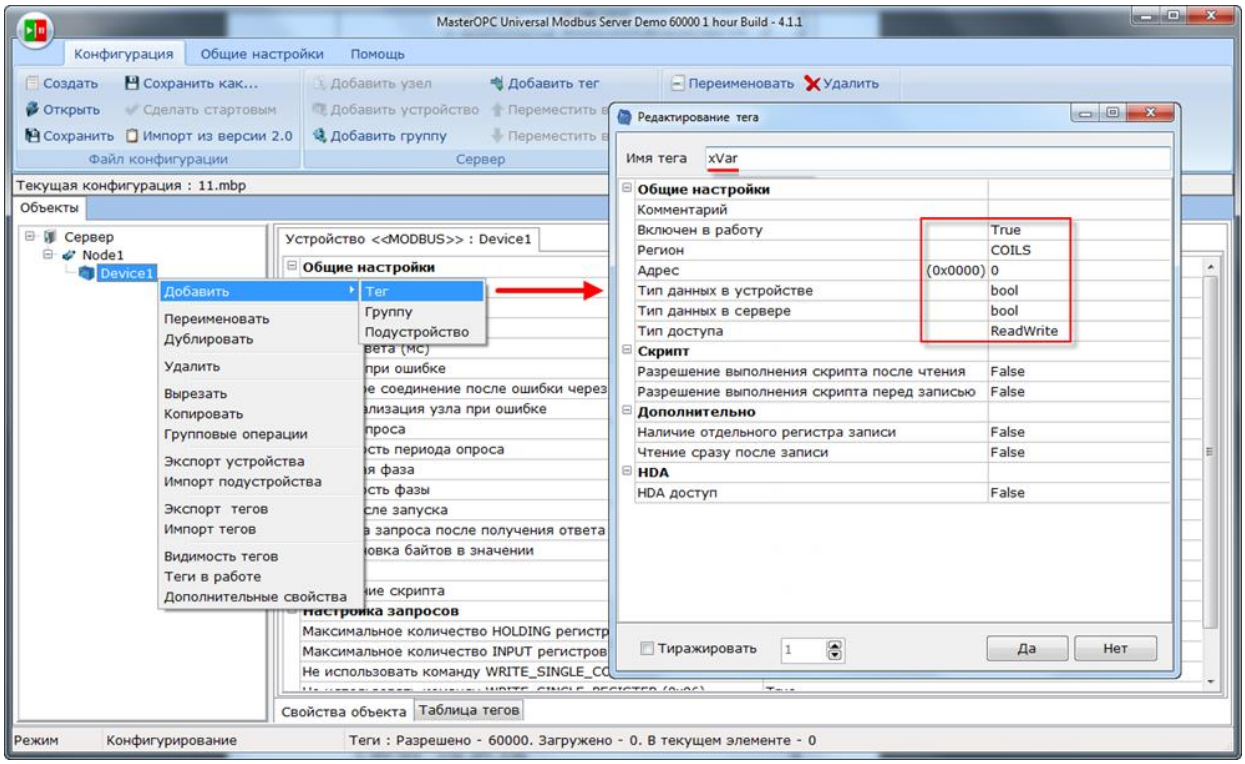


Рисунок 4.10.13 – Добавление тега xVar

Тег <<HOLDING_REGISTERS>> : wVar

Общие настройки	
Комментарий	
Включен в работу	True
Адрес (0x0000)	0
Тип данных в устройстве	uint16
Тип данных в сервере	uint32
Тип доступа	ReadWrite
Использовать перестановку байтов устройства	True
Последний тег в групповом запросе	False
Пересчет (A*X + B)	False

Рисунок 4.10.14 – Добавление тега wVar

Тег <<HOLDING_REGISTERS>> : rVar

Общие настройки	
Комментарий	
Включен в работу	True
Адрес (0x0001)	1
Тип данных в устройстве	float
Тип данных в сервере	float
Тип доступа	ReadWrite
Использовать перестановку байтов устройства	False
Перестановка байтов в значении	10325476
Последний тег в групповом запросе	False
Пересчет (A*X + B)	False

Рисунок 4.10.15 – Добавление тега rVar

Тег <<HOLDING_REGISTERS>> : sVar

Общие настройки	
Комментарий	
Включен в работу	True
Адрес (0x0003)	3
Тип данных в устройстве	string
Тип данных в сервере	string
Количество байт для строкового типа	16
Тип строки для строкового типа	ascii
Тип доступа	ReadWrite
Использовать перестановку байтов устройства	True
Последний тег в групповом запросе	False
Пересчет (A*X + B)	False

Рисунок 4.10.16 – Добавление тега sVar

13. Загрузить проект в контроллер и запустить его. Запустить OPC-сервер для контроля значений переменных.

В редакторе CODEYS следует изменить значения **_Prg** переменных, сгенерировать передний фронт в переменной **xWriteToSlave** и наблюдать соответствующие изменения в OPC-сервере. В OPC-сервере следует изменить значения переменных и наблюдать соответствующие значения в **_Modbus** переменных CODESYS.

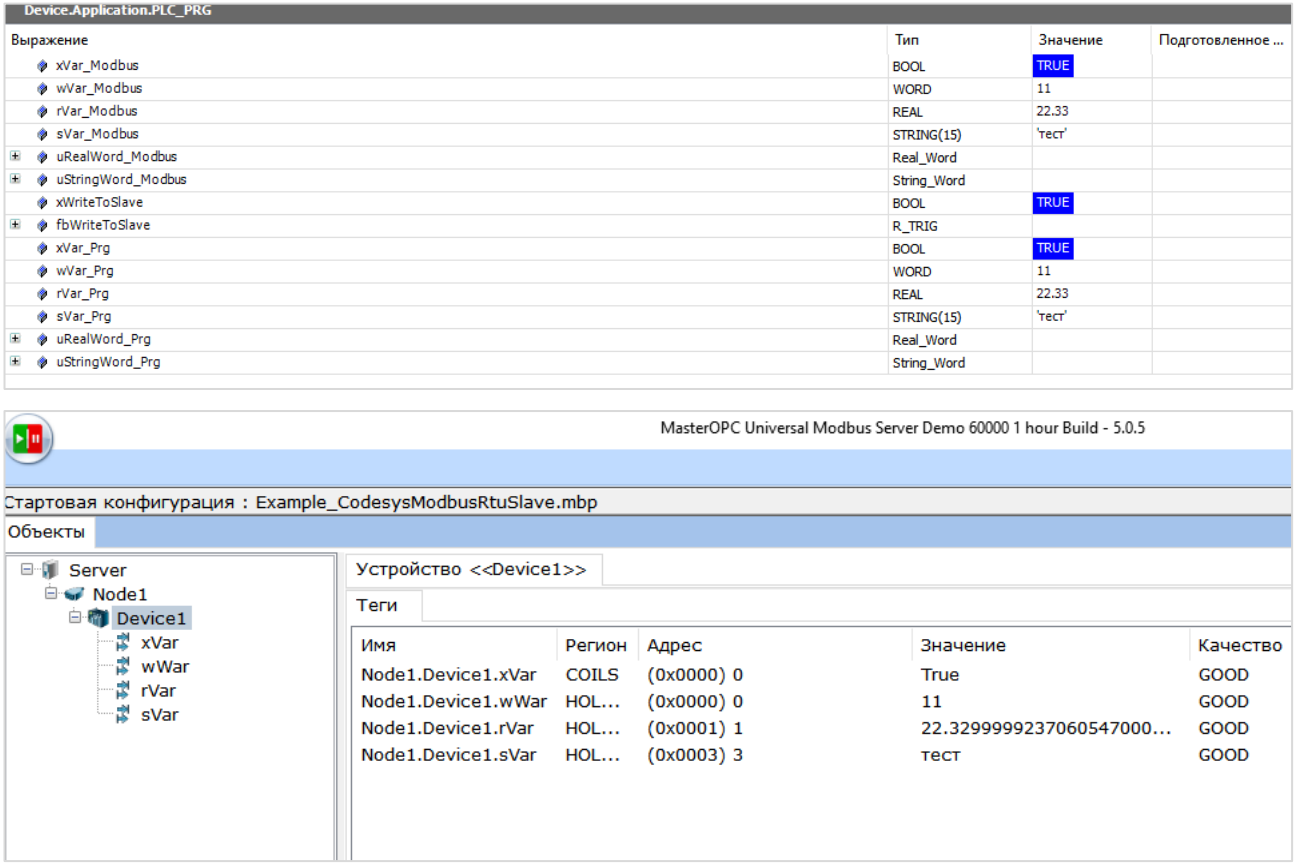


Рисунок 4.10.17 – Чтение и запись данных через OPC-сервер

4.11 Пример: СПК1xx [M01] (Modbus TCP Master) + модули Mx210

В качестве примера будет рассмотрена настройка обмена с модулями Mx210 (МВ210-101 и МК210-301) с использованием **стандартных средств конфигурации**.

Реализуемый алгоритм: если значение первого аналогового входа модуля **МВ210-101** превышает **30** и при этом первый дискретный вход модуля **МК210-301** имеет значение **TRUE** (замкнут), то первому дискретному выходу модуля **МК210-301** присваивается значение **TRUE** (замкнут). Во всех остальных случаях дискретному выходу присваивается значение **FALSE** (разомкнут).

Структурная схема примера приведена на рисунке ниже:

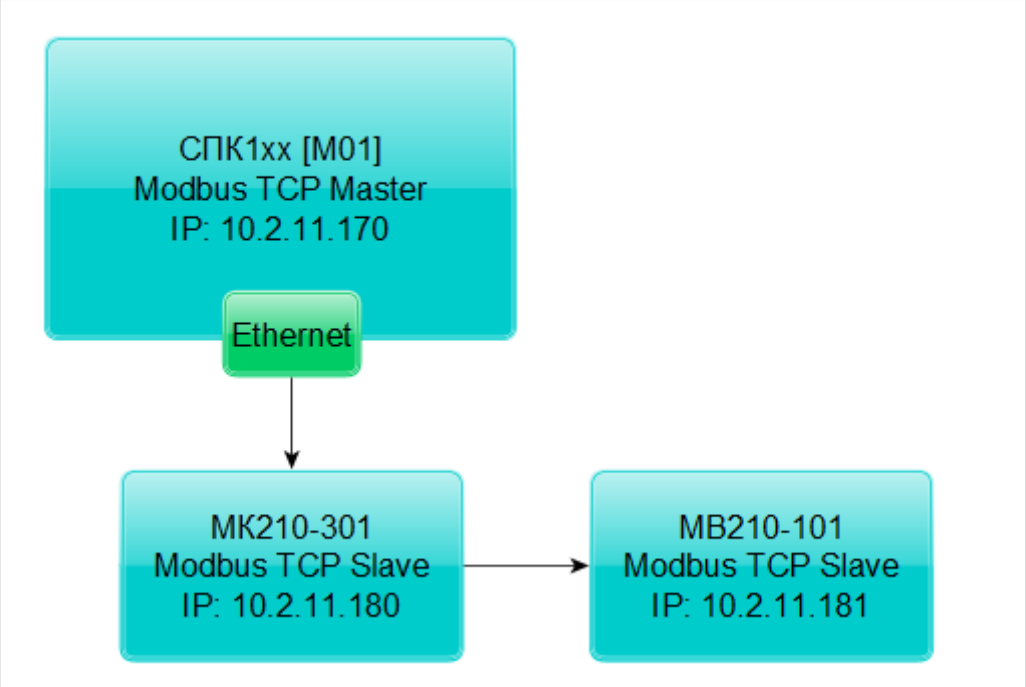


Рисунок 4.11.1 – Структурная схема примера

Пример создан в среде **CODESYS V3.5 SP17 Patch 3** и подразумевает запуск на **СПК1xx [M01]** с таргет-файлом **3.5.17.31.** 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_CodesysModbusTcpMasterMx210_3517v1.projectarchive

Видеоверсия примера доступна по ссылке.

Сетевые параметры устройств приведены в таблице ниже:

Таблица 4.11.1 – Сетевые параметры устройств

Параметр	СПК1xx [M01]	МК210-301	МВ210-101
Режим работы	master	slave	slave
IP-адрес	10.2.11.170	10.2.11.180	10.2.11.181
Маска подсети	255.255.0.0		
IP-адрес шлюза	10.2.1.1		
Порт	502		
Unit ID	-	1	1

Переменные примера описаны в таблице ниже:

Таблица 4.11.2 – Список переменных примера

Модуль	Имя переменной	Тип	Описание
МВ210-101	awModbusReal	ARRAY [0..1] OF WORD	Значение температуры в виде двух **WORD**, считываемое с модуля
	rRealValue	REAL	Значение температуры в виде числа с плавающей точкой для использования в программе
МК210-301	wDI	WORD	Значение дискретных входов в виде битовой маски. При обращении к отдельным входам указывается их номер, начиная с 0: **wDI.0** – состояние первого входа (TRUE/FALSE) **wDI.1** – состояние второго входа …
	wDO	WORD	Значение дискретных выходов в виде битовой маски. При обращении к отдельным выходам указывается их номер, начиная с 0: **wDO.0** – состояние первого выхода (TRUE/FALSE) **wDO.1** – состояние второго выхода …
-	wPrevDO	WORD	Значение дискретных выходов в виде битовой маски из предыдущего цикла программы. Используется для отправки команды записи только в случае изменения значений выходов (иначе будет производиться циклическая запись последнего значения)
-	xTrigger	BOOL	Триггерная переменная, управляющая функцией записи дискретного выхода (запись происходит по переднему фронту переменной)

Для настройки обмена следует:

1. Настроить модули **Mx210** с помощью программы **ОВЕН Конфигуратор** в соответствии с таблицей 4.11.1 (см. руководство **Mx210. Примеры настройки обмена**). Подключить модули к контроллеру.

2. Создать новый проект **CODESYS** с программой **PLC_PRG** на языке **CFC**:

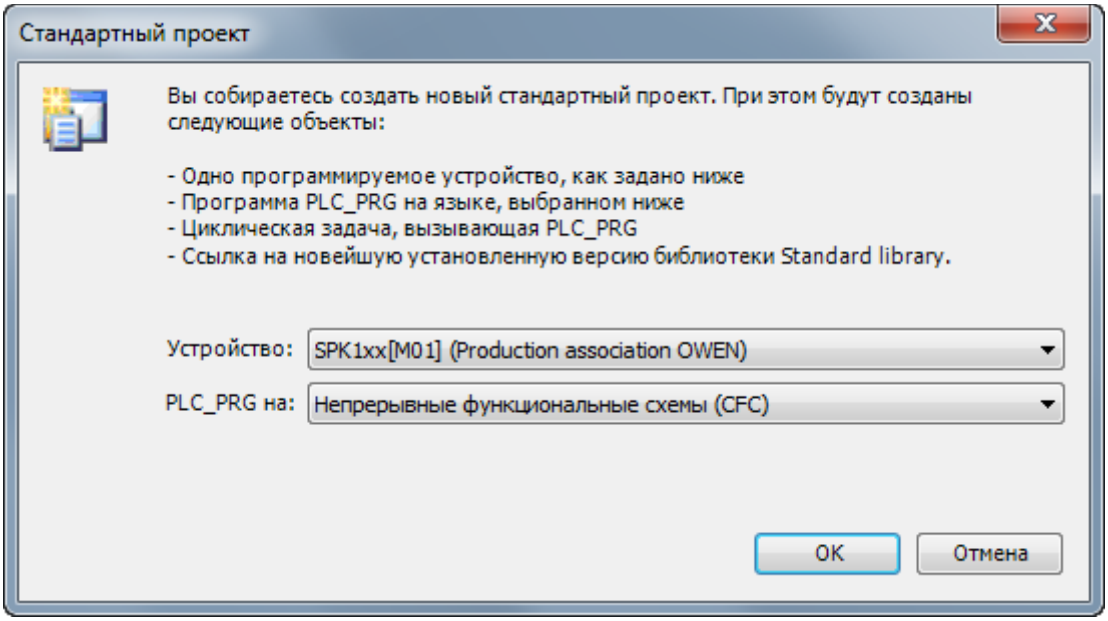


Рисунок 4.11.2 – Создание проекта CODESYS

3. Добавить в проект объединение с именем **Real_Word**:

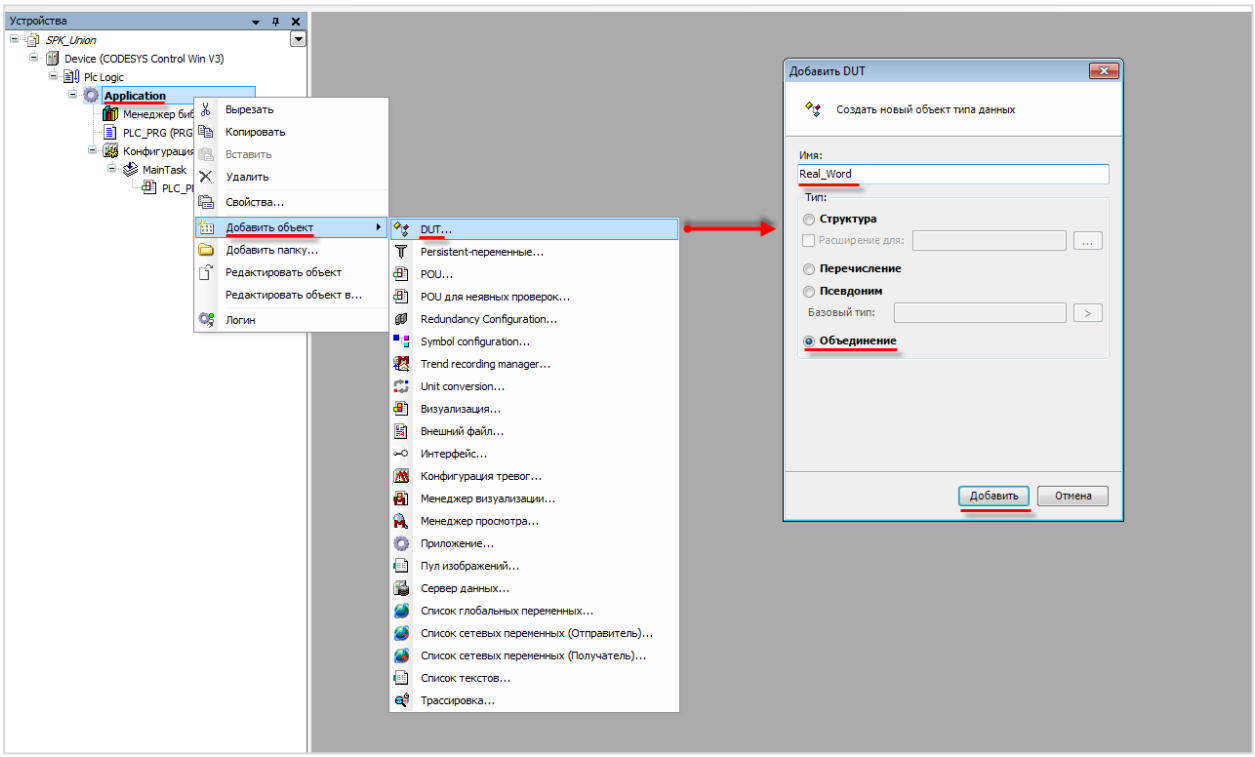


Рисунок 4.11.3 – Добавление в проект объединения

4. В объединении объявить переменную **rRealValue** типа **REAL** и массив **awModbusReal** типа **WORD**, содержащий два элемента:

Real_Word

```
1  TYPE Real_Word :
2  UNION
3      rRealValue      :REAL;
4      awModbusReal    :ARRAY [0..1] OF WORD;
5  END_UNION
6  END_TYPE
```

Рисунок 4.11.4 – Объявление переменных объединения

5. В программе **PLC_PRG** объявить экземпляр объединения **Real_Word** с названием **_2WORD_TO_REAL,** переменные **wDI, wDO** и **wPrevDO** типа **WORD** и переменную **xTrigger** типа **BOOL**. Описание переменных приведено в таблице 4.11.2.

PLC_PRG

```
1  PROGRAM PLC_PRG
2  VAR
3      _2WORD_TO_REAL:     Real_Word;  // значение 1-го входа MB210-101
4      wDI:                WORD;       // битовая маска входов MK210-301
5      wDO:                WORD;       // битовая маска выходов MK210-301
6      wPrevDO:            WORD;       // битовая маска предыдущей записи выходов MK210-301
7      xTrigger:           BOOL;       // триггер записи выходов
8  END_VAR
```

Рисунок 4.11.5 – Объявление переменных программы

Код программы будет выглядеть следующим образом:

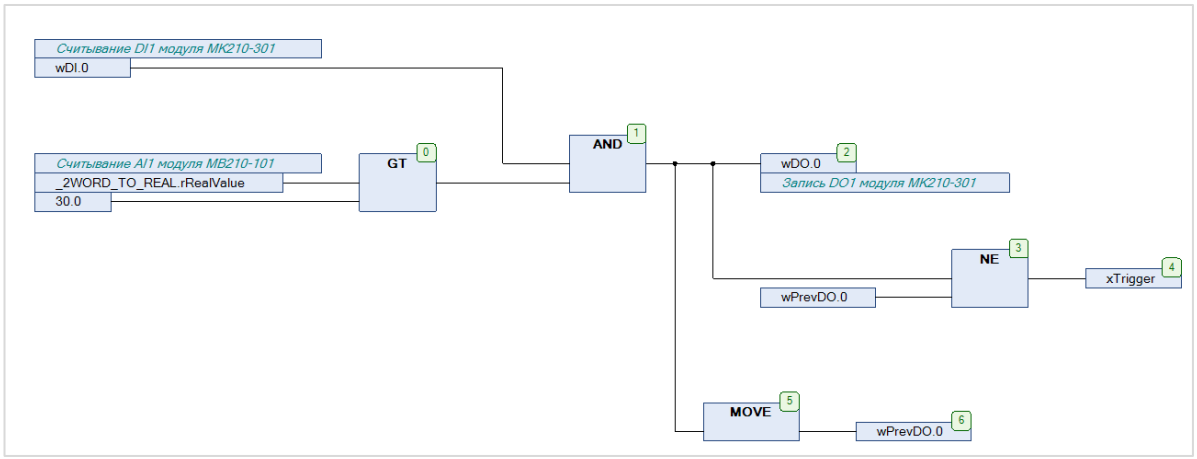


Рисунок 4.11.6 – Код программы PLC_PRG

Программа работает следующим образом: если значение переменной **rRealValue** (связанной с первым аналоговым входом модуля **MB210-101)** превышает **30** и при этом значение нулевого бита переменной **wDI** (связанной с первым дискретным входом модуля **MK210-301**) имеет значение **TRUE**, то нулевому биту переменной **wDO** присваивается значение **TRUE**. Если на предыдущем цикле значение нулевого бита **wDO** отличалось от текущего, то переменная **xTrigger** принимает значение **TRUE**, что приводит к однократной записи текущего значения бита в первый дискретный выход модуля **MK210-301**.

6. Добавить в проект компонент **Ethernet**.

ПРИМЕЧАНИЕ

Версия компонента должна соответствовать версии таргет-файла. Для отображения всех доступных версий компонента следует установить галочку **Отображать все версии**. См. рекомендации в приложении А.

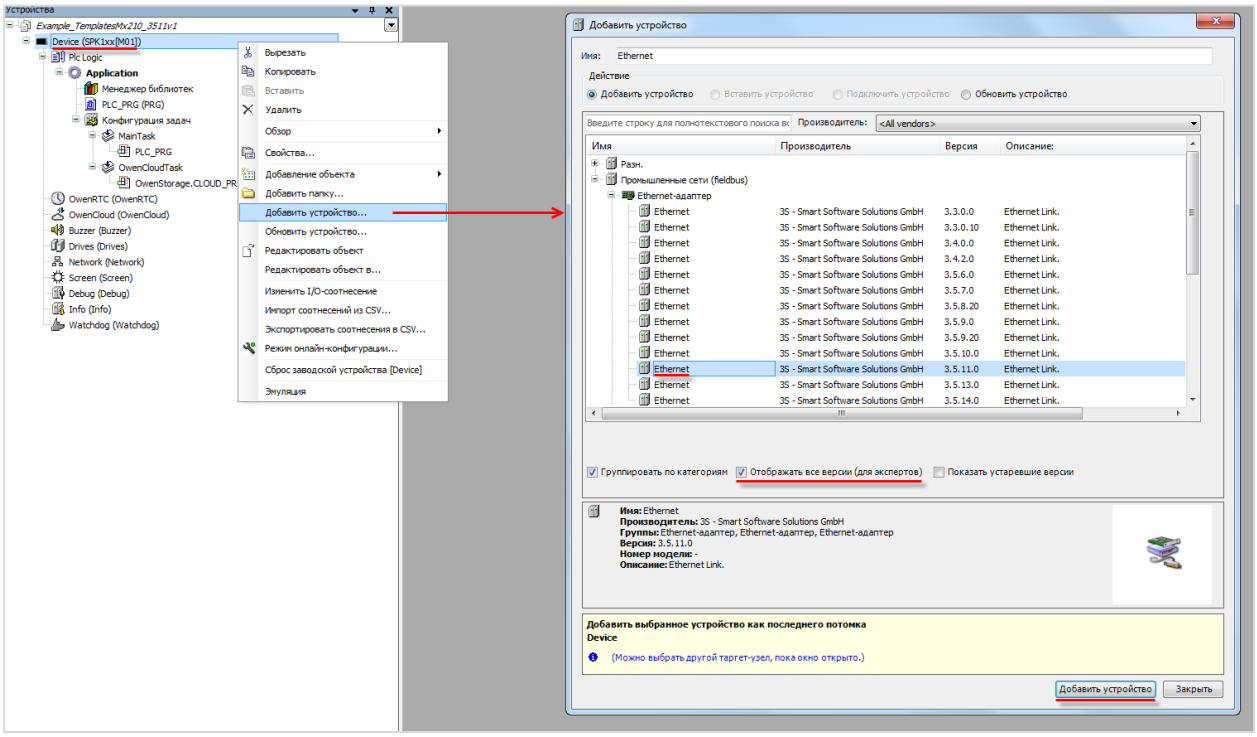


Рисунок 4.11.7 – Добавление компонента Ethernet

Затем следует установить соединение с контроллером, не загружая в него проект (**Device – Установка соединения – Сканировать сеть**) и в компоненте **Ethernet** на вкладке **Конфигурация Ethernet** выбрать нужный интерфейс.

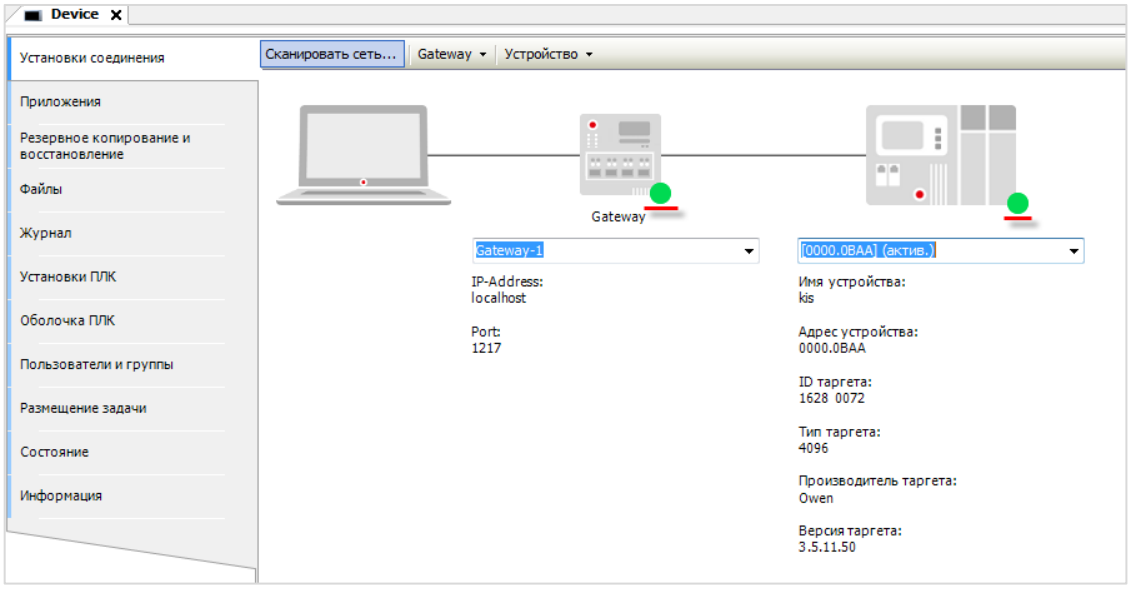


Рисунок 4.11.8 – Подключение к контроллеру

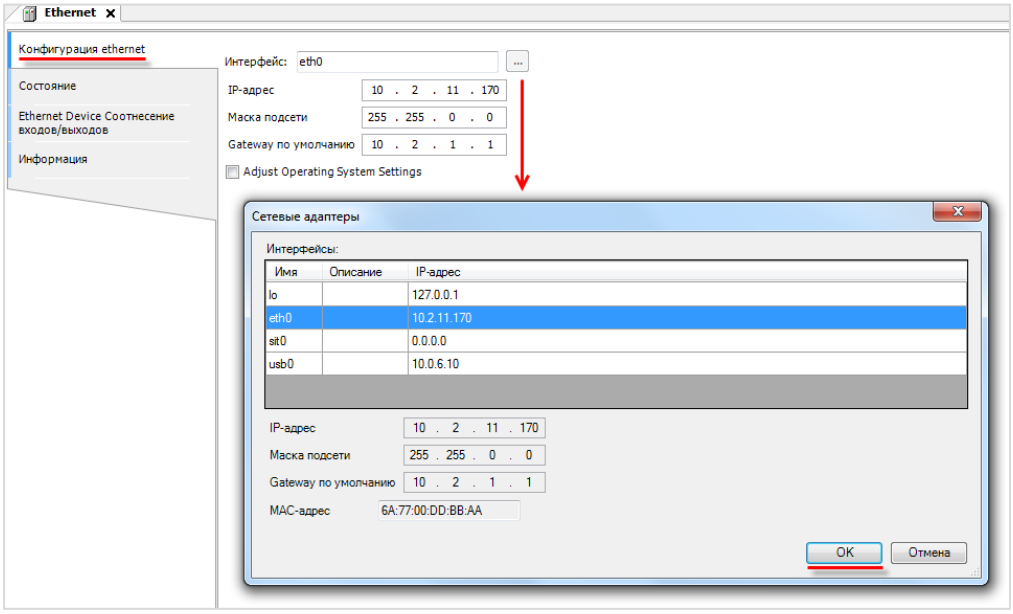


Рисунок 4.11.9 – Выбор используемого интерфейса

ПРИМЕЧАНИЕ
Настройки интерфейса задаются в конфигураторе контроллера (см. документ **CODESYS V3.5. FAQ**).

7. В компонент **Ethernet** добавить компонент **Modbus TCP Master**.

ПРИМЕЧАНИЕ
Версия компонента должна соответствовать версии таргет-файла. Для отображения всех доступных версий компонента следует установить галочку **Отображать все версии**. См. рекомендации в приложении А.

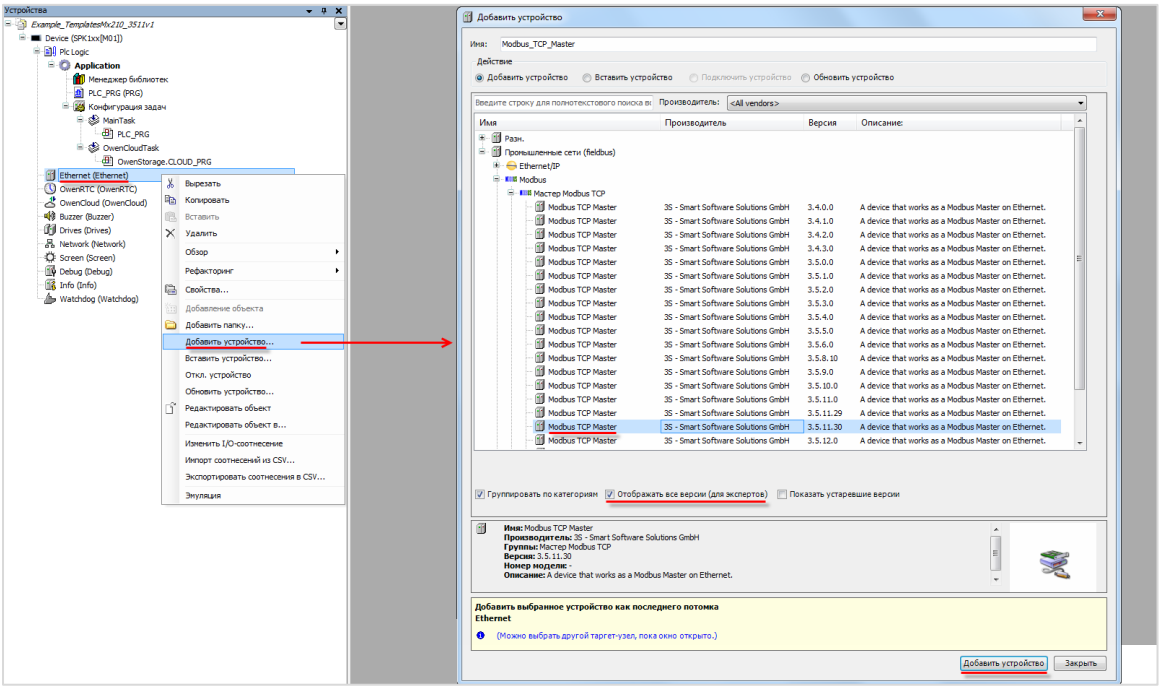


Рисунок 4.11.10 – Добавление компонента Modbus TCP Master

В настройках компонента вкладке **Общее** следует установить галочку **Автоподключение.**

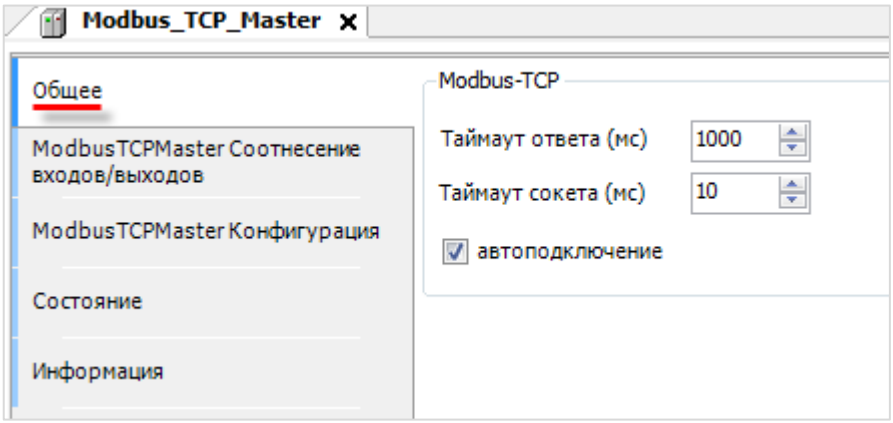


Рисунок 4.11.11 – Настройки компонента Modbus TCP Master

8. В компонент **Modbus TCP Master** следует добавить компоненты **Modbus TCP Slave** с именами **MV210_101** и **MK210_301**.

ПРИМЕЧАНИЕ
Версия компонента должна соответствовать версии таргет-файла. Для отображения всех доступных версий компонента следует установить галочку **Отображать все версии**. См. рекомендации в приложении А.

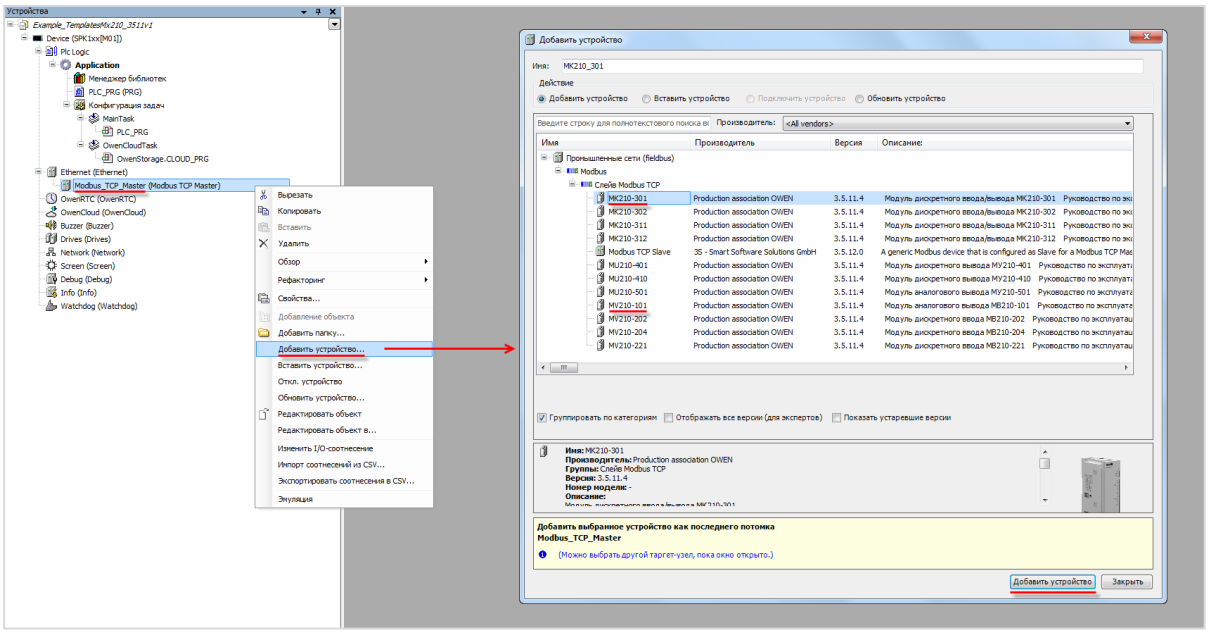


Рисунок 4.11.12 – Добавление slave-устройств в проект CODESYS

В настройках компонентов следует указать IP-адреса согласно таблице 4.11.1 (**МК210-301** – **10.2.11.180, MB210-101** – **10.2.11.181**). На вкладке **Modbus TCPSlave Конфигурация** следует установить для параметра **Unit ID** значение **1**.

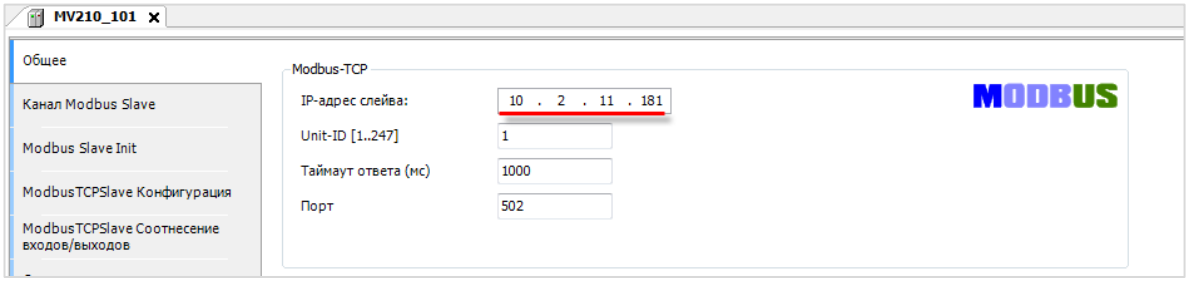


Рисунок 4.11.13 – Настройки компонента Modbus TCP Slave, вкладка Общее

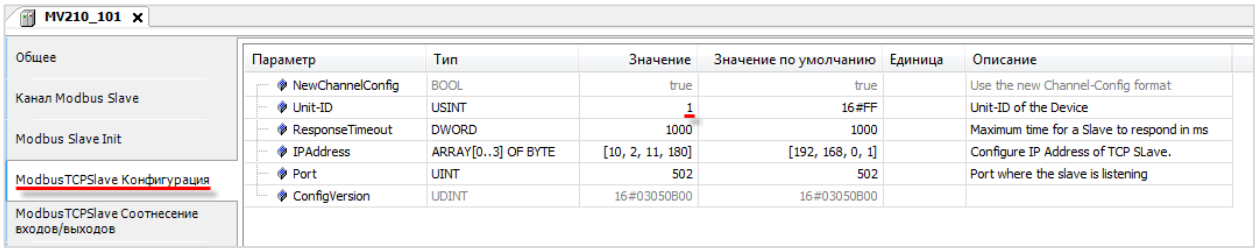


Рисунок 4.11.14 – Настройки компонента Modbus TCP Slave, вкладка Modbus TCPSlave Конфигурация

9. В настройках компонента **MV210_101** на вкладке **Канал Modbus Slave** следует добавить канал, в котором с помощью функции **Read Holding Registers** будет считываться значение **4000** и **4001** регистров модуля. В данных регистрах содержится значение входа 1 в представлении с плавающей точкой. Таблица регистров модуля и поддерживаемые функции приведены в руководстве по эксплуатации.

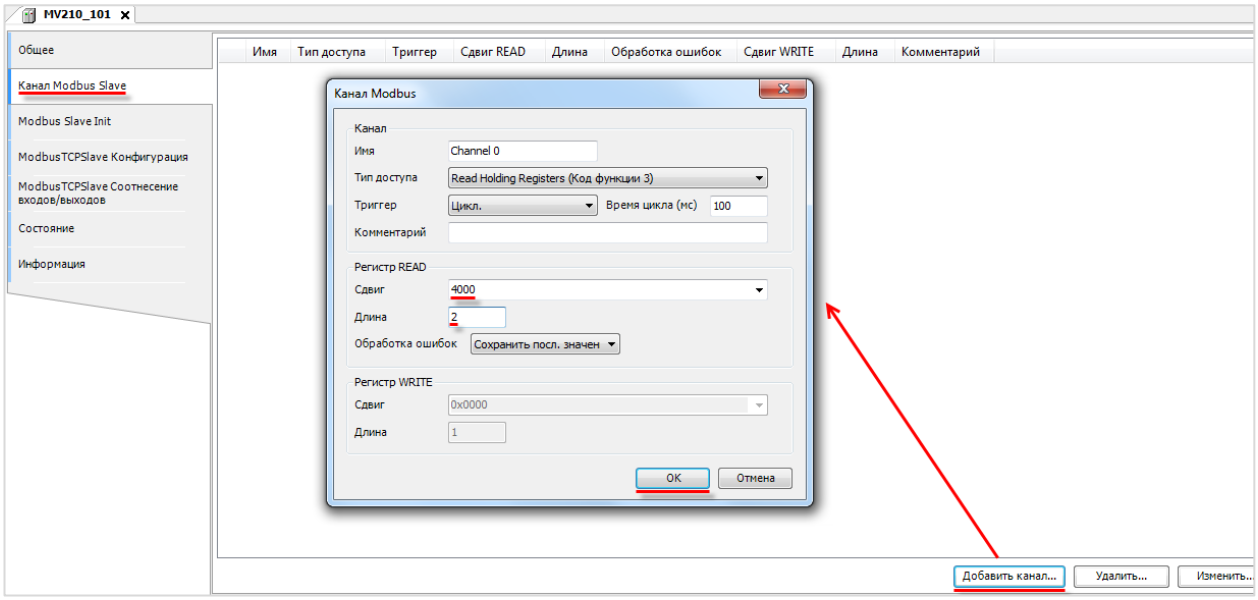


Рисунок 4.11.15 – Добавление канала в конфигурацию slave-устройства MV210_101

На вкладке **ModbusGenericSerialSlave Соотнесение входов/выходов** следует привязать к каналу элементы объединения **_2WORD_TO_REAL**.

Для параметра **Всегда обновлять переменные** следует установить значение **Включено 2**.

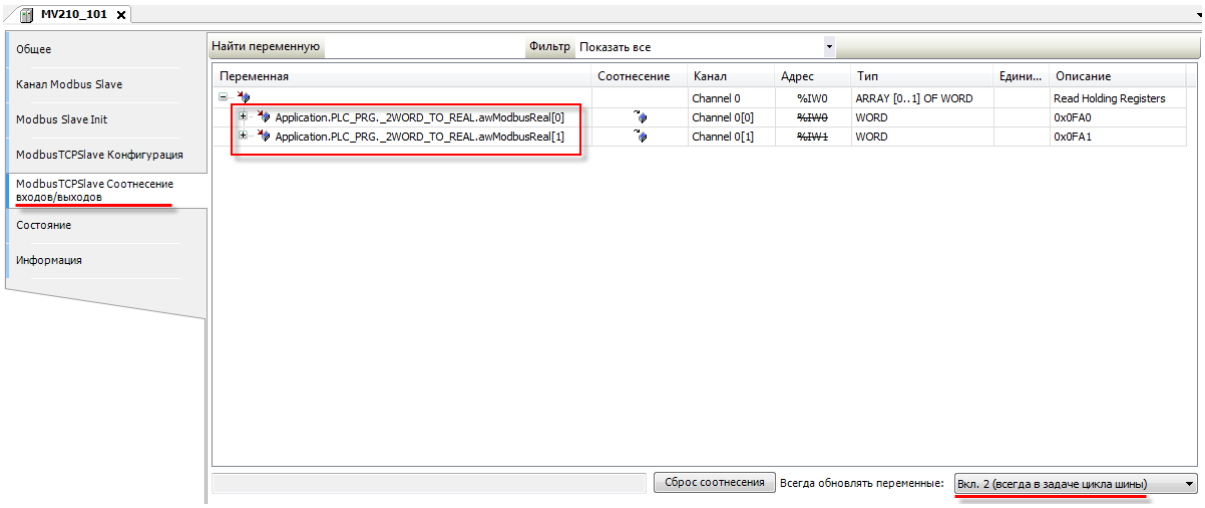
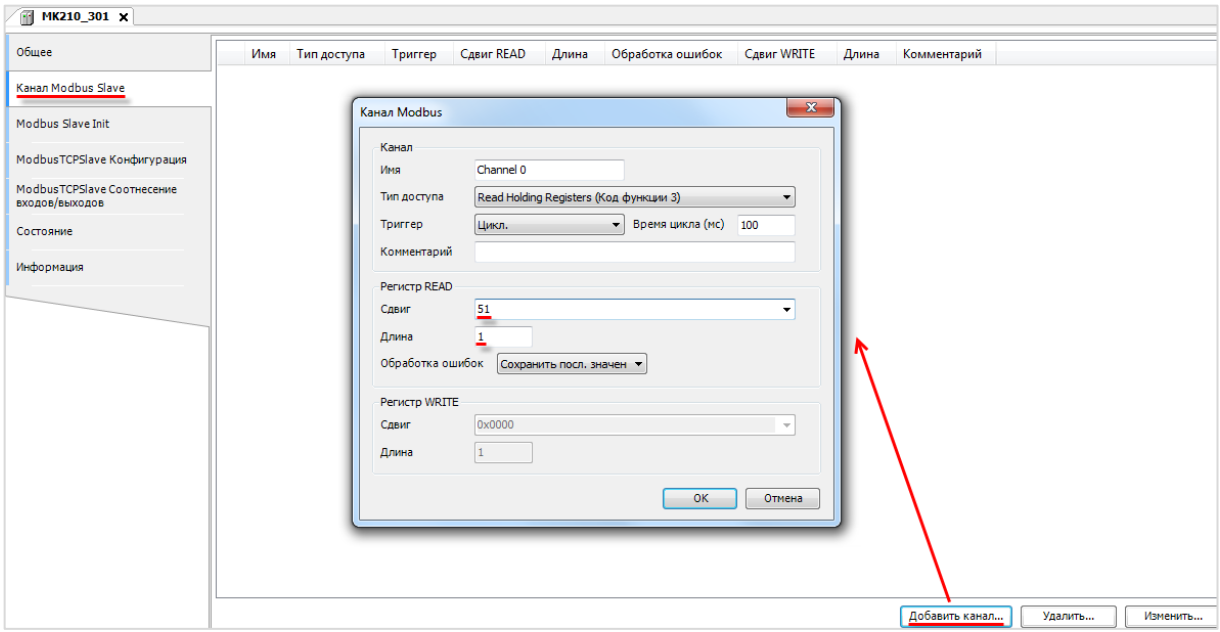


Рисунок 4.11.16 – Привязка переменных к каналу

10. В настройках компонента **MK210_301** на вкладке **Канал Modbus Slave** следует добавить канал, в котором с помощью функции **Read Holding Registers** будет считываться значение регистра **51**. В данном регистре содержится битовая маска состояний дискретных входов. Также следует добавить канал, в котором с помощью функции **Write Multiple Registers** будет записываться значение в регистр **470**. В данном регистре содержатся значения выходов модуля в виде битовой маски. У параметра **Триггер** следует установить значение **Передний фронт**, чтобы иметь возможность управлять записью в модуль с помощью логической переменной.

Таблица регистров модуля и поддерживаемые функции приведены в руководстве по эксплуатации.



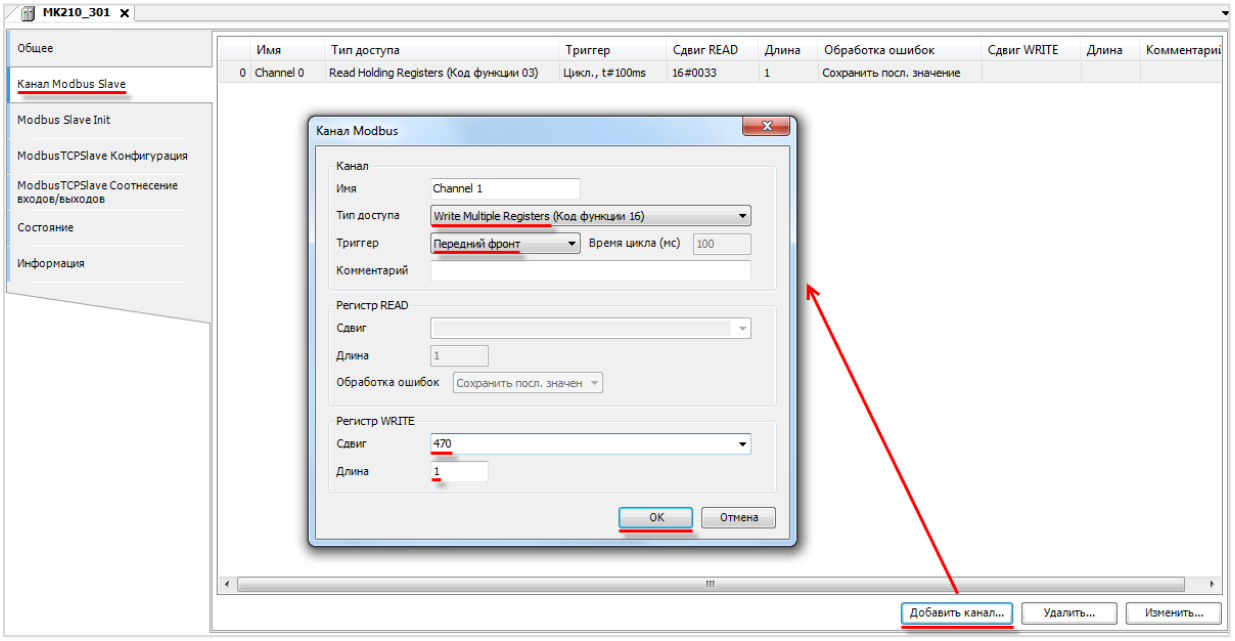


Рисунок 4.11.17 – Добавление каналов в конфигурацию slave-устройства MK210_301

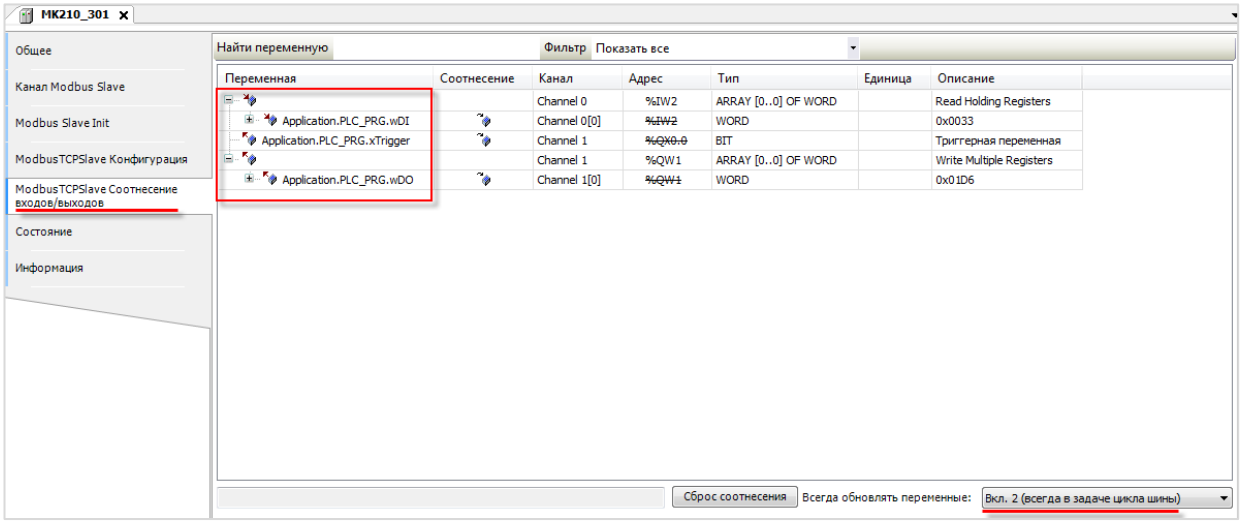


Рисунок 4.11.18 – Привязка переменных к каналам

12. Загрузить проект в контроллер и запустить его.

В переменной **_2WORD_TO_REAL.rRealValue** будет отображаться текущее значение первого аналогового входа модуля **MV210_101**. В нулевом бите переменной **wDI (wDI.0)** будет отображаться текущее значение первого дискретного входа модуля **MK210_301**.

Если значение **_2WORD_TO_REAL.rRealValue** превысит **30** и при этом значение **wDI.0** будет равно **TRUE**, то в нулевой бит переменной **wDO (wDO.0)** будет однократно (по триггеру) записано значение **TRUE,** что приведет к замыканию первого дискретного выхода модуля **MK210_301**. Если одно из условий перестанет выполняться, то выход будет разомкнут.

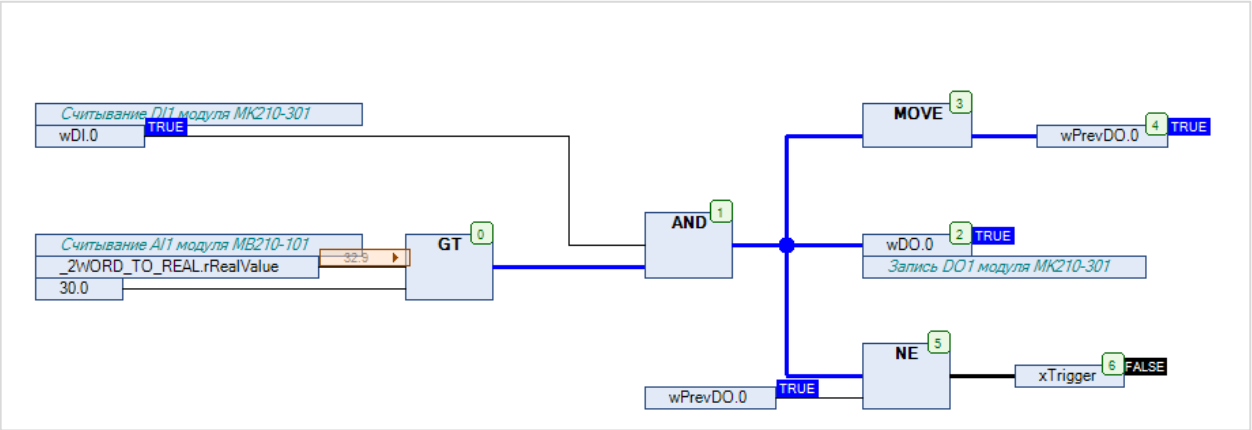


Рисунок 4.11.19 – Выполнение программы в режиме Online

4.12 Пример: СПК1xx [M01] (Modbus TCP Slave) + MasterOPC Universal Modbus Server

В качестве примера будет рассмотрена настройка обмена с OPC-сервером Insat MasterOPC Universal Modbus Server, который будет использоваться в режиме **Modbus TCP Master**.

Структурная схема примера приведена на рисунке ниже:

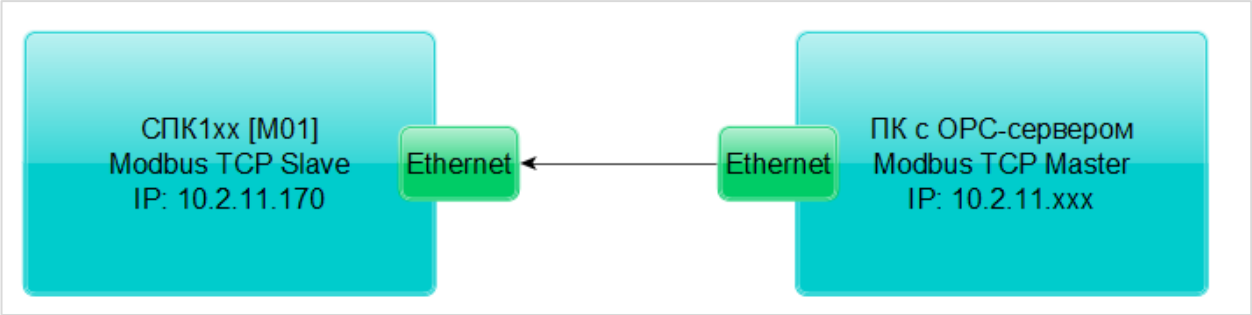


Рисунок 4.12.1 – Структурная схема примера

Пример создан в среде **CODESYS V3.5 SP17 Patch 3** и подразумевает запуск на **СПК1xx [M01]** с таргет-файлом **3.5.17.31.** 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_CodesysModbusTcpSlave_3517v1.zip

Сетевые параметры устройств приведены в таблице ниже:

Таблица 4.12.1 – Сетевые параметры устройств

Параметр	СПК1xx [M01]	ПК с OPC-сервером
IP-адрес	10.2.11.170	*любой адрес из сети, к которой подключен контроллер*
Порт	502	
Режим работы	slave	master

Переменные примера описаны в таблице ниже:

Таблица 4.12.2 – Список переменных примера

Имя	Тип	Область памяти Modbus	Адрес регистра/бита
Переменные Modbus Slave			
xVar_Modbus	BOOL	Coils	0/0
wVar_Modbus	WORD	Holding регистры	0
rVar_Modbus	REAL		1–2
sVar_Modbus	STRING(16)		3–10
Переменные программы для записи в Modbus Slave			
xVar_Prg	BOOL		
wVar_Prg	WORD		
rVar_Prg	REAL		
sVar_Prg	STRING(16)		

В рамках примера области **Coils** и **Holding** регистров являются независимыми (это определяется в настройках компонента **Modbus Serial Device**).

Для настройки обмена следует:

1. Подключить контроллер и ПК к общей локальной сети.
2. Создать новый проект **CODESYS** с программой **PLC_PRG** на языке **CFC**:

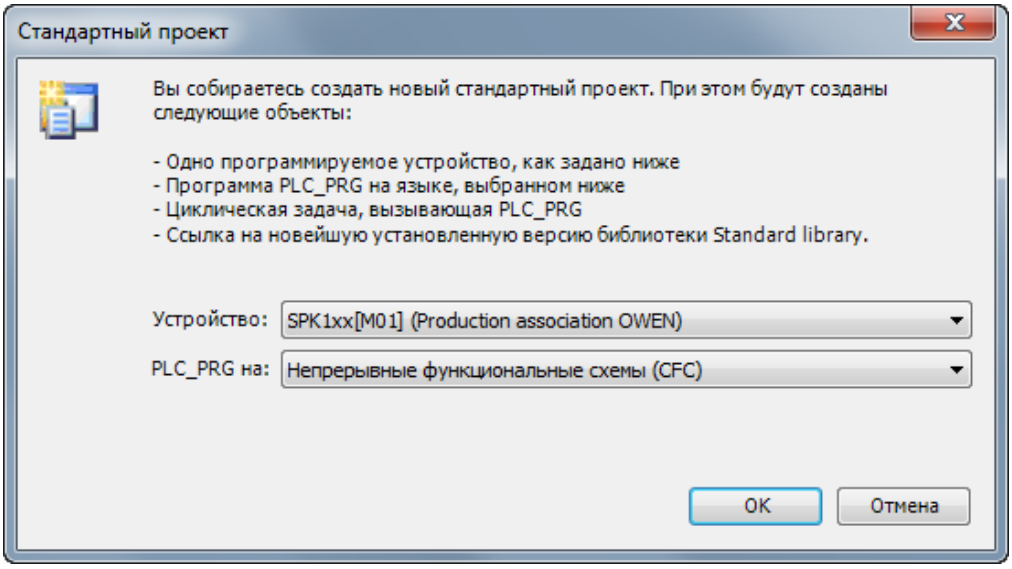


Рисунок 4.12.2 – Создание проекта CODESYS

3. Добавить в проект объединения с именами **Real_Word** и **String_Word:**

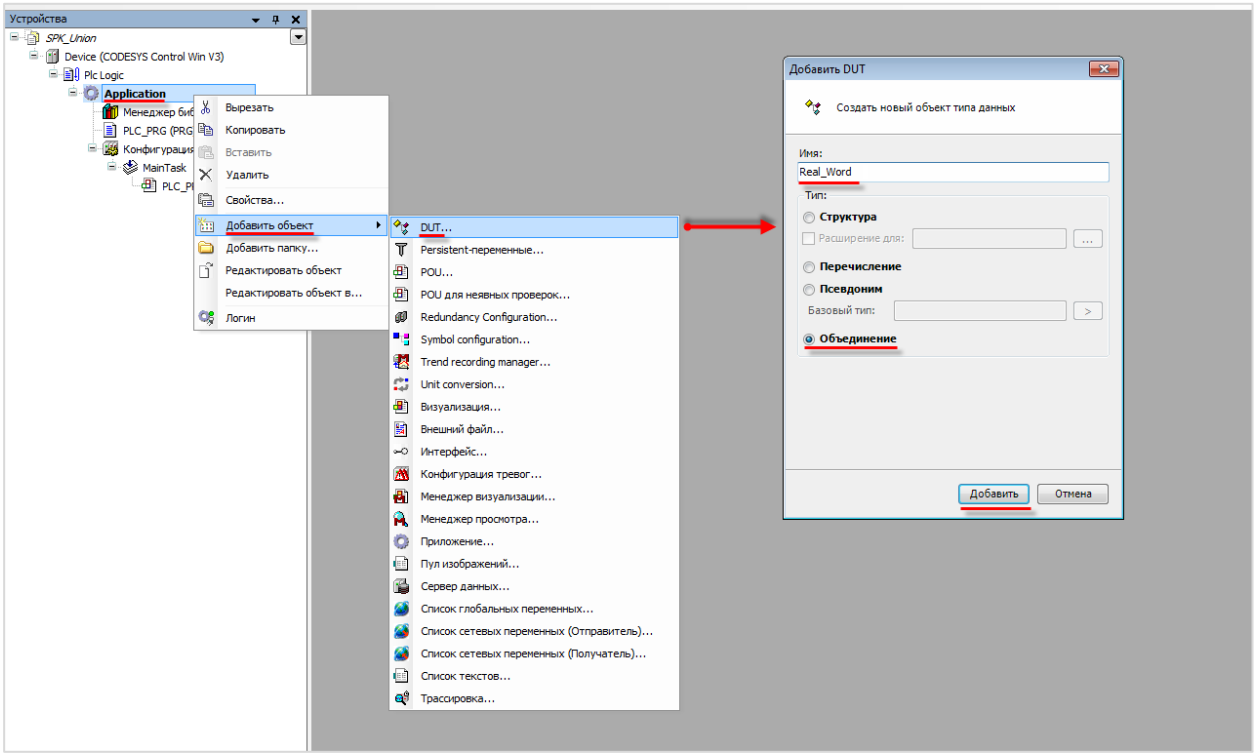


Рисунок 4.12.3 – Добавление в проект объединения Real_Word

Объединения потребуются для преобразования переменных типов **REAL** и **STRING** в набор переменных типа **WORD** для привязки к компоненту **Modbus TCP Slave Device**.

4. В объединениях объявить следующие переменные:

Real_Word

```
TYPE Real_Word :
UNION
    rRealValue      :REAL;
    awModbusReal    :ARRAY [0..1] OF WORD;
END_UNION
END_TYPE
```

Рисунок 4.12.4 – Объявление переменных объединения Real_Word

String_Word

```
TYPE String_Word :
UNION
    awModbusString: ARRAY [0..7] OF WORD;
    sStringValue:   STRING(16);
END_UNION
END_TYPE
```

Рисунок 4.12.5 – Объявление переменных объединения String_Word

5. В **менеджере библиотек** добавить библиотеку OwenCommunication.

6. В программе **PLC_PRG** объявить переменные в соответствии с таблицей 4.10.2.

PLC_PRG

```
PROGRAM PLC_PRG
VAR
    xVar_Modbus:            BOOL;
    wVar_Modbus:            WORD;
    rVar_Modbus:            REAL;
    sVar_Modbus:            STRING(15);
    uRealWord_Modbus:       Real_Word;
    uStringWord_Modbus:     String_Word;


    xWriteToSlave:          BOOL;
    fbWriteToSlave:         R_TRIG;

    xVar_Prg:               BOOL;
    wVar_Prg:               WORD;
    rVar_Prg:               REAL;
    sVar_Prg:               STRING(15);
    uRealWord_Prg:          Real_Word;
    uStringWord_Prg:        String_Word;
END_VAR
```

Рисунок 4.12.6 – Объявление переменных программы

Код программы будет выглядеть следующим образом:

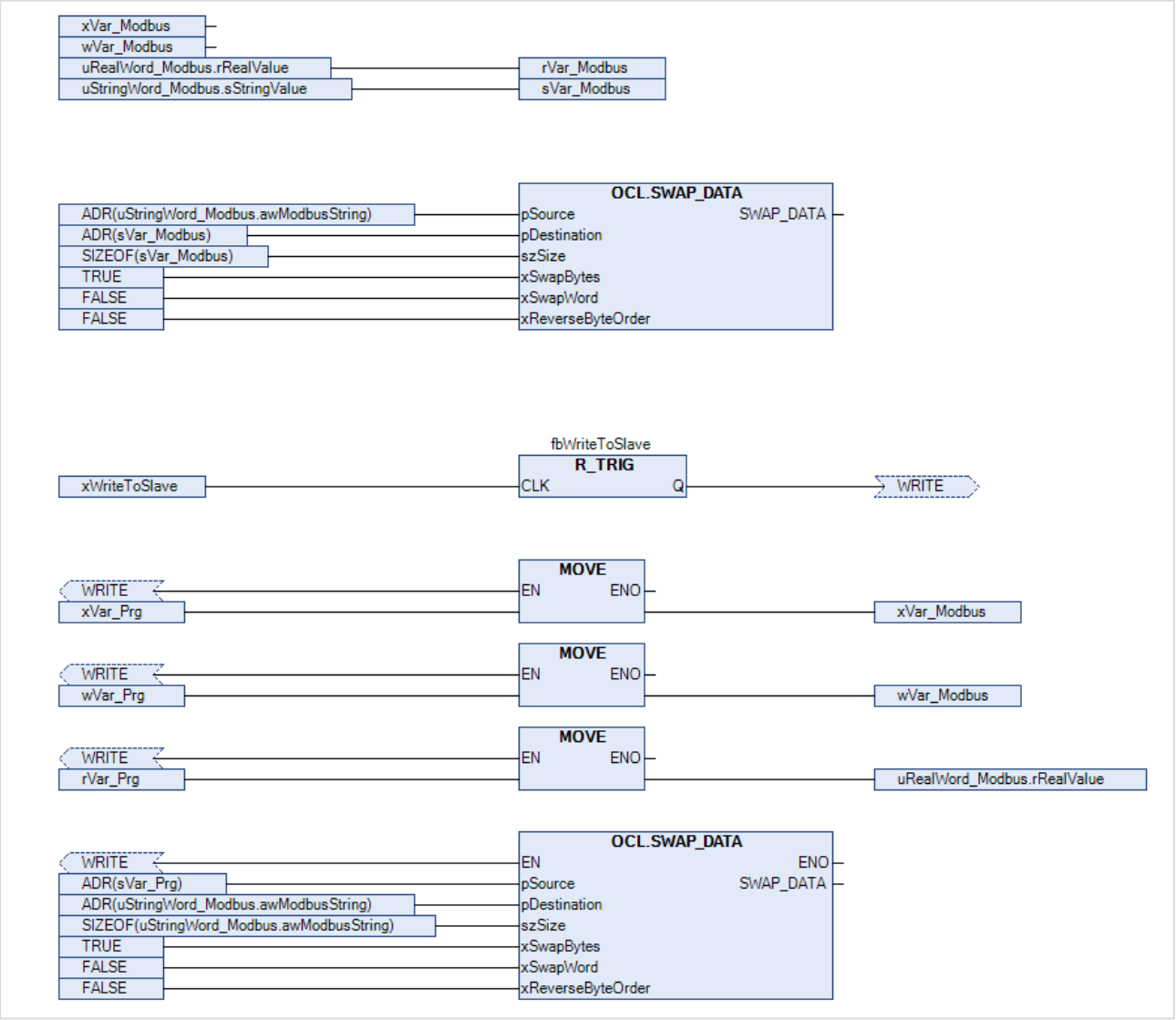


Рисунок 4.12.7 – Код программы PLC_PRG

Функция **SWAP_DATA** из библиотеки **OwenCommunication** используется для изменения порядка байтов в переменной типа **STRING** для соответствия порядку байтов в OPC-сервере (функционал перестановки байт в OPC-сервере не распространяется на тип STRING).

7. Добавить компоненты **Ethernet** и **Modbus TCP Slave Device** в соответствии с п. 4.4. Настроить компонент **Modbus TCP Slave Device** в соответствии с рисунком ниже:

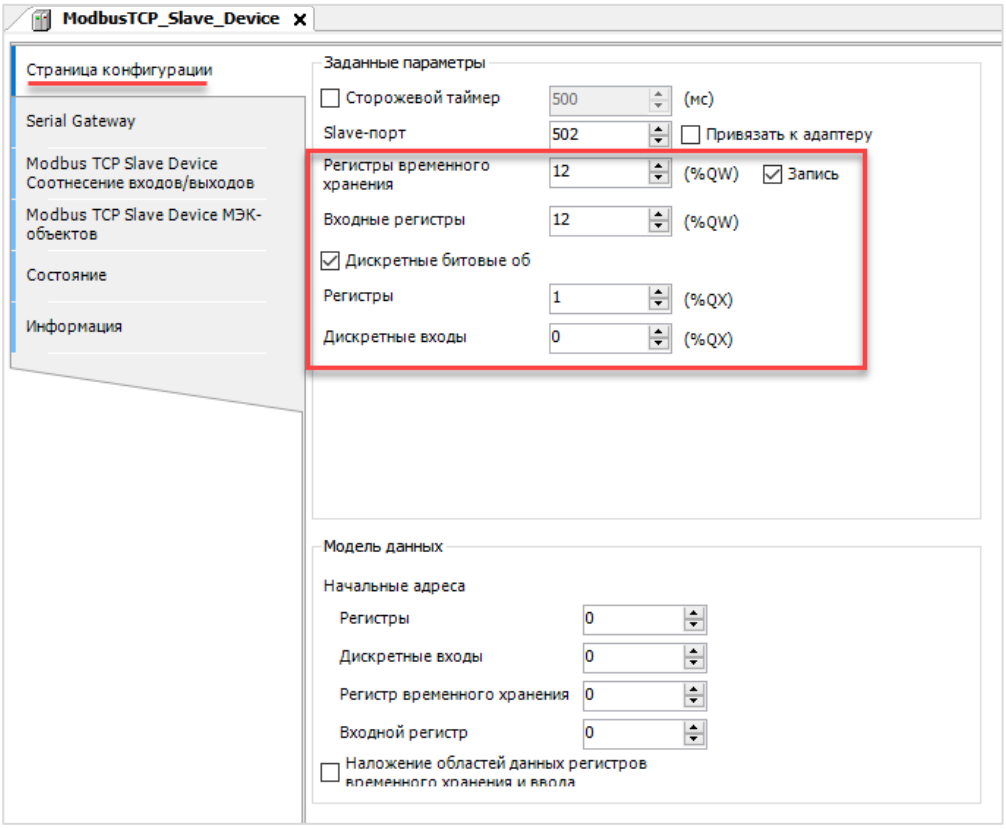


Рисунок 4.12.8 – Настройки компонента Modbus TCP Slave Device

8. Привязать к каналам компонента **Modbus TCP Slave Device** переменные программы в соответствии с таблицей 4.12.2. Установить галочку **Вкл. 2 (Всегда в задаче цикла шины)**.

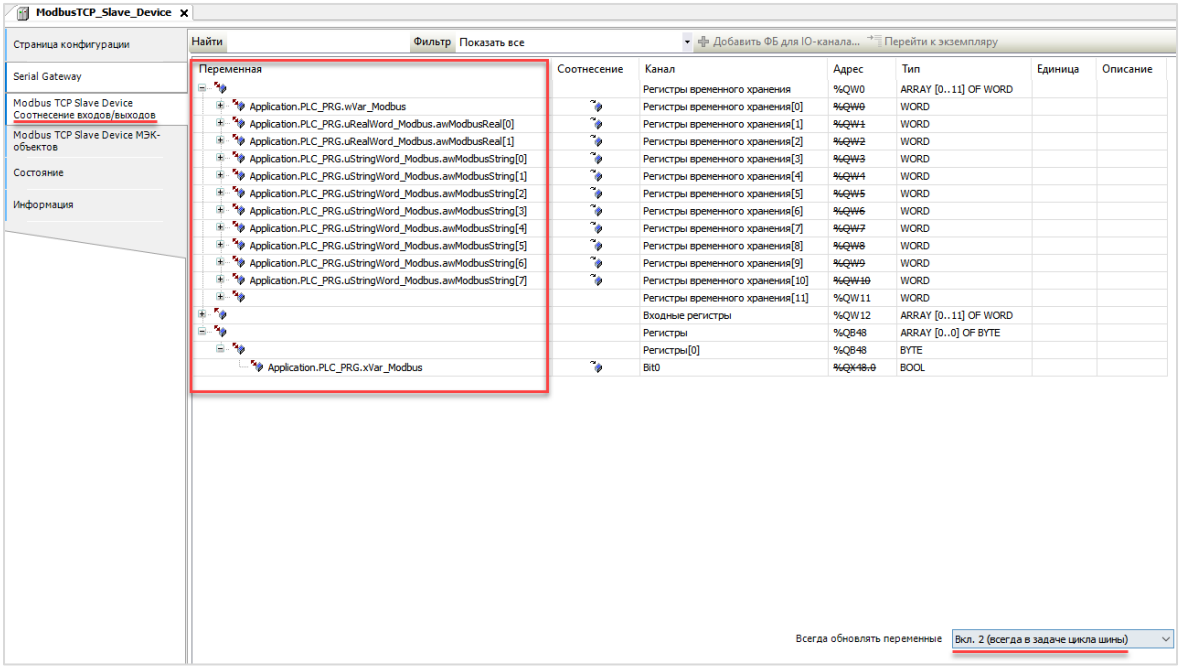


Рисунок 4.12.9 – Привязка переменных к компоненту Modbus TCP Slave Device

9. Установить и запустить MasterOPC Universal Modbus Server.

10. Нажать **ПКМ** на узел **Server** и добавить коммуникационный узел типа **TCP/IP**. В узле указать сетевые настройки в соответствии с таблицей 4.11.2. Для работы OPC-сервера в режиме **Modbus TCP Master** параметр **Slave подключение** должны иметь значение **FALSE**.

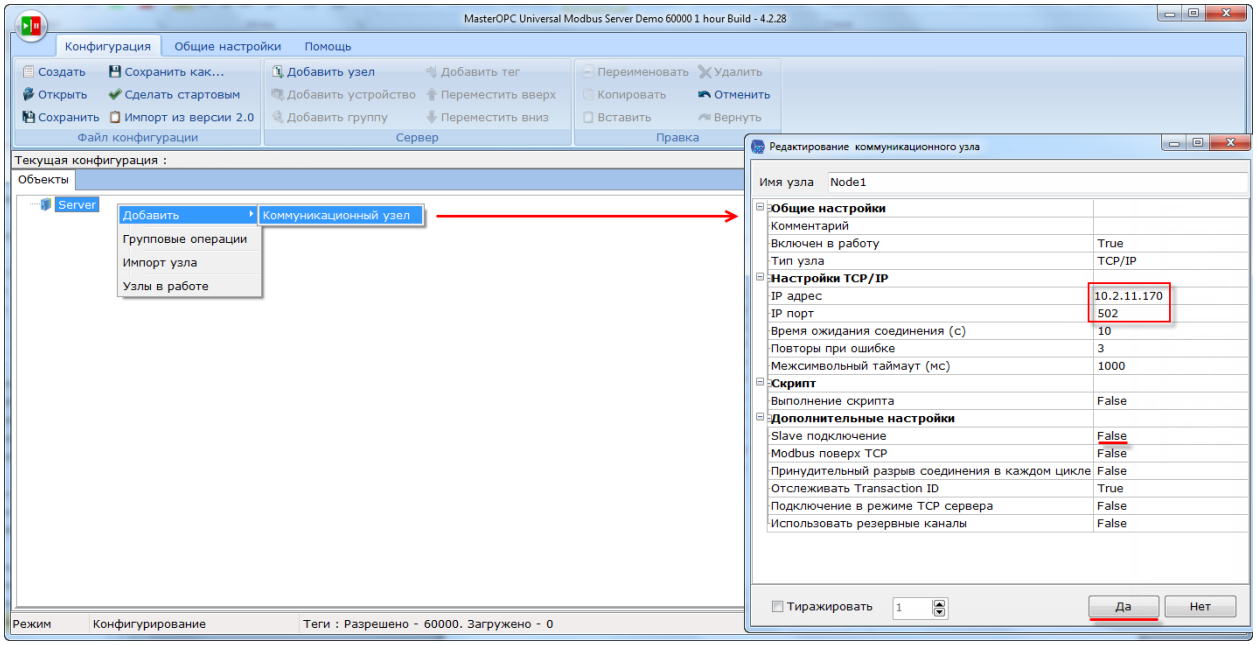


Рисунок 4.12.10 – Добавление коммуникационного узла

11. Нажать **ПКМ** на коммуникационный узел и добавить устройство с настройками по умолчанию.

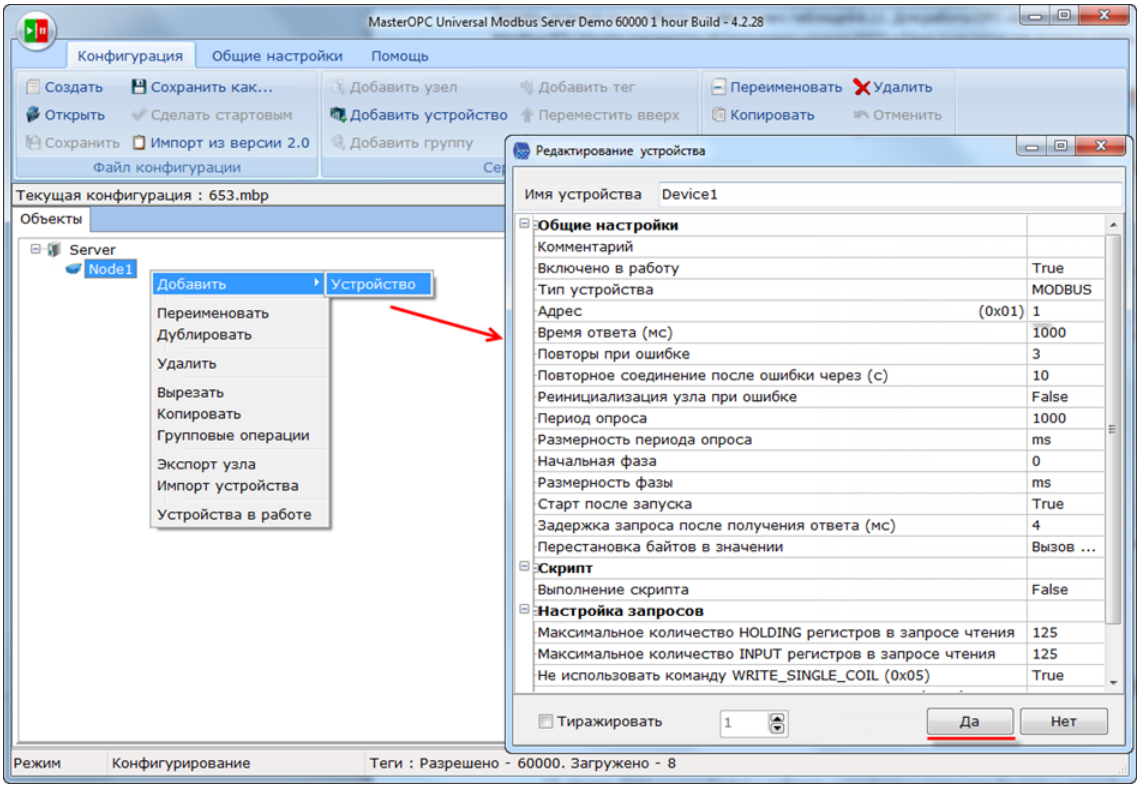


Рисунок 4.12.11 – Добавление устройства

12. Нажать **ПКМ** на устройство и добавить 4 тега. Число тегов соответствует числу переменных, считываемых/записываемых OPC-сервером. Настройки тегов приведены ниже.

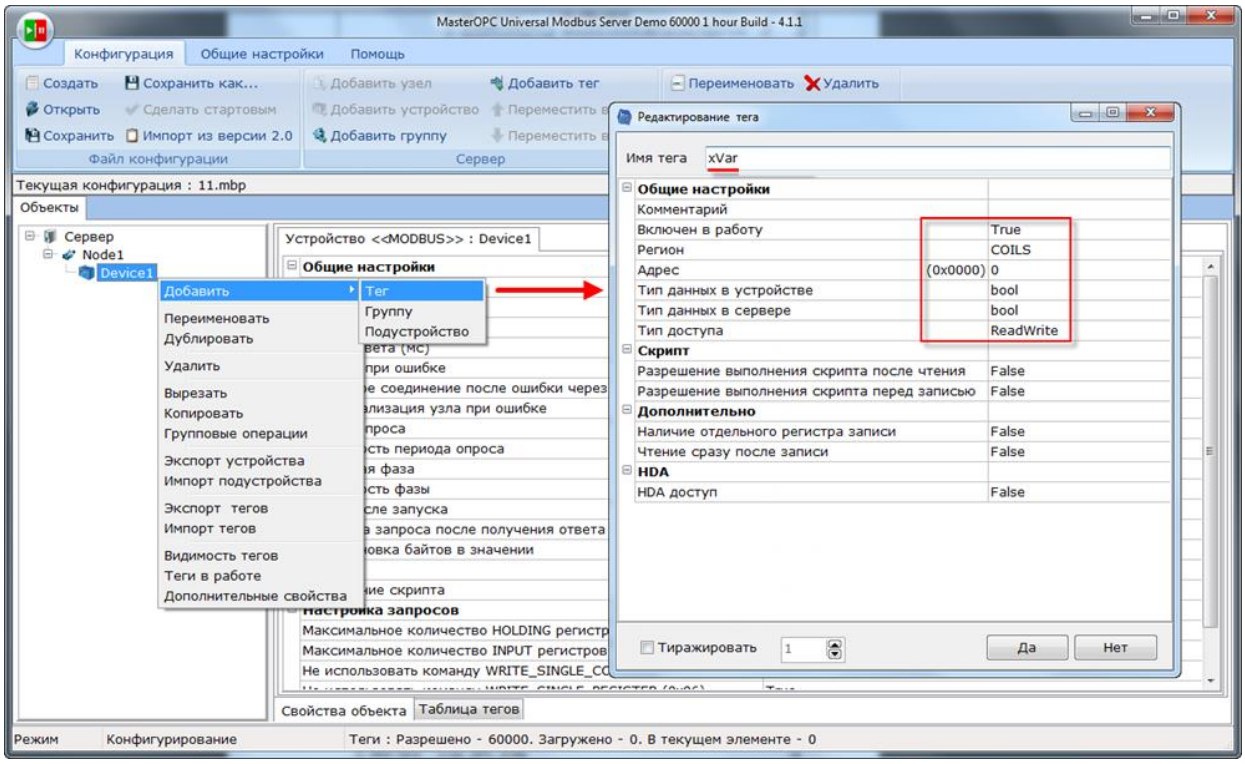


Рисунок 4.12.12 – Добавление тега xVar

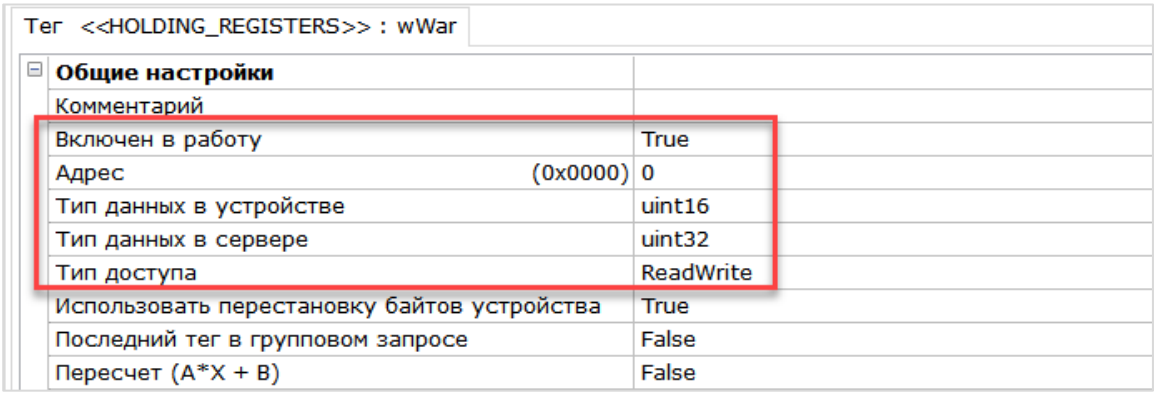


Рисунок 4.12.13 – Добавление тега wVar

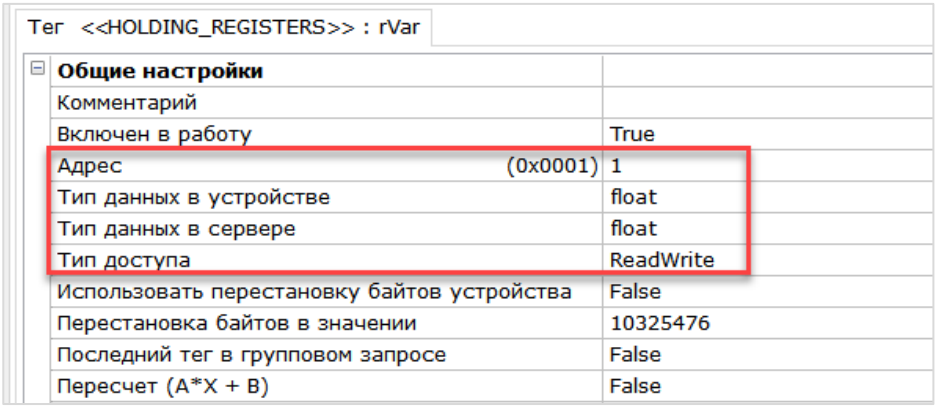


Рисунок 4.12.14 – Добавление тега rVar

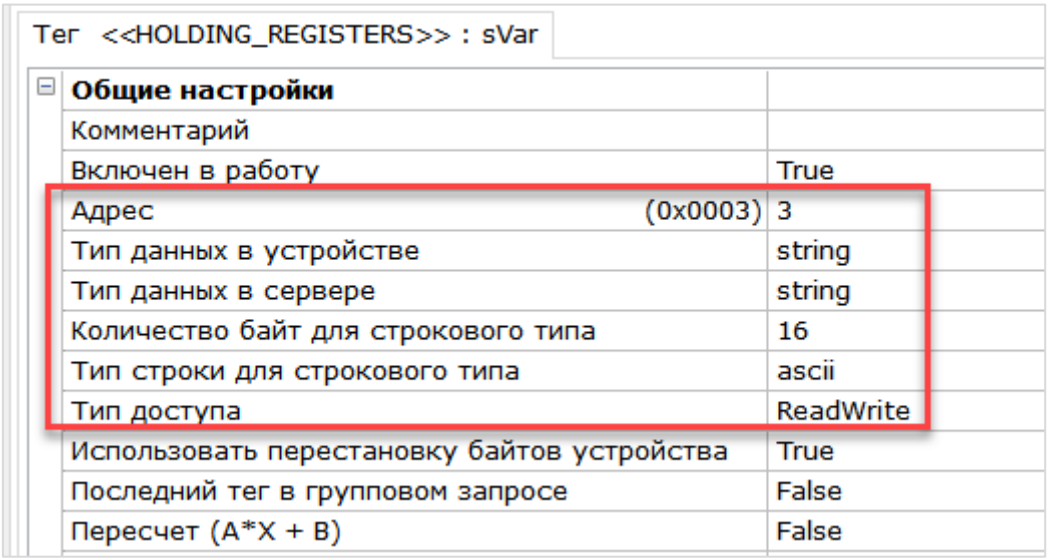


Рисунок 4.12.15 – Добавление тега sVar

13. Загрузить проект в контроллер и запустить его. Запустить OPC-сервер для контроля значений переменных.

В редакторе CODEYS следует изменить значения **_Prg** переменных, сгенерировать передний фронт в переменной **xWriteToSlave** и наблюдать соответствующие изменения в OPC-сервере. В OPC-сервере следует изменить значения переменных и наблюдать соответствующие значения в **_Modbus** переменных CODESYS.

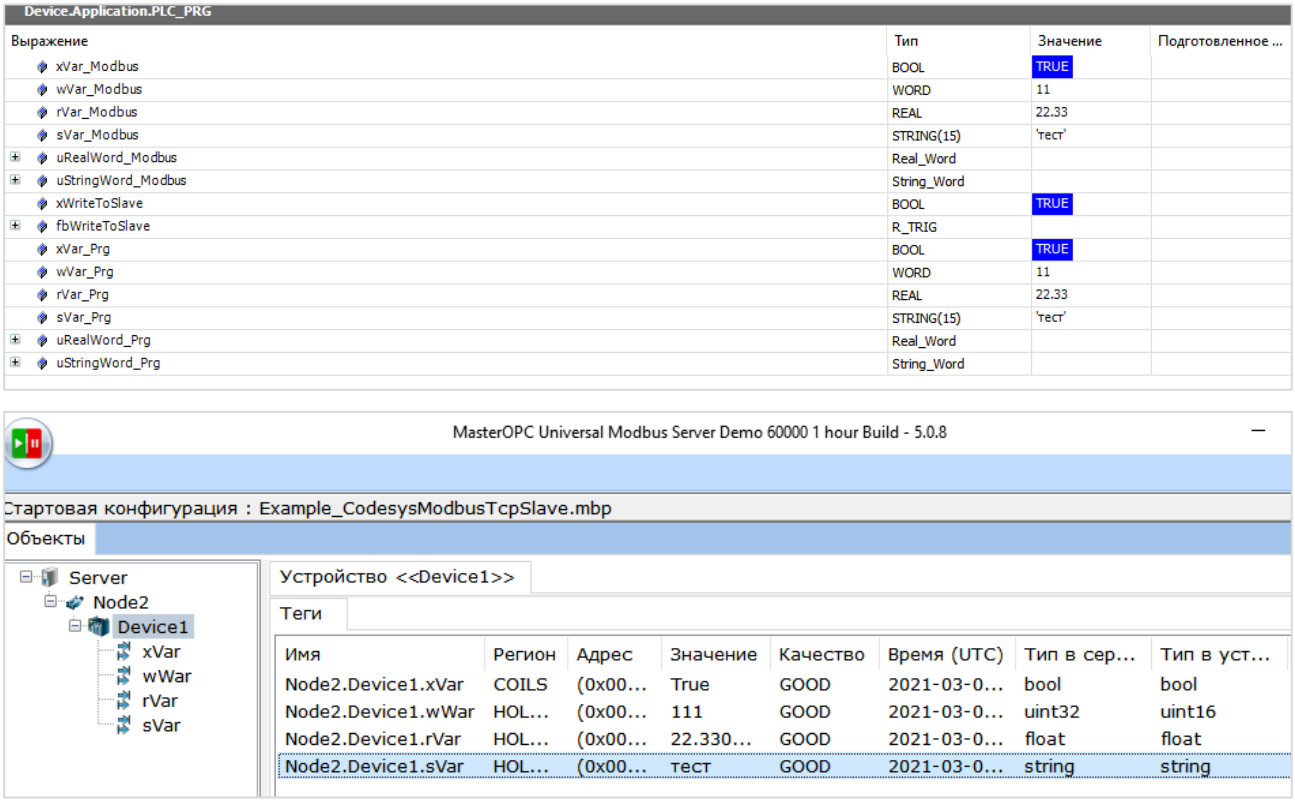


Рисунок 4.12.16 – Чтение и запись данных через OPC-сервер

4.13 Использование шлюзов Modbus RTU/Modbus TCP

Если контроллер используется в режиме **Modbus TCP Slave** – то он может работать в роли шлюза **Modbus TCP/Modbus RTU**. Для этого следует в настройках компонента Modbus TCP Slave Device на вкладке **Serial Gateway** установить галочку **Serial Gateway Active** и указать идентификатор COM-порта и скорость обмена. Запросы от **Modbus TCP Master**, опрашивающего контроллер, будут преобразованы в запросы **Modbus RTU** и отправлены в выбранный на вкладке COM-порт на заданной скорости (этот COM-порт не должен использоваться в других компонентах проекта). Поддерживается только режим настроек **8-N-1**. Ответы от **Modbus RTU Slave-устройств**, подключенных к COM-порту, будут преобразованы в **Modbus TCP** и отправлены обратно master-устройству. В запросе master-устройства должен быть указан значимый **Unit ID** (в диапазоне **1..247**).

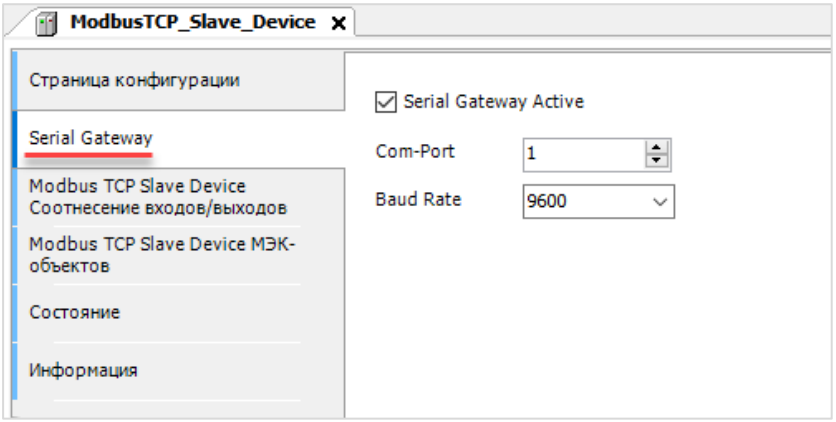


Рисунок 4.13.1 – Настройка шлюза Modbus TCP/Modbus RTU в компоненте Modbus TCP Slave Device

Если контроллер используется в режиме **Modbus TCP Master** – то он может работать совместно с преобразователями **Modbus TCP/Modbus RTU**. В этом случае рекомендуется добавить один компонент **Modbus TCP Slave** (который будет соответствовать преобразователю), а в качестве его дочерних компонентов - нужное количество компонентов **Modbus Slave COM Port** (которые будут соответствовать slave-устройствам шины Modbus RTU). В этом случае контроллер установит с конвертером протоколов только одно соединение – это важно, так как часто такие преобразователи поддерживают лишь несколько одновременных TCP-соединений. В текущих версиях CODESYS (включая **3.5.16.x**) не поддерживается обработка ошибок связи с RTU-устройствами, расположенными за шлюзом – при возникновении первой ошибки обмена опрос устройства прекращается. См. рекомендации по обходу этой проблемы.

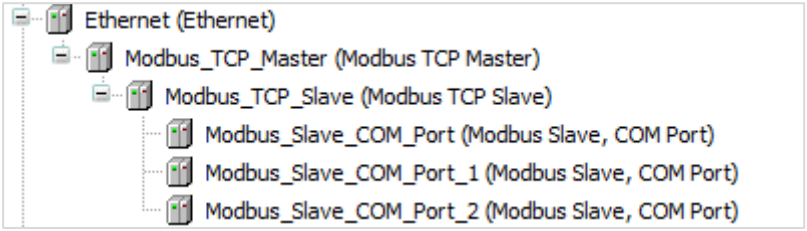


Рисунок 4.13.2 – Настройка соединения со шлюзом Modbus TCP/Modbus RTU

5 Библиотека OwenCommunication

5.1 Основная информация

Библиотека **OwenCommunication** используется для реализации обмена в программе пользователя. Библиотека лишена ограничений стандартных средств конфигурирования и включает дополнительный функционал. Библиотека содержит:

- ФБ для настройки интерфейсов;
- ФБ обмена по Modbus;
- ФБ для реализации нестандартных протоколов;
- функции и ФБ преобразования данных.

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

ПРИМЕЧАНИЕ
Библиотека соответствует PLCopen Behavior Model.

5.2 Установка библиотеки

Библиотека **OwenCommunication** доступна на сайте компании ОВЕН в разделе CODESYS V3/Библиотеки и компоненты. Для установки библиотеки в **CODESYS** в меню **Инструменты** следует выбрать пункт **Репозиторий библиотек**, после чего нажать **Установить** и указать путь к файлу библиотеки:

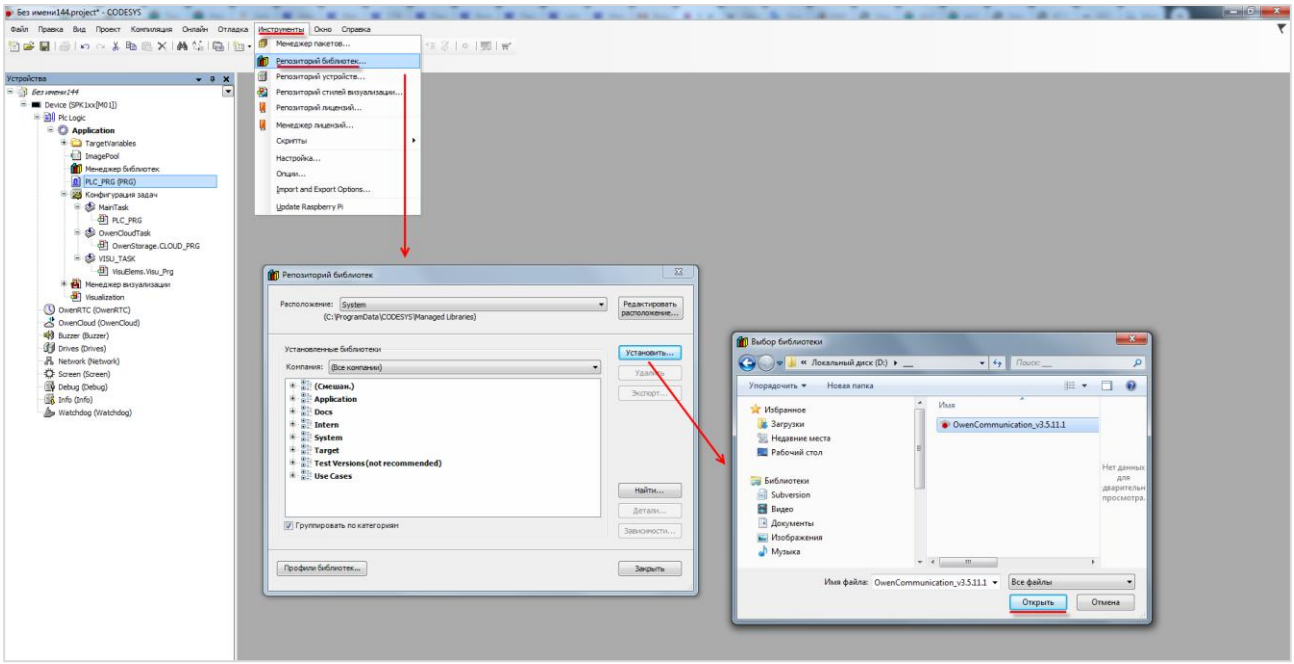


Рисунок 5.2.1 – Установка библиотеки

5.3 Добавление библиотеки в проект CODESYS

Для добавления библиотеки **OwenCommunication** в проект **CODESYS** в **Менеджере библиотек** следует нажать кнопку **Добавить библиотеку**, в появившемся списке выбрать библиотеку **OwenCommunication** и нажать **ОК**.

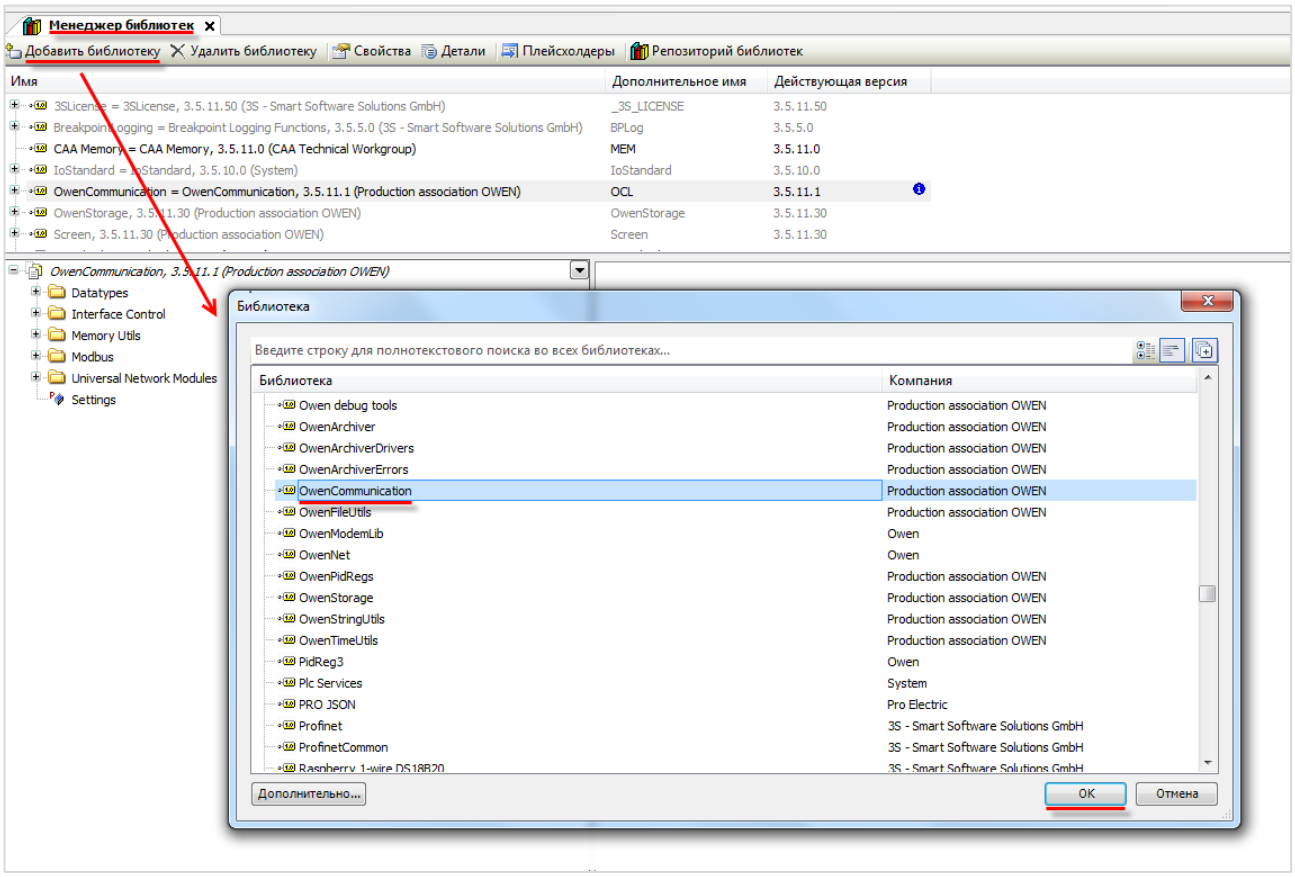


Рисунок 5.3.1 – Добавление библиотеки OwenCommunication

После добавления библиотека появится в списке **Менеджера библиотек**:

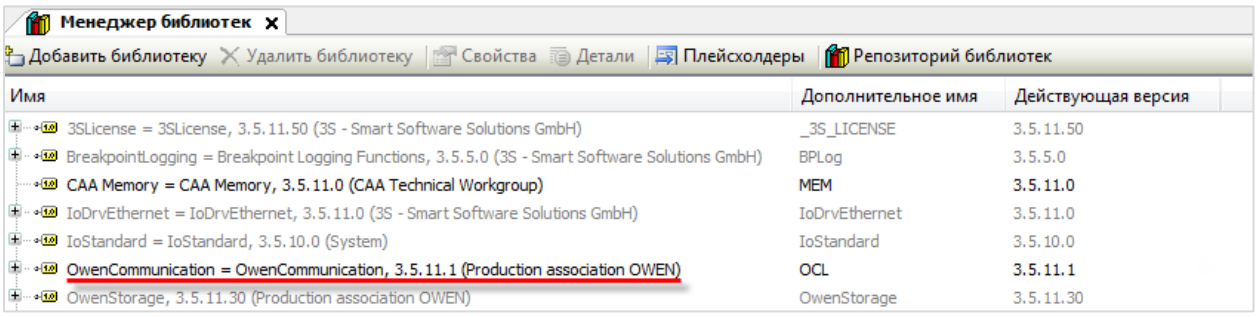


Рисунок 5.3.2 – Список библиотек проекта

ПРИМЕЧАНИЕ

При обращении к объектам библиотеки следует перед их названием указывать префикс **OCL** (пример: **OCL.COM_Control**).

5.4 Структуры и перечисления

5.4.1 Перечисление ERROR

Перечисление **ERROR** содержит ошибки, которые могут возникнуть при вызове ФБ библиотеки.

Таблица 5.4.1 – Описание элементов перечисления ERROR

Название	Значение	Описание
Общие ошибки		
NO_ERROR	0	Нет ошибок
TIME_OUT	1	Ошибка таймаута
HANDLE_INVALID	10	Некорректное значение дескриптора интерфейса
ERROR_UNKNOWN	11	Неизвестная ошибка (зарезервировано для будущих версий)
WRONG_PARAMETER	12	Как минимум один из аргументов ФБ имеет некорректное значение
WRITE_INCOMPLETE	13	Отправка сообщения не была завершена
INVALID_DATAPOINTER	20	Некорректный указатель на буфер данных
INVALID_DATASIZE	21	Некорректный размер буфера данных
INVALID_ADDR	22	Некорректное значение в поле IP-адреса
Ошибки UDP/TCP		
UDP_RECEIVE_ERROR	30	Ошибка получения UDP-запроса
UDP_SEND_ERROR	31	Ошибка отправки UDP-запроса
UDP_SEND_NOT_COMPLETE	32	Отправка UDP-запроса не была завершена (зарезервировано для будущих версий)
UDP_OPEN_ERROR	33	Ошибка создания UDP-сокета
UDP_CLOSE_ERROR	34	Ошибка закрытия UDP-сокета
TCP_SEND_ERROR	40	Ошибка отправки TCP-запроса
TCP_RECEIVE_ERROR	41	Ошибка получения TCP-запроса
TCP_OPEN_ERROR	42	Ошибка создания TCP-сокета
TCP_CONNECT_ERROR	43	Ошибка при установке TCP-соединения
TCP_CLOSE_ERROR	44	Ошибка при закрытии TCP-соединения
TCP_SERVER_ERROR	45	Ошибка TCP-сервера (зарезервировано для будущих версий)
TCP_NO_CONNECTION	46	TCP-соединение отсутствует
IOCTL_ERROR	47	Внутренняя ошибка при использовании системных вызовов
Ошибки Modbus		
ILLEGAL_FUNCTION	50	Данная функция Modbus не поддерживается slave-устройством
ILLEGAL_DATA_ADDRESS	51	Как минимум один из регистров, указанных в запросе, отсутствует в slave-устройстве
ILLEGAL_DATA_VALUE	52	Некорректное значение в поле данных
SLAVE_DEVICE_FAILURE	53	Slave-устройство не может обработать данный запрос
Специфические ошибки		
RESPONSE_CRC_FAIL	60	Рассчитанная CRC не соответствует CRC посылки
NOT_OWEN_DEVICE	61	Данное устройство не является контроллером ОВЕН

5.4.2 Перечисление COM_PARITY

Перечисление **COM_PARITY** описывает режим контроля четности COM-порта.

Таблица 5.4.2 – Описание элементов перечисления COM_PARITY

Название	Значение	Описание
EVEN	0	Проверка на четность
ODD	1	Проверка на нечетность
NONE	2	Проверка отсутствует

5.4.3 Перечисление COM_STOPBIT

Перечисление **COM_STOPBIT** описывает возможное число стоп-битов при обмене через COM-порт.

Таблица 5.4.3 – Описание элементов перечисления COM_STOPBIT

Название	Значение	Описание
ONE	0	Один стоп-бит
ONE_HALF	1	Полтора стоп-бита (*не поддерживается для Linux*)
TWO	2	Два стоп-бита

5.4.4 Перечисление MB_FC

Перечисление **MB_FC** описывает используемую функцию Modbus.

Таблица 5.4.4 – Описание элементов перечисления MB_FC

Название	Значение	Описание
READ_COILS	16#01	Чтение значений из регистров флагов
READ_DISCRETE_INPUTS	16#02	Чтение значений из дискретных входов
READ_HOLDING_REGISTERS	16#03	Чтение значений из регистров хранения
READ_INPUT_REGISTERS	16#04	Чтение значений из регистров ввода
WRITE_SINGLE_COIL	16#05	Запись значения в один регистр флага
WRITE_SINGLE_REGISTER	16#06	Запись значения в один регистр хранения
WRITE_MULTIPLE_COILS	16#0F	Запись значений в несколько регистров флагов
WRITE_MULTIPLE_REGISTERS	16#10	Запись значений в несколько регистров хранения

5.4.5 Структура MB_REQ_INFO

Структура **MB_REQ_INFO** описывает запрос Modbus, полученный slave-устройством. Структура используется в ФБ MB_SerialSlave и MB_TcpSlave для предоставления пользователю информации о запросах, поступающих от master-устройства, а также для запрета обработки определенных запросов.

Таблица 5.4.5 – Описание элементов структуры MB_REQ_INFO

Название	Тип	Описание
usiSlaveId	USINT	Адрес slave-устройства, указанный в запросе
eFuncCode	MB_FC	Код функции Modbus, указанный в запросе
uiDataAddr	UINT	Начальный адрес регистра, указанный в запросе
uiDataCount	UINT	Количество считываемых или записываемых битов/регистров, указанное в запросе
xForbidden	BOOL	**TRUE** – запрос считается запрещенным для данного экземпляра ФБ, **FALSE** – запрос считается разрешенным
eError	ERROR	Код ошибки (только для ФБ MB_SerialSlave в режиме Spy). Возможные коды ошибок: NO_ERROR, ILLEGAL_FUNCTION, ILLEGAL_DATA_ADDRESS, ILLEGAL_DATA_VALUE, SLAVE_DEVICE_FAILURE

Структура используется в следующих случаях:

1. При получении запроса от master-устройства выход **xNewRequest** принимает значение **TRUE** на один цикл контроллера и выход **stRequestInfo** (типа **MB_REQ_INFO**) в данном цикле содержит информацию об этом запросе.

2. Для запрета обработки определенных запросов от master-устройства. В этом случае:

Если блок MB_SerialSlave с **usiSlaveId** = **255** (адрес «режима отладки»; по умолчанию в этом режиме блок отвечает на запросы с любым Slave ID, эмулируя все устройства шины) получает запрос от master-устройства на адрес, который присутствует в списке запрещенных запросов **pastForbiddenRequests** (т. е. поле **MB_REQ_INFO.usiSlaveId** <> **0**), то запрос с указанным **usiSlaveId** игнорируется. Это позволяет эмулировать с помощью блока определенный набор slave-устройств.

Если у блока MB_SerialSlave вход **usiSlaveId** имеет значение, отличное от **255**, то в списке запрещенных запросов **pastForbiddenRequests** поле **MB_REQ_INFO.usiSlaveId** должно иметь значение **0**. Для ФБ MB_TcpSlave поле **MB_REQ_INFO.usiSlaveId** не используется.

Если блок MB_SerialSlave или MB_TcpSlave получает запрос с кодом функции **eFuncCode**, в котором хотя бы один из считываемых или записываемых битов/регистров попадает в диапазон **[uiDataAddr…uiDataAddr+uiDataCount – 1]**, то этот запрос игнорируется, а master-устройству в ответ отправляется сообщение с кодом ошибки **02 (ILLEGAL_DATA_ADDRESS)**.

Если **uiDataAddr = 16#FFFF**, то блок игнорирует все запросы с кодом функции **eFuncCode**. В ответ master-устройству будет отправлено сообщение с кодом ошибки **01 (ILLEGAL_FUNCTION)**. Это, например, может использоваться для создания slave-устройства, все регистры которого доступны только для чтения.

Изменение значений полей структуры сразу влияет на работу блока (то есть воздействие на вход **xEnable** для применения новых значений не требуется).

3. В блоке MB_SerialSlave в режиме Spy (**xSpyMode = TRUE**) структура используется для отображения данных прослушанных запросов (см. выход **stRequestInfo**).

5.4.6 Структура MB_SUBREQUEST_PARAMS

Структура **MB_SUBREQUEST_PARAMS** описывает параметры запроса для функции **20 (0x14) Read File Record**. Структура используется в ФБ MB_SerialReadFile.

Таблица 5.4.6 – Описание элементов структуры MB_SUBREQUEST_PARAMS

Название	Тип	Описание
uiFileNumber	UINT	Номер считываемого файла
uiRecordNumber	UINT	Номер считываемой записи
usiRecordLength	USINT(1..124)	Длина считываемой записи в регистрах

5.5 ФБ настройки интерфейсов

5.5.1 ФБ COM_Control

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

ПРИМЕЧАНИЕ
Одновременно (в пределах одного цикла контроллера) могут вызываться не более 20 экземпляров асинхронно выполняемых блоков (см. п. 6.5.3).

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

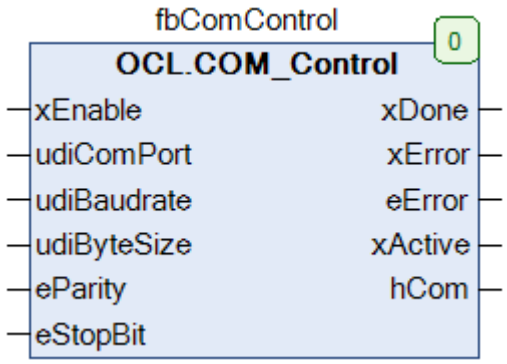


Рисунок 5.5.1 – Внешний вид ФБ COM_Control на языке CFC

Таблица 5.5.1 – 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	COM_PARITY	Режим контроля четности
eStopBit	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	ERROR	Статус работы ФБ (или код ошибки)
xActive	BOOL	Пока порт открыт, данный выход имеет значение **TRUE**
hCom	CAA.HANDLE	Дескриптор COM-порта

5.5.2 ФБ TCP_Client

Функциональный блок **TCP_Client** используется для открытия и закрытия TCP-соединения.

ПРИМЕЧАНИЕ
Одновременно (в пределах одного цикла контроллера) могут вызываться не более 20 экземпляров асинхронно выполняемых блоков (см. п. 6.5.3).

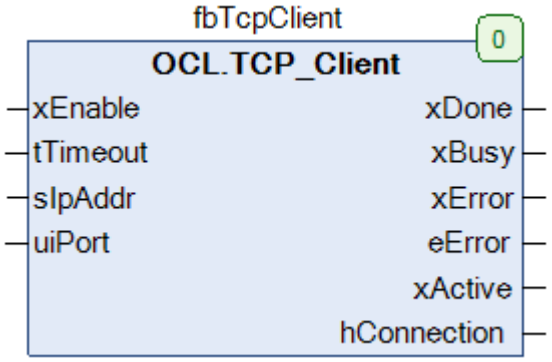


Рисунок 5.5.2 – Внешний вид ФБ TCP_Client на языке CFC

Таблица 5.5.2 – Описание входов и выходов ФБ TCP_Client

Название	Тип	Описание
Входы		
xEnable	BOOL	По переднему фронту происходит открытие TCP-соединения, по заднему – закрытие
tTimeOut	TIME	Таймаут установки соединения (0 – время ожидания не ограничено)
sIpAddr	STRING	IP-адрес сервера в формате IPv4 (‘xxx.xxx.xxx.xxx’)
uiPort	UINT	Порт сервера
Выходы		
xDone	BOOL	**TRUE** – сервер закрыл соединение. Для повторного соединения требуется создать передний фронт на входе **xEnable**
xBusy	BOOL	**TRUE** – выполняется установка соединения
xError	BOOL	Принимает значение **TRUE** в случае возникновения ошибки
eError	ERROR	Статус работы ФБ (или код ошибки)
xActive	BOOL	Пока соединение активно, данный выход имеет значение **TRUE**
hConnection	CAA.HANDLE	Дескриптор соединения

5.6 ФБ протокола Modbus

5.6.1 ФБ MB_SerialRequest

Функциональный блок **MB_SerialRequest** используется для работы в режиме **Modbus Serial Master**. По переднему фронту на входе **xExecute** происходит отправка запроса, определяемого параметрами **usiSlaveId**, **eFuncCode**, **uiDataAddr** и **uiDataCount** по протоколу **Modbus RTU** (если **xIsAsciiMode** = **FALSE**) или **Modbus ASCII** (если **xIsAsciiMode** = **TRUE**) через COM-порт, определяемый дескриптором **hCom**, полученным от ФБ COM_Control. Считываемые или записываемые данные размещаются в буфере, расположенном по указателю **pData** размером **szSize** байт.

Ответ от slave-устройства ожидается в течение времени **tTimeout**. 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**. В случае получения корректного ответа выход **xDone** принимает значение **TRUE**, а выход **eError** = **NO_ERROR**. В случае получения ответа с кодом ошибки Modbus **xError** принимает значение **TRUE**, а выход **eError** содержит код ошибки (при этом выход **xDone** не принимает значение **TRUE**). Для отправки следующего запроса следует создать передний фронт на входе **xExecute**.

ПРИМЕЧАНИЕ
Одновременно (в пределах одного цикла контроллера) могут вызываться не более 20 экземпляров асинхронно выполняемых блоков (см. п. 6.5.3).

ПРИМЕЧАНИЕ
В случае использования широковещательной рассылки (на адрес **0**) рекомендуется для параметра **tTimeout** установить значение **T#1ms**, так как получение ответа в данном случае не подразумевается.

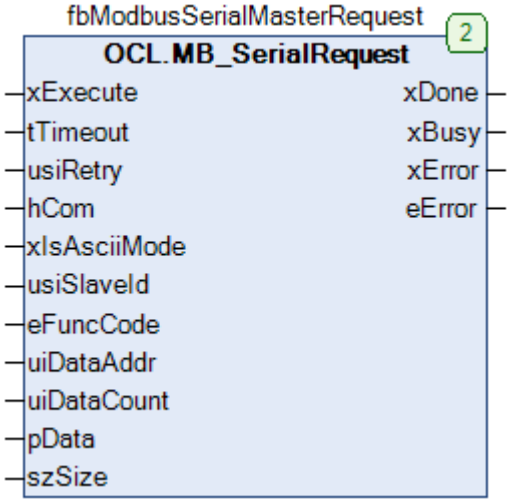


Рисунок 5.6.1 – Внешний вид ФБ MB_SerialRequest на языке CFC

Таблица 5.6.1 – Описание входов и выходов ФБ MB_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
xIsAsciiMode	BOOL	Используемый протокол: **FALSE** – Modbus RTU, **TRUE** – Modbus ASCII
usiSlaveId	USINT	Адрес slave-устройства (**0** – широковещательная рассылка)
eFuncCode	MB_FC	Используемая функция Modbus
uiDataAddr	UINT	Начальный адрес бита/регистра в запросе
uiDataCount	UINT	Число битов/регистров в запросе
pData	CAA.PVOID	Указатель на буфер записываемых или считываемых данных. **Примечание**: для функции **WRITE_SINGLE_COIL** следует использовать указатель на переменную типу **WORD** с возможными значениям **0** и **1**. Во всех остальных битовых функциях используется битовая маска
szSize	CAA.SIZE	Размер буфера в байтах
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	ERROR	Статус работы ФБ (или код ошибки)

5.6.2 ФБ MB_SerialSlave

ПРИМЕЧАНИЕ
Одновременно (в пределах одного цикла контроллера) могут вызываться не более 20 экземпляров асинхронно выполняемых блоков (см. п. 6.5.3).

Функциональный блок **MB_SerialSlave** используется для работы в режиме **Modbus Serial Slave**. Пока вход **xEnable** имеет значение **TRUE**, блок находится в работе. На вход **hCom** следует подать дескриптор используемого COM-порта, полученный с помощью ФБ COM_Control. Вход **usiSlaveId** определяет адрес slave'а. Под регистры slave'а выделяется область памяти по указателю **pData** размером **szSize** байт.

Блок поддерживает протоколы Modbus RTU и Modbus ASCII. Протокол запроса определяется автоматически, и ответ отправляется в том же формате.

Блок поддерживает все стандартные функции Modbus, приведенные в таблице 2.2.

В реализации блока все области памяти Modbus наложены друг на друга и имеют общую адресацию (**MODBUS Data Model with only 1 block** согласно спецификации Modbus). Максимальный размер буфера slave'а – **65536** регистров.

Блок поддерживает получение широковещательных запросов (отправленных на адрес **0**).

При получении запроса от master-устройства выход **xNewRequest** на один цикл контроллера принимает значение **TRUE**, при этом выход **stRequestInfo** содержит информацию о полученном запросе.

Блок позволяет запретить обработку определенных запросов, полученных от master-устройства. Для этого на вход **pastForbiddenRequest** передается указатель на структуру (или массив структур) типа MB_REQ_INFO, а на входе **szForbiddenRequests** указывается размер этой структуры или массива в байтах. Каждый экземпляр структуры описывает один (или несколько – при использовании специальных заполнителей, см. п. 5.4.5) запрещенный запрос. В случае запрещенного запроса slave отправляет master-устройству ответ с кодом ошибки (см. подробнее в п. 5.4.5). В случае получения запрещенного запроса с функцией записи полученные значения игнорируются.

Блок поддерживает работу в режиме Spy (прослушивание «чужих» запросов). Для включения режима используется вход **xSpyMode.** Пока вход имеет значение **TRUE** – блок прослушивает все корректные запросы от master-устройств и все корректные ответы от slave-устройств на шине для функций **0x03, 0x04, 0x06, 0x10**. Под «корректным» понимается пакет, который соответствует спецификации Modbus (в том числе ответ с сообщением об ошибке). При получении пакета на один цикл активируется выход **xNewRequest** (для запроса) или **xNewResponse** (для ответа). Выходы могут быть активированы одновременно, если пауза между запросом и ответом меньше цикла ПЛК. Параметры запроса (адрес, код функции и т. д.) можно определить с помощью выхода **stRequestInfo**. Данные запроса размещаются в буфере по указателю **pSpyData**, который имеет размер **szSpyData** байт.

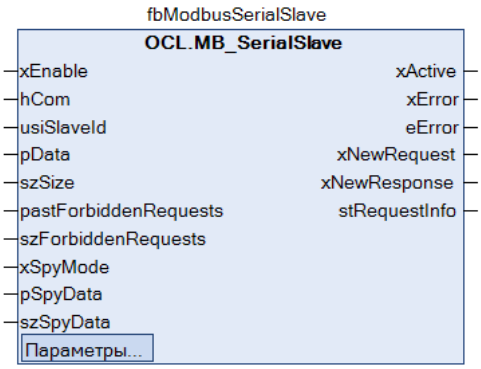


Рисунок 5.6.2 – Внешний вид ФБ MB_SerialSlave на языке CFC

Таблица 5.6.2 – Описание входов и выходов ФБ MB_SerialSlave

Название	Тип	Описание
Входы		
xEnable	BOOL	По переднему фронту происходит включение slave'а, по заднему – отключение
hCom	CAA.HANDLE	Дескриптор COM-порта, полученный от ФБ COM_Control
usiSlaveId	USINT	Адрес slave-устройства. Если указан адрес **255**, то slave отвечает на запросы с любым адресом, кроме запрещенных (см. п. 5.4.5) – это может быть удобным при отладке для имитации нескольких slave-устройств
pData	CAA.PVOID	Указатель на буфер данных slave'а
szSize	CAA.SIZE	Размер буфера в байтах
pastForbiddenRequest	CAA.PVOID	Указатель на массив структур запрещенных запросов
szForbiddenRequests	CAA.SIZE	Размер массива структур запрещенных запросов в байтах
xSpyMode	BOOL	**TRUE** – включен режим Spy (см. информацию в описании ФБ)
pSpyData	CAA.PVOID	Указатель на буфер данных прослушиваемых запросов
szSpyData	CAA.SIZE	Размер буфера данных прослушиваемых запросов в байтах
Выходы		
xActive	BOOL	**TRUE** – ФБ находится в работе
xError	BOOL	Принимает значение **TRUE** в случае возникновения ошибки
eError	ERROR	Статус работы ФБ (или код ошибки)
xNewRequest	BOOL	Принимает **TRUE** на один цикл контроллера при получении запроса от master-устройства
xNewResponse	BOOL	*Только для режима Spy* Принимает **TRUE** на один цикл контроллера при детектировании на шине ответа от slave-устройства
stRequestInfo	MB_REQ_INFO	Информация о полученном запросе (актуальна, пока **xNewRequest** или **xNewResponse** = TRUE). Поле **stRequestInfo.uiDataAddr** невалидно (имеет значение 0) при детектировании ответа от slave-устройства (в режиме Spy) для функций **0x03** и **0x04**
Параметры		
c_xReverseByteOrder	BOOL	**TRUE** – изменить порядок байт в буфере данных slave'а на противоположный
c_uiStartAddr	UINT	Начальный адрес slave'а. При получении запроса, в котором присутствует регистр с адресом < **c_uiStartAddr**, slave отправит master-устройству ошибку **ILLEGAL_DATA_ADDRESS**

5.6.3 ФБ MB_TcpRequest

ПРИМЕЧАНИЕ
Одновременно (в пределах одного цикла контроллера) могут вызываться не более 20 экземпляров асинхронно выполняемых блоков (см. п. 6.5.3).

Функциональный блок **MB_TcpRequest** используется для работы в режиме **Modbus TCP Master**. По переднему фронту на входе **xExecute** происходит отправка запроса, определяемого параметрами **usiUnitId**, **eFuncCode, uiDataAddr** и **uiDataCount** по протоколу **Modbus TCP** (если **xIsRtuOverTcp** = **FALSE**) или **Modbus RTU over TCP** (если **xIsRtuOverTcp** = **TRUE**) через TCP-соединение, определяемое дескриптором **hConnection**, полученным от ФБ TCP_Client. Считываемые или записываемые данные размещаются в буфере, расположенном по указателю **pData** размером **szSize** байт.

Ответ от slave-устройства ожидается в течение времени **tTimeout**. 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**. В случае получения корректного ответа выход **xDone** принимает значение **TRUE**, а выход **eError** = **NO_ERROR**. В случае получения ответа с кодом ошибки Modbus **xError** принимает значение **TRUE**, а выход **eError** содержит код ошибки (при этом выход **xDone** не принимает значение **TRUE**). Для отправки следующего запроса следует создать передний фронт на входе **xExecute**.

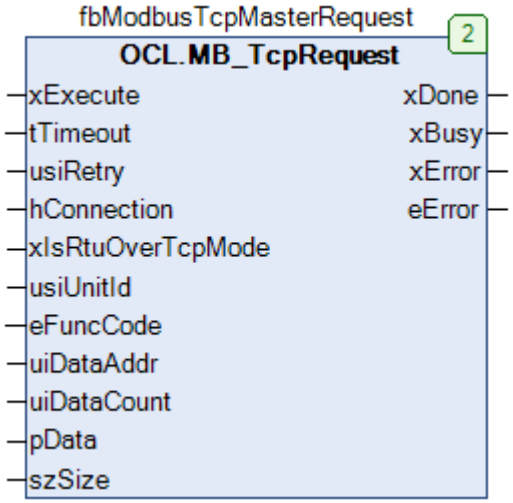


Рисунок 5.6.3 – Внешний вид ФБ MB_TcpRequest на языке CFC

Таблица 5.6.3 – Описание входов и выходов ФБ MB_TcpRequest

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
hConnection	CAA.HANDLE	Дескриптор соединения, полученный от ФБ TCP_Client
xIsRtuOverTcp	BOOL	Используемый протокол: **FALSE** – Modbus TCP, **TRUE** – Modbus RTU over TCP
usiUnitId	USINT	Адрес slave-устройства (значение по умолчанию – **16#FF**, другие значения требуются только при работе со шлюзами Modbus TCP/Modbus Serial и специфичными slave-устройствами)
eFuncCode	MB_FC	Используемая функция Modbus
uiDataAddr	UINT	Начальный адрес бита/регистра в запросе
uiDataCount	UINT	Число битов/регистров в запросе
pData	CAA.PVOID	Указатель на буфер записываемых или считываемых данных. **Примечание**: для функции **WRITE_SINGLE_COIL** следует использовать указатель на переменную типу **WORD** с возможными значениям **0** и **1**. Во всех остальных битовых функциях используется битовая маска
szSize	CAA.SIZE	Размер буфера в байтах
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	ERROR	Статус работы ФБ (или код ошибки)

5.6.4 ФБ MB_TcpSlave

ПРИМЕЧАНИЕ
Одновременно (в пределах одного цикла контроллера) могут вызываться не более 20 экземпляров асинхронно выполняемых блоков (см. п. 6.5.3).

Функциональный блок **MB_TcpSlave** используется для работы в режиме **Modbus TCP Slave**. Пока вход **xEnable** имеет значение **TRUE**, блок находится в работе. На входе **sIpAddr** следует указать IP-адрес используемого сетевого интерфейса контроллера, на входе **uiPort** – используемый порт.

Вход **usiUnitId** определяет адрес slave'а (slave также отвечает на запросы с Unit ID = 255). Под регистры slave'а выделяется область памяти по указателю **pData** размером **szSize** байт.

Блок поддерживает все стандартные функции Modbus, приведенные в таблице 2.2, а также функцию **20 (Read File Record).** Для функции **20** поддерживается доступ к 8 файлам, нумерация файлов ведется с **1**.

Блок поддерживает до 16 одновременно подключенных клиентов. Максимально допустимое число клиентов определяется глобальным параметром библиотеки **g_c_usiMaxCountClients** (вкладка **Settings** в дереве библиотеки в **Менеджере библиотек**). Значение по умолчанию – **1**.

В реализации блока все области памяти Modbus наложены друг на друга и имеют общую адресацию (**MODBUS Data Model with only 1 block** согласно спецификации Modbus). Максимальный размер буфера slave'а – **65536** регистров.

При получении запроса от master-устройства выход **xNewRequest** на один цикл контроллера принимает значение **TRUE**, при этом выход **stRequestInfo** содержит информацию о полученном запросе. Если контроллер одновременно опрашивается несколькими master-устройствами, то будет отображена информация о последнем полученном в цикле запросе.

Блок позволяет запретить обработку определенных запросов, полученных от master-устройства. Для этого на вход **pastForbiddenRequest** передается указатель на структуру (или массив структур) типа MB_REQ_INFO, а на входе **szForbiddenRequests** указывается размер этой структуры или массива в байтах. Каждый экземпляр структуры описывает один (или несколько – при использовании специальных заполнителей, см. п. 5.4.5) запрещенный запрос. В случае запрещенного запроса slave отправляет master-устройству ответ с кодом ошибки (см. подробнее в п. 5.4.5). В случае получения запрещенного запроса с функцией записи полученные значения игнорируются.

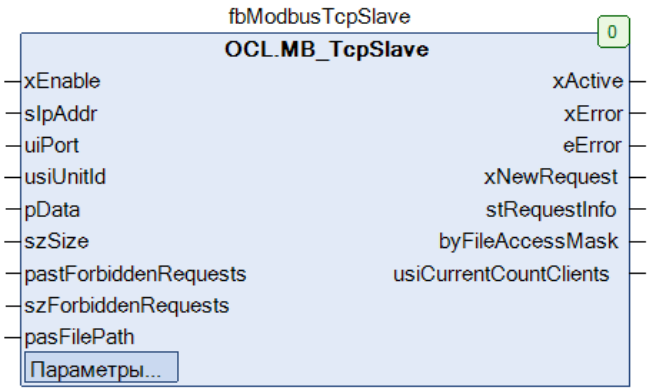


Рисунок 5.6.4 – Внешний вид ФБ MB_TcpSlave на языке CFC

Таблица 5.6.4 – Описание входов и выходов ФБ MB_TcpSlave

Название	Тип	Описание
Входы		
xEnable	BOOL	По переднему фронту происходит включение slave'а, по заднему – отключение
sIpAddr	STRING	IP-адрес используемого сетевого интерфейса в формате IPv4 ('xxx.xxx.xxx.xxx').
uiPort	UINT	Используемый порт
usiUnitId	USINT	Адрес slave-устройства
pData	CAA.PVOID	Указатель на буфер данных slave'а
szSize	CAA.SIZE	Размер буфера в байтах
pastForbiddenRequest	CAA.PVOID	Указатель на массив структур запрещенных запросов
szForbiddenRequests	CAA.SIZE	Размер массива структур запрещенных запросов в байтах
pasFilePath	POINTER TO ARRAY [1..8] OF STRING	Указатель на массив путей к файлам архива (для функции **20**)
Выходы		
xActive	BOOL	**TRUE** – ФБ находится в работе
xError	BOOL	Принимает значение **TRUE** в случае возникновения ошибки
eError	ERROR	Статус работы ФБ (или код ошибки)
xNewRequest	BOOL	Принимает **TRUE** на один цикл контроллера при получении запроса от master-устройства
stRequestInfo	MB_REQ_INFO	Информация о полученном запросе (актуальна, пока **xNewRequest** = **TRUE**)
byFileAccessMask	BYTE	Битовая маска открытых файлов (для функции **20**)
usiCurrentCountClients	USINT	Число клиентов, подключенных к slave'у. Максимальное число клиентов определяется глобальным параметром библиотеки **g_c_usiMaxCountClients** (вкладка **Settings** в дереве библиотеки в **Менеджере библиотек**). Максимальное число клиентов – **16**
Параметры		
c_xReverseByteOrder	BOOL	**TRUE** – изменить порядок байт в буфере данных slave'а на противоположный
c_xReverseByteOrderFiles	BOOL	**TRUE** – изменить порядок байт при передаче файлов для функции **20** на противоположный. Для работы с **MasterOPC Universal Modbus Server** следует установить значение **TRUE**
c_uiStartAddr	UINT	Начальный адрес slave'а. При получении запроса, в котором присутствует регистр с адресом < **c_uiStartAddr**, slave отправит master-устройству ошибку **ILLEGAL_DATA_ADDRESS**
c_usiAmountBytes	USINT	Размер записи для функции **20**. По спецификации Modbus этот параметр должен иметь значение **2**. Для работы с **MasterOPC Universal Modbus Server** следует установить значение **10**

5.6.5 ФБ MB_SerialReadFile

Функциональный блок **MB_SerialReafFile** используется для чтения файлов со slave-устройств с помощью функции Modbus **20 (0x14) Read File Record** по интерфейсу RS-232/RS-485. Контроллер в данном случае выступает в роли master-устройства.

По переднему фронту на входе **xExecute** происходит отправка запроса, определяемого параметрами **usiSlaveId** (адрес slave-устройства), **pastSubrequestParams** (указатель на параметры подзапросов, см. структуру MB_SUBREQUEST_PARAMS) и **usiSubrequestCount** (число подзапросов в запросе). Считанные записи размещаются в буфере, расположенном по указателю **pData** размером **szSize** байт.

Ответ от slave-устройства ожидается в течение времени **tTimeout**. 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**. В случае получения корректного ответа выход **xDone** принимает значение **TRUE**, а выход **eError** = **NO_ERROR**. В случае получения ответа с кодом ошибки Modbus **xError** принимает значение **TRUE**, а выход **eError** содержит код ошибки (при этом выход **xDone** не принимает значение **TRUE**). Для отправки следующего запроса следует создать передний фронт на входе **xExecute**.

См. пример использования блока для считывания заголовка архива с теплосчетчика ВИС.Т (описание протокола обмена).

ПРИМЕЧАНИЕ
Одновременно (в пределах одного цикла контроллера) могут вызываться не более 20 экземпляров асинхронно выполняемых блоков (см. п. 6.5.3).

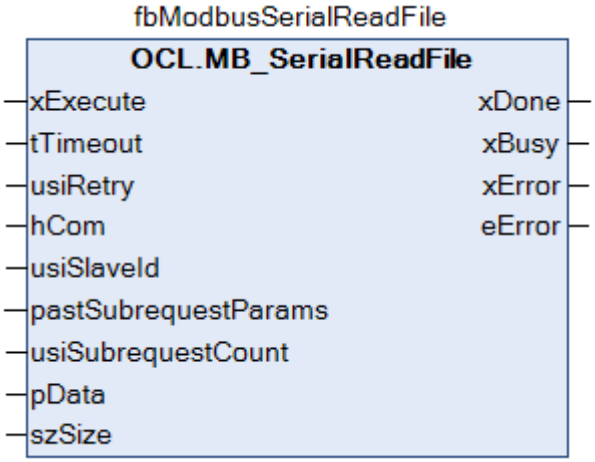


Рисунок 5.6.5 – Внешний вид ФБ MB_SerialReadFile на языке CFC

Таблица 5.6.5 – Описание входов и выходов ФБ MB_SerialReadFile

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
usiSlaveId	USINT	Адрес slave-устройства
pastSubrequestParams	CAA.PVOID	Указатель на данные подзапросов (на экземпляр структуры MB_SUBREQUEST_PARAMS или на массив таких экземпляров)
usiSubrequestCount	USINT(1..8)	Количество подзапросов в запросе
pData	CAA.PVOID	Указатель на буфер считываемых записей
szSize	CAA.SIZE	Размер буфера в байтах
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	ERROR	Статус работы ФБ (или код ошибки)

5.7 ФБ нестандартных протоколов

5.7.1 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. 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 **wStopChar** = **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
Одновременно (в пределах одного цикла контроллера) могут вызываться не более 20 экземпляров асинхронно выполняемых блоков (см. п. 6.5.3).

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

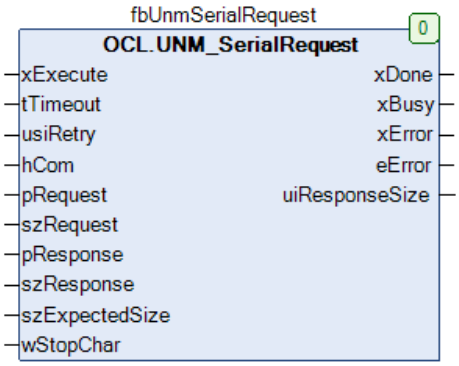


Рисунок 5.7.1 – Внешний вид ФБ UNM_SerialRequest на языке CFC

Таблица 5.7.1 – 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
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	ERROR	Статус работы ФБ (или код ошибки)
uiResponseSize	UINT	Размер полученного ответа в байтах

5.7.2 ФБ UNM_TcpRequest

Функциональный блок **UNM_TcpRequest** используется для реализации нестандартного протокола поверх протокола **TCP**. По переднему фронту на входе **xExecute** происходит отправка содержимого буфера запроса, расположенного по указателю **pRequest**, размером **szRequest** байт через соединение, определяемое дескриптором **hConnection**, полученным от ФБ TCP_Client. 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 **wStopChar** = **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
Одновременно (в пределах одного цикла контроллера) могут вызываться не более 20 экземпляров асинхронно выполняемых блоков (см. п. 6.5.3).

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

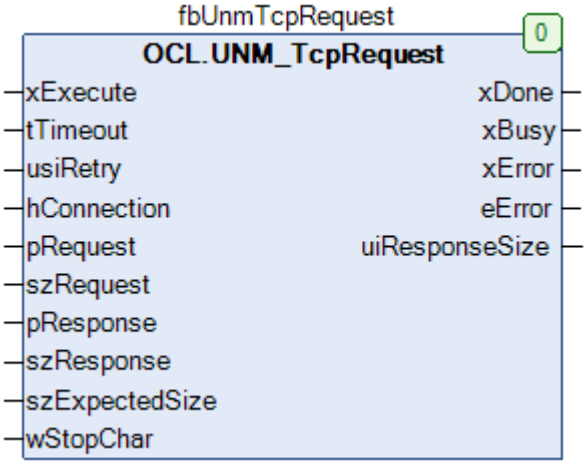


Рисунок 5.7.2 – Внешний вид ФБ UNM_TcpRequest на языке CFC

Таблица 5.7.2 – Описание входов и выходов ФБ UNM_TcpRequest

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
hConnection	CAA.HANDLE	Дескриптор TCP-соединения, полученный от ФБ TCP_Client
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 отличный 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	ERROR	Статус работы ФБ (или код ошибки)
uiResponseSize	UINT	Размер полученного ответа в байтах

5.7.3 ФБ UNM_UdpRequest

Функциональный блок **UNM_UdpRequest** используется для реализации нестандартного протокола поверх протокола **UDP**. По переднему фронту на входе **xExecute** происходит отправка содержимого буфера запроса, расположенного по указателю **pRequest**, размером **szRequest** байт на IP-адрес **sServerIpAddr** и порт **uiServerPort**. На стороне контроллера для отправки используется порт **uiLocalPort** и IP-адрес **0.0.0.0**. (т. е. отправка запроса осуществляется по всем доступным интерфейсам).

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 **wStopChar** = **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
Одновременно (в пределах одного цикла контроллера) могут вызываться не более 20 экземпляров асинхронно выполняемых блоков (см. п. 6.5.3).

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

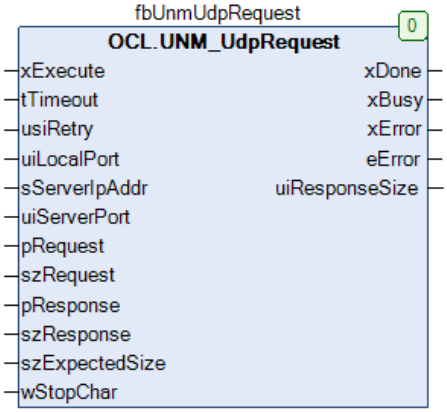


Рисунок 5.7.3 – Внешний вид ФБ UNM_UdpRequest на языке CFC

Таблица 5.7.3 – Описание входов и выходов ФБ UNM_UdpRequest

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
uiLocalPort	UINT	Порт контроллера, через который отправляется запрос
sServerIpAddr	STRING	IP-адрес slave-устройства в формате IPv4 (‘xxx.xxx.xxx.xxx’)
uiServerPort	UINT	Порт slave-устройства
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 отличный 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	ERROR	Статус работы ФБ (или код ошибки)
uiResponseSize	UINT	Размер полученного ответа в байтах

5.8 Функции и ФБ преобразования данных

5.8.1 ФБ DWORD_TO_WORD2

Функциональный блок **DWORD_TO_WORD2** используется для преобразования переменной типа **DWORD** в две переменные типа **WORD**.

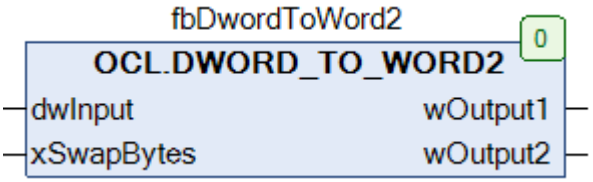


Рисунок 5.8.1 – Внешний вид ФБ DWORD_TO_WORD2 на языке CFC

Таблица 5.8.1 – Описание входов и выходов ФБ DWORD_TO_WORD2

Название	Тип	Описание
Входы		
dwInput	DWORD	Исходная переменная
xSwapBytes	BOOL	**TRUE** – выполнить перестановку байт (A1 B2 C3 D4 ---> B2 A1 D4 C3)
Выходы		
wOutput1	WORD	Старшее слово исходной переменной
wOutput2	WORD	Младшее слово исходной переменной

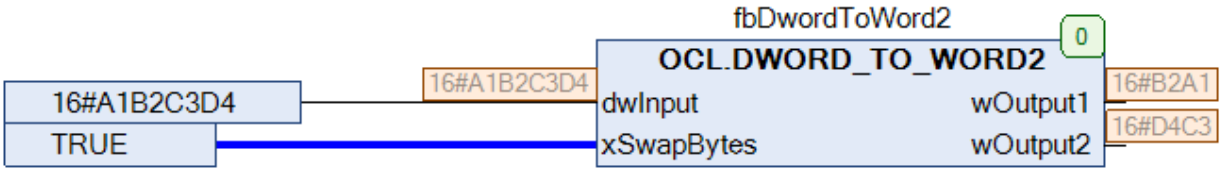


Рисунок 5.8.2 – Пример использования ФБ DWORD_TO_WORD2 на языке CFC

5.8.2 ФБ REAL_TO_WORD2

Функциональный блок **REAL_TO_WORD2** используется для преобразования переменной типа **REAL** в две переменные типа **WORD**.

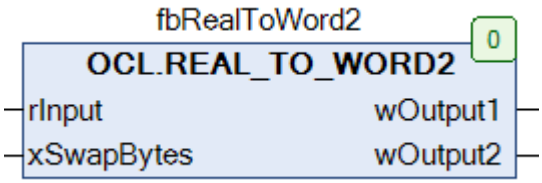


Рисунок 5.8.3 – Внешний вид ФБ REAL_TO_WORD2 на языке CFC

Таблица 5.8.2 – Описание входов и выходов ФБ REAL_TO_WORD2

Название	Тип	Описание
Входы		
rInput	REAL	Исходная переменная
xSwapBytes	BOOL	**TRUE** – выполнить перестановку байт (A1 B2 C3 D4 ---> B2 A1 D4 C3)
Выходы		
wOutput1	WORD	Старшее слово исходной переменной
wOutput2	WORD	Младшее слово исходной переменной

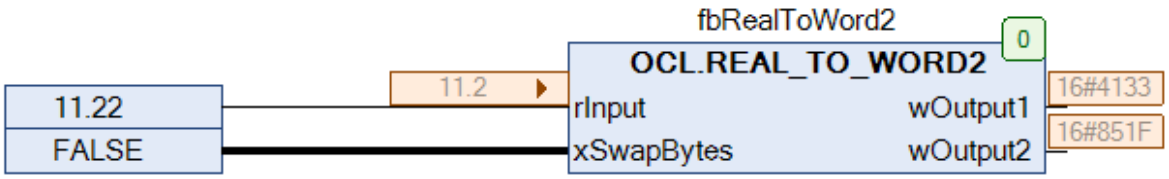


Рисунок 5.8.4 – Пример использования ФБ REAL_TO_WORD2 на языке CFC (см. онлайн-конвертер для проверки)

5.8.3 Функция WORD2_TO_DWORD

Функция **WORD2_TO_DWORD** используется для преобразования двух переменных типа **WORD** в переменную типа **DWORD**.

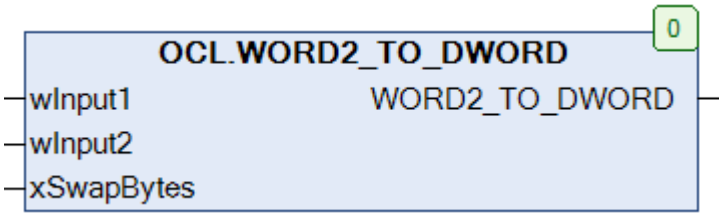


Рисунок 5.8.5 – Внешний вид функции WORD2_TO_DWORD на языке CFC

Таблица 5.8.3 – Описание входов и выходов функции WORD2_TO_DWORD

Название	Тип	Описание
Входы		
wInput1	WORD	Исходная переменная 1
wInput2	WORD	Исходная переменная 2
xSwapBytes	BOOL	**TRUE** – выполнить перестановку байт (A1 B2 C3 D4 ---> B2 A1 D4 C3)
Выходы		
WORD2_TO_DWORD	DWORD	Новая переменная типа **DWORD**

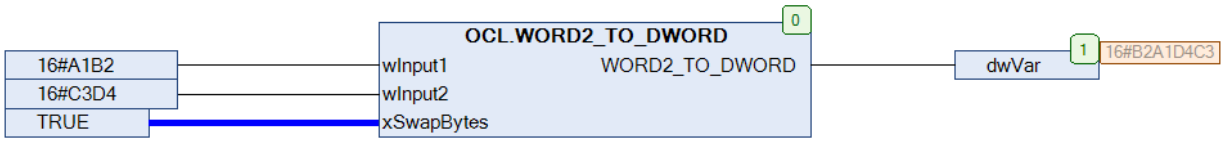


Рисунок 5.8.6 – Пример использования функции WORD2_TO_DWORD на языке CFC

5.8.4 Функция WORD2_TO_REAL

Функция **WORD2_TO_REAL** используется для преобразования двух переменных типа **WORD** в переменную типа **REAL**.

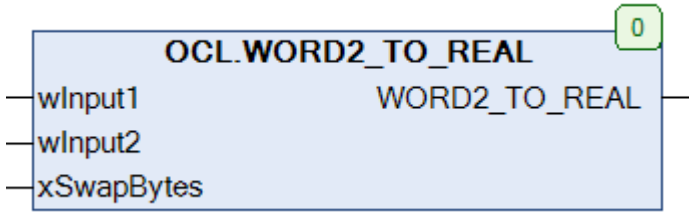


Рисунок 5.8.7 – Внешний вид функции WORD2_TO_REAL на языке CFC

Таблица 5.8.4 – Описание входов и выходов функции WORD2_TO_REAL

Название	Тип	Описание
Входы		
wInput1	WORD	Исходная переменная 1
wInput2	WORD	Исходная переменная 2
xSwapBytes	BOOL	**TRUE** – выполнить перестановку байт (A1 B2 C3 D4 ---> B2 A1 D4 C3)
Выходы		
WORD2_TO_REAL	REAL	Новая переменная типа **REAL**

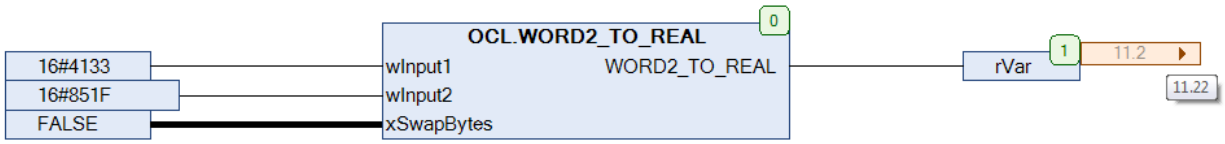


Рисунок 5.8.8 – Пример использования функции WORD2_TO_REAL на языке CFC (см. онлайн-конвертер для проверки)

5.8.5 Функция SWAP_DATA

Функция **SWAP_DATA** используется для копирования данных из одного буфера в другой с перестановкой байт и регистров.

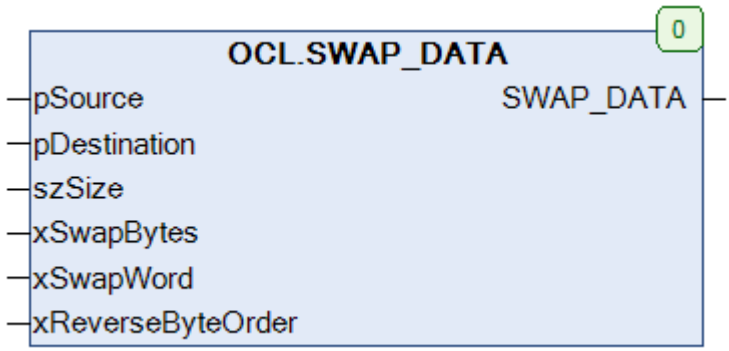


Рисунок 5.8.9 – Внешний вид функции SWAP_DATA на языке CFC

Таблица 5.8.5 – Описание входов и выходов функции SWAP_DATA

Название	Тип	Описание
Входы		
pSource	CAA.PVOID	Указатель на буфер исходных данных
pDestination	CAA.PVOID	Указатель на буфер, в который будут скопированы данные. Может совпадать с **pSource** – тогда после изменения данные будут помещены в тот же буфер
szSize	CAA.SIZE	Размер копируемых данных в байтах (должен не превышать размеры буферов)
xSwapBytes	BOOL	**TRUE** – выполнить перестановку байт (A1 B2 C3 D4 ---> B2 A1 D4 C3)
xSwapWord	BOOL	**TRUE** – выполнить перестановку регистров (A1 B2 C3 D4 ---> C3 D4 A1 B2)
xReverseByteOrder	BOOL	**TRUE** – изменить порядок байт на противоположный (A1 B2 C3 D4 ---> D4 C3 B2 A1). Если данный вход имеет значение **TRUE**, то входы **xSwapBytes** и **xSwapWord** не обрабатываются
Выходы		
SWAP_DATA	BOOL	**TRUE** – операция выполнена

5.9 Примеры

5.9.1 СПК1xx [M01] (Modbus RTU Master) + модули Mx110

В качестве примера будет рассмотрена настройка обмена с модулями Mx110 (МВ110-8А, МВ110-16Д, МУ110-16Р) с использованием библиотеки **OwenCommunication**. В примере используется библиотека версии **3.5.11.6**.

Реализуемый алгоритм: если значение первого аналогового входа модуля **МВ110-8А** превышает **30** и при этом первый дискретный вход модуля **МВ110-16Д** имеет значение **TRUE** (замкнут), то первому дискретному выходу модуля **МУ110-16Р** присваивается значение **TRUE** (замкнут). Во всех остальных случаях дискретному выходу присваивается значение **FALSE** (разомкнут).

Структурная схема примера приведена на рисунке ниже:

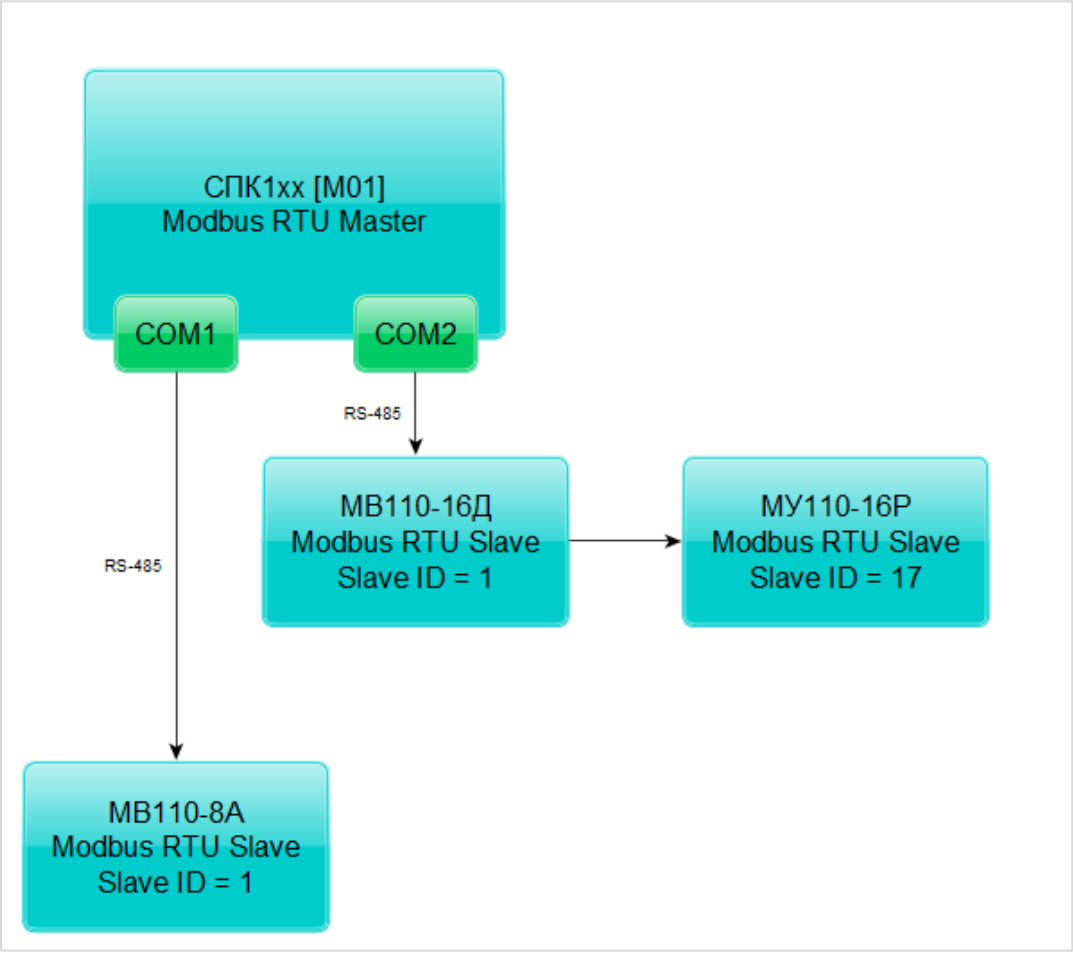


Рисунок 5.9.1 – Структурная схема примера

Пример создан в среде **CODESYS V3.5 SP17 Patch 3** и подразумевает запуск на **СПК1xx [M01]** с таргет-файлом **3.5.17.31.** 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_OwenCommunicationModbusRtuMaster_3517v1.projectarchive

Сетевые параметры модулей приведены в таблице ниже:

Таблица 5.9.1 – Сетевые параметры модулей Mx110

Параметр	МВ110-8А	МВ110-16Д	МУ110-16Р
COM-порт контроллера, к которому подключен модуль	COM1	COM2	
ID COM-порта	1	2	
Адрес модуля	1	1	17
Скорость обмена	115200		
Количество бит данных	8		
Контроль четности	Отсутствует		
Количество стоп-бит	1		

Для настройки обмена следует:

1. Настроить модули **Mx110** с помощью программы **Конфигуратор Mx110** в соответствии с таблицей 5.9.1. Подключить модули к COM-портам контроллера в соответствии с рисунком 5.9.1.

2. Установить в CODESYS библиотеку **OwenCommunication** (см. п. 5.2).

3. Создать новый проект **CODESYS** с программой на языке **ST** или **CFC**:

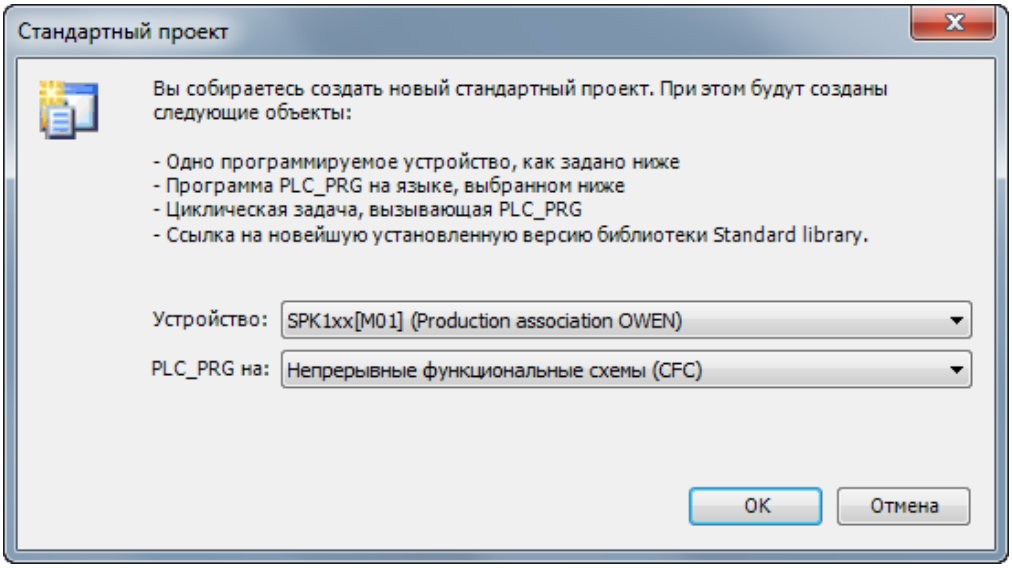


Рисунок 5.9.2 – Создание проекта CODESYS

ПРИМЕЧАНИЕ

Проект примера содержит программы на языках CFC и ST. По умолчанию используется программа на CFC (**PLC_PRG_CFC**). Для работы с программой на ST следует в конфигурации задач для задачи **MainTask** удалить вызов **PLC_PRG_CFC** и добавить вызов **PLC_PRG_ST**.

4. Добавить в проект библиотеки **OwenCommunication** и **Util** (**Менеджер библиотек – Добавить библиотеку**).

ПРИМЕЧАНИЕ

Библиотека **Util** используется только в программе на языке CFC.

5. Объявить в программе следующие переменные:

PLC_PRG_CFC

```
PROGRAM PLC_PRG_CFC
VAR
    fbComControl1:  OCL.COM_Control;        // ФБ управления портом COM1
    fbComControl2:  OCL.COM_Control;        // ФБ управления портом COM2
    fbMV110_8A_AI1: OCL.MB_SerialRequest;   // ФБ опроса модуля MB110-8A
    fbMV110_16D_DI: OCL.MB_SerialRequest;   // ФБ опроса модуля MB110-16Д
    fbMU110_16R_DO: OCL.MB_SerialRequest;   // ФБ опроса модуля MУ110-16Р

    rAI1:           REAL;                   // значение 1-го входа модуля MB110-8A
    wDiMask:        WORD;                   // битовая маска входов модуля MB110-16Д
    wDoMask:        WORD;                   // битовая маска выходов модуля MУ110-16Р
    xDi0:           BOOL;                   // значение 1-го входа модуля MB110-16Д
    xDo0:           BOOL;                   // значение 1-го выхода модуля MУ110-16Р

    awAI1:          ARRAY [0..1] OF WORD;   // регистры, считанные с модуля MB110-8A

    fbUnpackWord:   UTIL.WORD_AS_BIT;       // ФБ распаковки битовой маски
    fbPackWord:     UTIL.BIT_AS_WORD;       // ФБ упаковки битовой маски

    iStateCom1:     INT;                    // шаг опроса по порту COM1
    iStateCom2:     INT;                    // шаг опроса по порту COM2
END_VAR
```

Рисунок 5.9.3 – Объявление переменных в программе

ПРИМЕЧАНИЕ

Переменные **fbUnpackWord** и **fbPackWord** используются только в программе на языке CFC. Переменные **iStateCom1** и **iStateCom2** используется только в программе на языке ST.

6. Нажать **ПКМ** на программу, выбрать команду **Добавление объекта – Действие** и добавить действия с названиями **COM1** и **COM2** (язык реализации действий совпадает с языком программы). В рамках примера действия используются для повышения читабельности кода.

7. Код действий и программы на языке CFC будет выглядеть следующим образом:

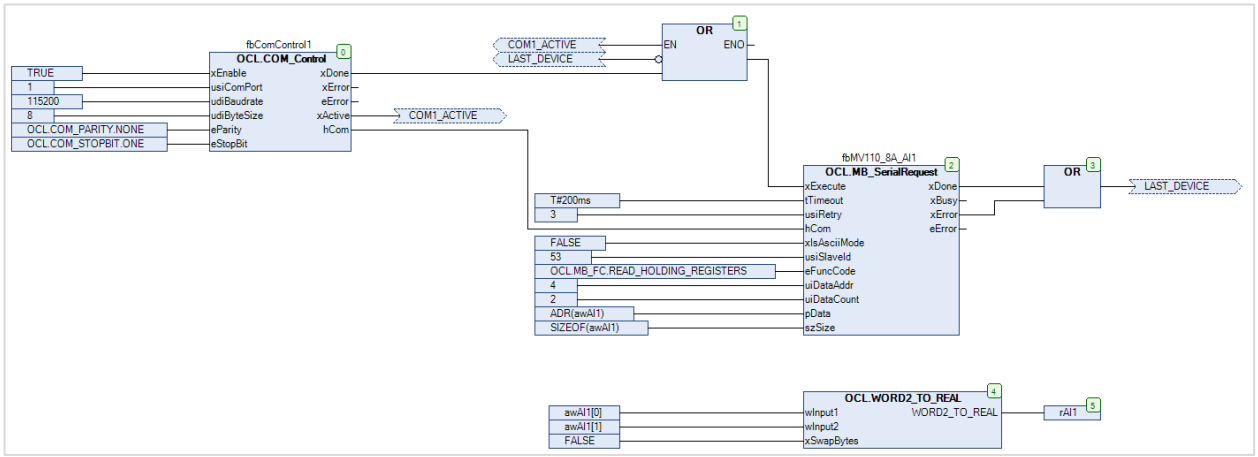


Рисунок 5.9.4 – Код действия COM1

При первом вызове действия происходит открытие COM-порта с заданными настройками с помощью экземпляра ФБ COM_Control (**блок 0**). После успешного открытия порта на выходе **xDone** генерируется единичный импульс, что приводит к вызову экземпляра ФБ MB_SerialRequest (**блок 2**), который производит опрос первого аналогового входа модуля **MB110-8A** с адресом **1** (**usiSlaveId**). Требуемый код функции (**eFunc**), начальный адрес регистра (**uiDataAddr**) и их количество (**uiDataCount**) приведены в РЭ на модуль. Полученные данные помещаются в переменную **awAI1** типа **ARRAY [0..1] OF WORD**. С помощью функции WORD2_TO_REAL (**блоки 4–5**) этот массив преобразуется в переменную типа **rAI1** типа **REAL**.

После завершения работы экземпляра ФБ MB_SerialRequest один из его выходов принимает значение **TRUE**: если опрос произведен успешно, то значение **TRUE** принимает выход **xDone**, если ответ не получен или получен ответ с кодом ошибки Modbus – то значение **TRUE** принимает выход **xError**. В любой из этих ситуаций с помощью **блока 1** происходит вызов экземпляра ФБ MB_SerialRequest со значением **FALSE** на входе **xExecute** (сброс блока), а потом – его следующий вызов, что приводит к началу следующего сеанса опроса.

Если порт закрыт или находится в состоянии ошибки (выход **xActive** у экземпляра ФБ COM_CONTROL имеет значение **FALSE**), то опрос прекращается.

ПРИМЕЧАНИЕ

В рамках примера для повышения читабельности схемы вместо некоторых линий связи использованы **метки соединения**.

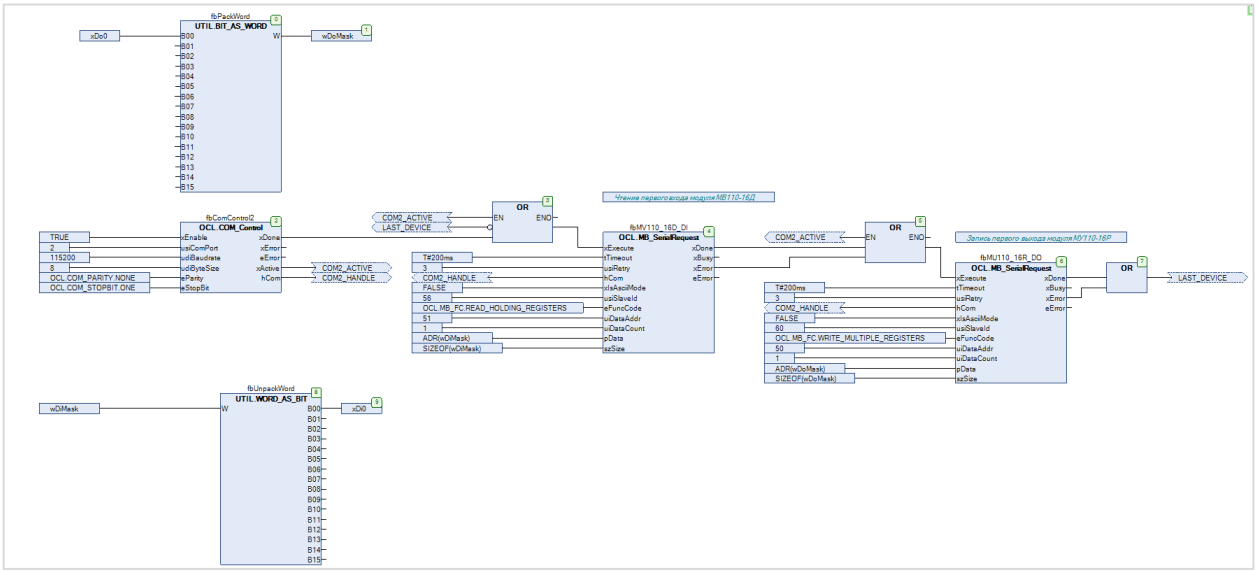


Рисунок 5.9.5 – Код действия COM2

При первом вызове действия происходит открытие COM-порта с заданными настройками с помощью ФБ COM_CONTROL (**блок 2**). После успешного открытия порта на выходе **xDone** генерируется единичный импульс, что приводит к вызову ФБ MB_SerialRequest (**блок 4**), который производит опрос битовой маски дискретных входов модуля **MB110-16Д** с адресом **1** (**usiSlaveId**). Требуемый код функции (**eFunc**), начальный адрес регистра (**uiDataAddr**) и их количество (**uiDataCount**) приведены в РЭ на модуль. Полученные данные помещаются в переменную **wDiMask** типа **WORD** и разделяются на отдельные переменные типа **BOOL** с помощью экземпляра ФБ **WORD_AS_BIT** из библиотеки **Util**.

После завершения работы ФБ MB_SerialRequest один из его выходов принимает значение **TRUE**: если опрос произведен успешно, то значение **TRUE** принимает выход **xDone**, если ответ не получен или

получен ответ с кодом ошибки Modbus – то значение **TRUE** принимает выход **xError**. В любой из этих ситуаций начинается работа следующего блока MB_SerialRequest (**блоки 5–6**), который производит запись переменной **wDoMask** в качестве битовой маски дискретных выходов модуля **МУ110-16Р** с адресом **17**. Требуемый код функции (**eFunc**), начальный адрес регистра (**uiDataAddr**) и их количество (**uiDataCount**) приведены в РЭ на модуль. Экземпляр блока **BIT_AS_WORD** (**блоки 0–1**) позволяет упаковать отдельные переменные типа **BOOL** в битовую маску типа **WORD**.

После окончания работы экземпляра последнего блока MB_SerialRequest с помощью **блока 3** происходит вызов экземпляра первого ФБ MB_SerialRequest со значением **FALSE** на входе **xExecute** (сброс блока), а потом – его следующий вызов, что приводит к началу следующего сеанса опроса.

Если порт закрыт или находится в состоянии ошибки (выход **xActive** у ФБ COM_CONTROL имеет значение **FALSE**), то опрос прекращается.

ПРИМЕЧАНИЕ
В рамках примера для повышения читабельности схемы вместо некоторых линий связи использованы **метки соединения**.

В программе **PLC_PRG_CFC** производится вызов действий **COM1** и **COM2**, а также выполнение алгоритма, описанного в условии примера.

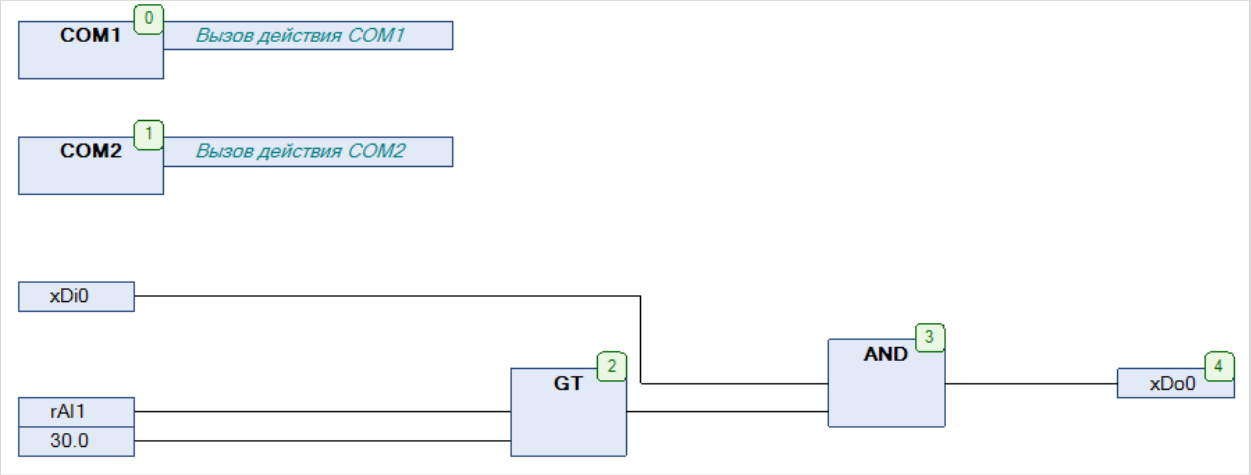


Рисунок 5.9.6 – Код программы PLC_PRG_CFC

8. Код действий и программы на языке ST будет выглядеть следующим образом:

```
CASE iStateCom1 OF

    0:  // открытие COM-порта COM1

        fbComControl1
        (
            xEnable     := TRUE,
            usiComPort  := 1,
            udiBaudrate := 115200,
            udiByteSize := 8,
            eParity     := OCL.COM_PARITY.NONE,
            eStopBit    := OCL.COM_STOPBIT.ONE
        );

        IF fbComControl1.xDone THEN
            iStateCom1 := 1;
        END_IF


    1: // опрос MB110-8A

        fbMV110_8A_AI1
        (
            xExecute        := fbComControl1.xActive,
            tTimeout        := T#200MS,
            usiRetry        := 3,
            hCom            := fbComControl1.hCom,
            xIsAsciiMode    := FALSE ,
            usiSlaveId      := 1,
            eFuncCode       := OCL.MB_FC.READ_HOLDING_REGISTERS,
            uiDataAddr      := 4,
            uiDataCount     := 2,
            pData           := ADR(awAI1),
            szSize          := SIZEOF(awAI1)
        );

        IF fbMV110_8A_AI1.xDone OR fbMV110_8A_AI1.xError THEN
            // после выполнения блока его надо сбросить
            fbMV110_8A_AI1(xExecute := FALSE);

            rAI1 := OCL.WORD2_TO_REAL(awAI1[0], awAI1[1], FALSE);

            iStateCom1 := 2;
        END_IF


    2: // здесь можно добавить опрос следующего устройства
       // после опроса последнего опроса возвращаемся к опросу первого
            iStateCom1 := 1;

END_CASE
```

Рисунок 5.9.7 – Код действия COM1

При первом вызове действия в **шаге 0** происходит открытие COM-порта с заданными настройками с помощью экземпляра ФБ COM_Control. После успешного открытия порта на выходе **xDone** генерируется единичный импульс, что приводит к переходу на **шаг 1**.

На **шаге 1** выполняется вызов экземпляра ФБ MB_SerialRequest, который производит опрос первого аналогового входа модуля **MB110-8A** с адресом **1** (**usiSlaveId**). Требуемый код функции (**eFunc**), начальный адрес регистра (**uiDataAddr**) и их количество (**uiDataCount**) приведены в РЭ на модуль. Полученные данные помещаются в переменную **awAI1** типа **ARRAY [0..1] OF WORD**. С помощью функции WORD2_TO_REAL этот массив преобразуется в переменную типа **rAI1** типа **REAL**.

После завершения работы экземпляра ФБ MB_SerialRequest один из его выходов принимает значение **TRUE**: если опрос произведен успешно, то значение **TRUE** принимает выход **xDone**, если ответ не получен или получен ответ с кодом ошибки Modbus – то значение **TRUE** принимает выход **xError**. В любой из этих ситуаций происходит вызов экземпляра ФБ MB_SerialRequest со значением **FALSE** на входе **xExecute** (сброс блока) и переход на **шаг 2**.

В рамках примера на **шаге 2** происходит переход на **шаг 1**, что приводит к началу следующего сеанса опроса. В случае опроса нескольких устройств (или одного устройства с помощью нескольких запросов) число шагов можно увеличить. После выполнения последнего шага должен происходить переход на шаг опроса первого устройства.

Если порт закрыт или находится в состоянии ошибки (выход **xActive** у экземпляра ФБ COM_Control имеет значение **FALSE**), то опрос прекращается.

```
CASE iStateCom2 OF

    0:  // открытие COM-порта COM2

        fbComControl2
        (
            xEnable     := TRUE,
            usiComPort  := 2,
            udiBaudrate := 115200,
            udiByteSize := 8,
            eParity     := OCL.COM_PARITY.NONE,
            eStopBit    := OCL.COM_STOPBIT.ONE
        );

        IF fbComControl2.xDone THEN
            iStateCom2 := 1;
        END_IF

    1:  // опрос модуля МВ110-16Д

        fbMV110_16D_DI
        (
            xExecute        := fbComControl2.xActive,
            tTimeout        := T#200MS,
            usiRetry        := 3,
            hCom            := fbComControl2.hCom,
            xIsAsciiMode    := FALSE ,
            usiSlaveId      := 1,
            eFuncCode       := OCL.MB_FC.READ_HOLDING_REGISTERS,
            uiDataAddr      := 51,
            uiDataCount     := 1,
            pData           := ADR(wDiMask),
            szSize          := SIZEOF(wDiMask)
        );

        IF fbMV110_16D_DI.xDone OR fbMV110_16D_DI.xError THEN
            // после выполнения блока его надо сбросить
            fbMV110_16D_DI(xExecute := FALSE);

            xDi0 := wDiMask.0;

            iStateCom2 := 2;
        END_IF

    2:  // опрос модуля МУ110-16Р

        wDoMask.0 := xDo0;


        fbMU110_16R_DO
        (
            xExecute        := fbComControl2.xActive,
            tTimeout        := T#200MS,
            usiRetry        := 17,
            hCom            := fbComControl2.hCom,
            xIsAsciiMode    := FALSE ,
            usiSlaveId      := 60,
            eFuncCode       := OCL.MB_FC.WRITE_MULTIPLE_REGISTERS,
            uiDataAddr      := 50,
            uiDataCount     := 1,
            pData           := ADR(wDoMask),
            szSize          := SIZEOF(wDoMask)
        );

        IF fbMU110_16R_DO.xDone OR fbMU110_16R_DO.xError THEN
            // после выполнения блока его надо сбросить
            fbMU110_16R_DO(xExecute := FALSE);

            // возвращаемся к опросу первого модуля
            iStateCom2 := 1;
        END_IF
```

Рисунок 5.9.8 – Код действия COM2

При первом вызове действия в **шаге 0** происходит открытие COM-порта с заданными настройками с помощью экземпляра ФБ COM_Control. После успешного открытия порта на выходе **xDone** генерируется единичный импульс, что приводит к переходу на **шаг 1**.

На **шаге 1** выполняется вызов экземпляра ФБ MB_SerialRequest, который производит опрос битовой маски дискретных входов модуля **МВ110-16Д** с адресом **1** (**usiSlaveId**). Требуемый код функции (**eFunc**), начальный адрес регистра (**uiDataAddr**) и их количество (**uiDataCount**) приведены в РЭ на модуль. Полученные данные помещаются в переменную **wDiMask** типа **WORD**, из которой происходит копирование данных в нужные переменные типа **BOOL** с помощью побитового доступа (*переменная.номер_бита*).

После завершения работы экземпляра ФБ MB_SerialRequest один из его выходов принимает значение **TRUE**: если опрос произведен успешно, то значение **TRUE** принимает выход **xDone**, если ответ не получен или получен ответ с кодом ошибки Modbus – то значение **TRUE** принимает выход **xError**. В любой из этих ситуаций происходит вызов экземпляра ФБ MB_SerialRequest со значением **FALSE** на входе **xExecute** (сброс блока) и переход на **шаг 2**.

На **шаге 2** выполняется вызов экземпляра ФБ MB_SerialRequest, который производит запись переменной **wDoMask** в качестве битовой маски дискретных выходов модуля **МУ110-16Р** с адресом **17**. Требуемый код функции (**eFunc**), начальный адрес регистра (**uiDataAddr**) и их количество (**uiDataCount**) приведены в РЭ на модуль. Битовая маска может формироваться из отдельных переменных типа **BOOL** с помощью побитового доступа.

В рамках примера на **шаге 2** происходит переход на **шаг 1**, что приводит к началу следующего сеанса опроса. В случае опроса нескольких устройств (или одного устройства с помощью нескольких запросов) число шагов можно увеличить. После выполнения последнего шага должен происходить переход на шаг опроса первого устройства.

Если порт закрыт или находится в состоянии ошибки (выход **xActive** у экземпляра ФБ COM_Control имеет значение **FALSE**), то опрос прекращается.

В программе **PLC_PRG_ST** производится вызов действий **COM1** и **COM2**, а также выполнение алгоритма, описанного в условии примера.

```
1  // чтобы запустить пример на ST на контроллере требуется:
2  // 1. Удалить из задачи MainTask вызов программы PLC_PRG_CFC
3  // 2. Привязать к задаче MainTask вызов программа PLC_PRG_ST
4
5  COM1();
6  COM2();
7
8  xDo0 := xDi0 AND (rAI1 > 30.0);
```

Рисунок 5.9.9 – Код программы PLC_PRG_ST

9. Загрузить проект в контроллер и запустить его.

В переменной **rAI1** будет отображаться текущее значение первого аналогового входа модуля **МВ110-8А**. В переменной **xDi0** будет отображаться текущее значение первого дискретного входа модуля **МВ110-16Д**.

Если значение **rAI1**превысит **30** и при этом значение **xDi0** будет равно **TRUE**, то в переменную **xDo0** будет записано значение **TRUE,** что приведет к замыканию первого дискретного выхода модуля **МУ110-16Р**. Если одно из условий перестанет выполняться, то выход будет разомкнут.

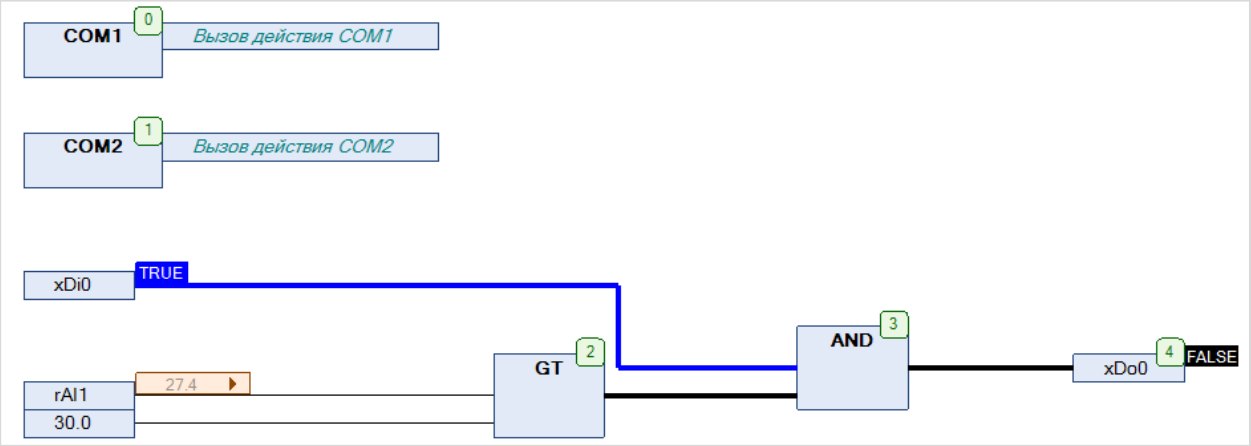


Рисунок 5.9.10 – Выполнение программы в режиме Online

5.9.2 СПК1xx [M01] (Modbus RTU Slave) + MasterOPC Universal Modbus Server

В качестве примера будет рассмотрена настройка обмена с OPC-сервером Insat MasterOPC Universal Modbus Server, который будет использоваться в режиме **Modbus RTU Master**, с помощью библиотеки **OwenCommunication**. В примере используется библиотека версии **3.5.11.6**.

Структурная схема примера приведена на рисунке ниже:

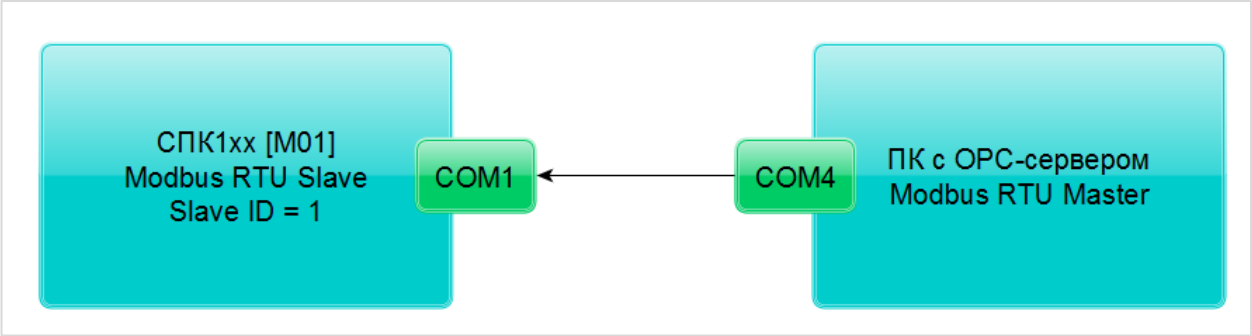


Рисунок 5.9.11 – Структурная схема примера

Пример создан в среде **CODESYS V3.5 SP17 Patch 3** и подразумевает запуск на **СПК1xx [M01]** с таргет-файлом **3.5.17.31.** 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_OwenCommunicationModbusRtuSlave_3517v1.zip

Сетевые параметры модулей приведены в таблице ниже:

Таблица 5.9.2 – Сетевые параметры устройств

Параметр	СПК1xx [M01]	ПК с OPC-сервером
COM-порт контроллера, к которому подключен модуль	COM1	COM4
ID COM-порта	1	4
Режим работы	slave	master
Slave ID	1	-
Скорость обмена	115200	
Количество бит данных	8	
Контроль четности	Отсутствует	
Количество стоп-бит	1	

Переменные примера описаны в таблице ниже:

Таблица 5.9.3 – Список переменных примера

Имя	Тип	Область памяти Modbus	Адрес регистра/бита
xVar_Opc	BOOL	Coils	0/0
wVar_Opc	WORD	Holding регистры	1
rVar_Opc	REAL		2–3
sVar_Opc	STRING(15)		4–11

ПРИМЕЧАНИЕ
В рамках примера значения переменных slave'а могут быть изменены как из OPC, так и из программы контроллера (с помощью переменных с постфиксом **_Plc**).

Для настройки обмена следует:

1. Подключить контроллер к ПК (например, с помощью конвертера ОВЕН АС4).

2. Создать новый проект **CODESYS** с программой на языке **ST** или **CFC**:

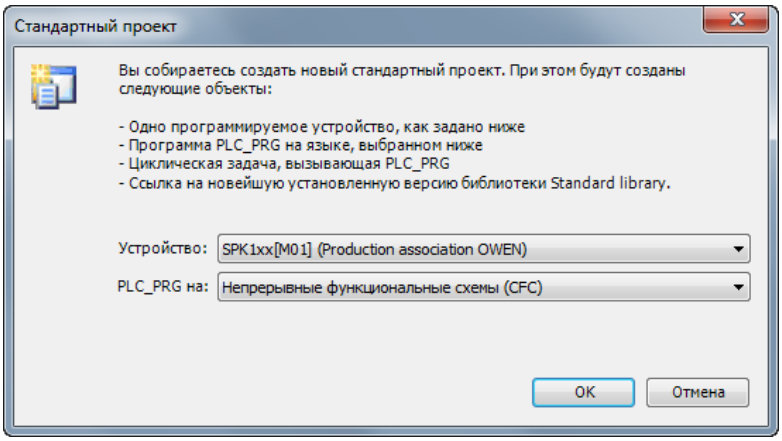


Рисунок 5.9.12 – Создание проекта CODESYS

ПРИМЕЧАНИЕ

Проект примера содержит программы на языках CFC и ST. По умолчанию используется программа на CFC (**PLC_PRG_CFC**). Для работы с программой на ST следует в конфигурации задач для задачи **MainTask** удалить вызов **PLC_PRG_CFC** и добавить вызов **PLC_PRG_ST**.

3. Добавить в проект библиотеки **OwenCommunication** и **Util** (**Менеджер библиотек – Добавить библиотеку**).

ПРИМЕЧАНИЕ

Библиотека **Util** используется только в программе на языке CFC.

4. Объявить в программе следующие переменные:

```
PROGRAM PLC_PRG_CFC
VAR
    fbComControl1:          OCL.COM_Control;        // ФБ управления портом COM1

    fbUnpackWord:           UTIL.WORD_AS_BIT;       // ФБ распаковки битовой маски
    fbPackWord:             UTIL.BIT_AS_WORD;       // ФБ упаковки битовой маски
    fbRealToWord2:          OCL.REAL_TO_WORD2;      // ФБ преобразования REAL в две переменные типа WORD


    fbModbusSerialSlave:    OCL.MB_SerialSlave;     // ФБ для реализации Modbus Slave

    awSlaveData:            ARRAY [0..15] OF WORD;  // буфер данных Modbus Slave

    (* значения, полученные от OPC *)
    xVar_Opc:               BOOL;
    wVar_Opc:               WORD;
    rVar_Opc:               REAL;
    sVar_Opc:               STRING(15);

    (* значения для передачи в OPC *)
    xVar_Plc:               BOOL;
    wVar_Plc:               WORD;
    rVar_Plc:               REAL;
    sVar_Plc:               STRING(15);

    xWrite:                 BOOL;                   // команда записи данных из программы в регистры Modbus Slave
    fbWriteEdge:            R_TRIG;                 // триггер для однократной записи
END_VAR
```

Рисунок 5.9.13 – Объявление переменных программы

ПРИМЕЧАНИЕ

Переменные **fbUnpackWord** и **fbPackWord** используются только в программе на языке CFC.

5. Код программы на языке CFC будет выглядеть следующим образом:

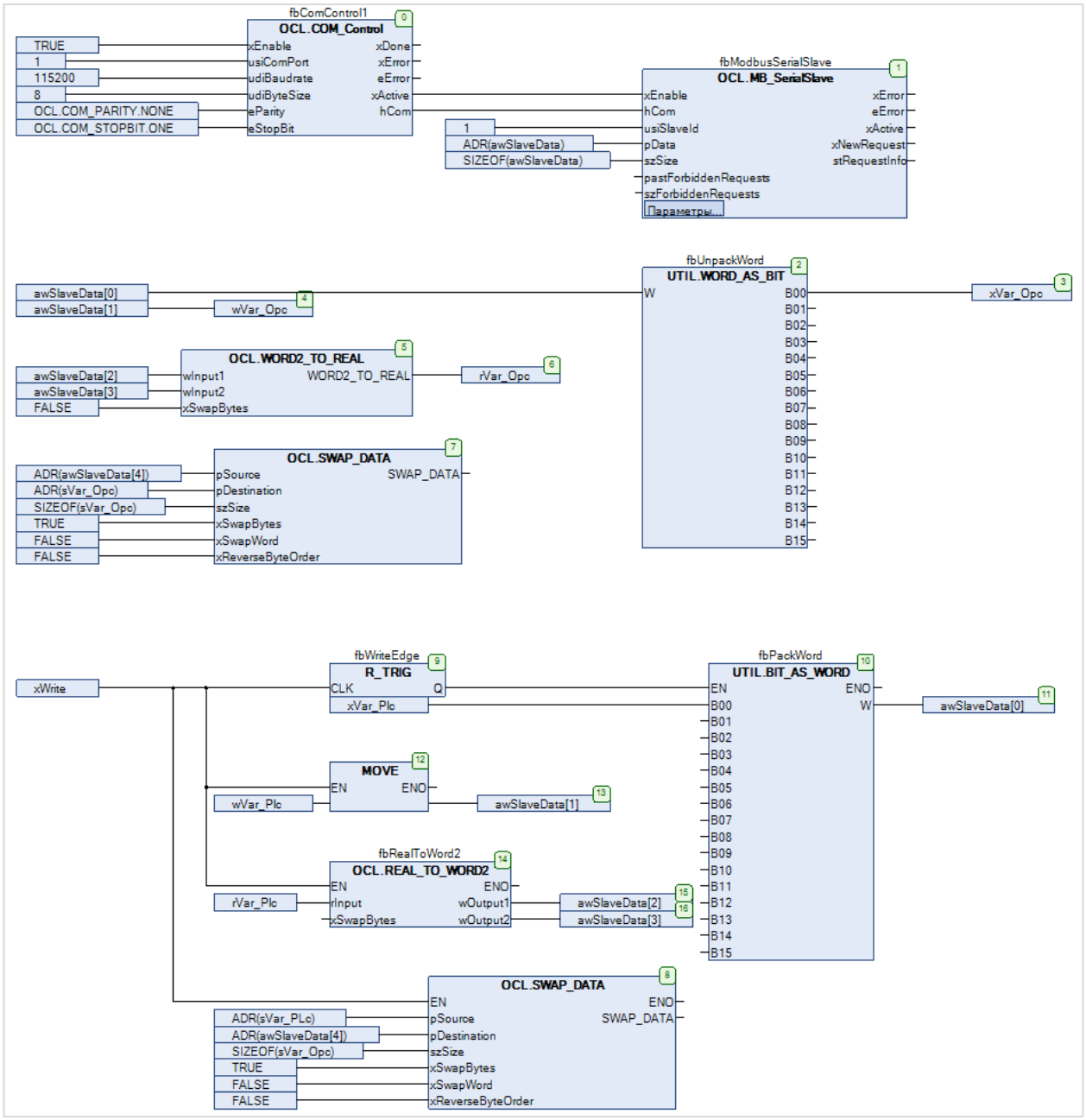


Рисунок 5.9.14 – Код программы на языке CFC

При первом вызове программы происходит открытие COM-порта с заданными настройками с помощью экземпляра ФБ COM_Control (**блок 0**). После успешного открытия порта выход **xActive** принимает и сохраняет значение **TRUE**, что приводит к вызову экземпляра ФБ MB_SerialSlave (**блок 1**), который выполняет функцию Modbus Slave с адресом **1** (**usiSlaveId**). Данные slave'а хранятся в массиве **awSlaveData**. В экземпляр ФБ передается указатель на этот массив (**pData**) и его размер в байтах (**szSize**).

В **блоках 2–7** происходит копирование данных из регистров slave'а в переменные программы. Для выделения переменных типа **BOOL** из переменной типа **WORD** используется экземпляр ФБ **WORD_AS_BIT** из библиотеки **Util**. Для преобразования двух переменных типа **WORD** в переменную

типа **REAL** используется функция WORD2_TO_REAL. Для преобразования набора переменных типа **WORD** в переменную типа **STRING** используется функция SWAP_DATA. Для обеспечения порядка байт, принятого в OPC-сервере, в процессе копирования происходит перестановка байт в регистрах (**xSwapBytes**=**TRUE**).

ПРИМЕЧАНИЕ

Для определения размера строки используется оператор **SIZEOF**, который учитывает терминирующий ноль. В рамках примера этот способ работает корректно, так как размер строки является нечетным (15 символов), и с учетом терминирующего нуля равен 16 байт (8 регистров Modbus).

По переднему фронту переменной **xWrite** (**блок 9**) происходит запись переменных программы в регистры slave'a. Для записи переменной типа **BOOL** используется экземпляр ФБ **BIT_AS_WORD** из библиотеки **Util**. Для записи переменной типа **REAL** используется экземпляр ФБ REAL_TO_WORD2. Для записи переменной типа **STRING** используется функция SWAP_DATA.

6. Код программы на языке ST будет выглядеть следующим образом:

```
// чтобы запустить пример на ST на контроллере требуется:
// 1. Удалить из задачи MainTask вызов программа PLC_PRG_CFC
// 2. Привязать к задаче MainTask вызов программы PLC_PRG_ST


fbComControl1
(
    xEnable     := TRUE,
    usiComPort  := 1,
    udiBaudrate := 115200,
    udiByteSize := 8,
    eParity     := OCL.COM_PARITY.NONE,
    eStopBit    := OCL.COM_STOPBIT.ONE
);

fbModbusSerialSlave
(
    xEnable     :=fbComControl1.xActive,
    hCom        := fbComControl1.hCom,
    usiSlaveId  := 1,
    pData       := ADR(awSlaveData),
    szSize      := SIZEOF(awSlaveData)
);

// данные, полученные от OPC
xVar_Opc := awSlaveData[0].0;
wVar_Opc := awSlaveData[1];
rVar_Opc := OCL.WORD2_TO_REAL(awSlaveData[2], awSlaveData[3], FALSE);
OCL.SWAP_DATA( ADR(awSlaveData[4]), ADR(sVar_Opc), SIZEOF(sVar_Opc), TRUE, FALSE, FALSE );


// по команде записываем переменные из программы в регистры Modbus Slave
fbWriteEdge(CLK := xWrite);

IF fbWriteEdge.Q THEN

    awSlaveData[0].0 := xVar_Plc;
    awSlaveData[1]   := wVar_Plc;
    fbRealToWord2(rInput := rVar_Plc, wOutput1 => awSlaveData[2], wOutput2 => awSlaveData[3]);
    OCL.SWAP_DATA( ADR(sVar_Plc), ADR(awSlaveData[4]), SIZEOF(sVar_Plc), TRUE, FALSE, FALSE );

END_IF
```

Рисунок 5.9.15 – Код программы на языке ST

При первом вызове программы происходит открытие COM-порта с заданными настройками с помощью экземпляра ФБ COM_Control. После успешного открытия порта выход **xActive** принимает и сохраняет значение **TRUE**, что приводит к вызову экземпляра ФБ MB_SerialSlave, который выполняет функцию

Modbus Slave с адресом **1** (**usiSlaveId**). Данные slave'а хранятся в массиве **awSlaveData**. В экземпляр ФБ передается указатель на этот массив (**pData**) и его размер в байтах (**szSize**).

Данные из регистров slave'а копируются в переменные программы. Для преобразования двух переменных типа **WORD** в переменную типа **REAL** используется функция WORD2_TO_REAL. Для преобразования набора переменных типа **WORD** в переменную типа **STRING** используется функция SWAP_DATA. Для обеспечения порядка байт, принятого в OPC-сервере, в процессе копирования происходит перестановка байт в регистрах (**xSwapBytes**=**TRUE**).

ПРИМЕЧАНИЕ

Для определения размера строки используется оператор **SIZEOF**, который учитывает терминирующий ноль. В рамках примера этот способ работает корректно, так как размер строки является нечетным (15 символов), и с учетом терминирующего нуля равен 16 байт (8 регистров Modbus).

По переднему фронту переменной **xWrite** происходит запись переменных программы в регистры slave'а. Для записи переменной типа **REAL** используется экземпляр ФБ REAL_TO_WORD2. Для записи переменной типа **STRING** используется функция SWAP_DATA.

7. Установить и запустить MasterOPC Universal Modbus Server.

8. Нажать **ПКМ** на узел **Server** и добавить коммуникационный узел типа **COM**. В узле следует указать сетевые настройки в соответствии с таблицей 5.9.2. Для работы OPC-сервера в режиме **Modbus RTU Master** параметры **Использовать режим ASCII** и **Slave подключение** должны иметь значение **FALSE**.

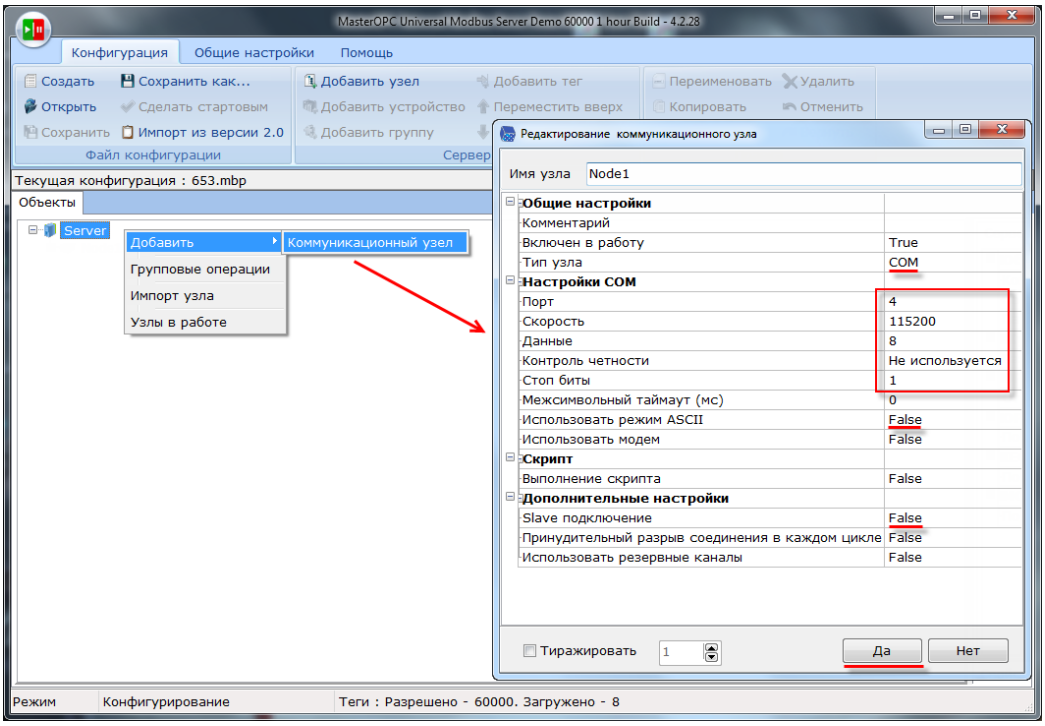


Рисунок 5.9.16 – Добавление коммуникационного узла

9. Нажать **ПКМ** на коммуникационный узел и добавить устройство с настройками по умолчанию (Slave ID = **1** в соответствии с таблицей 5.9.2).

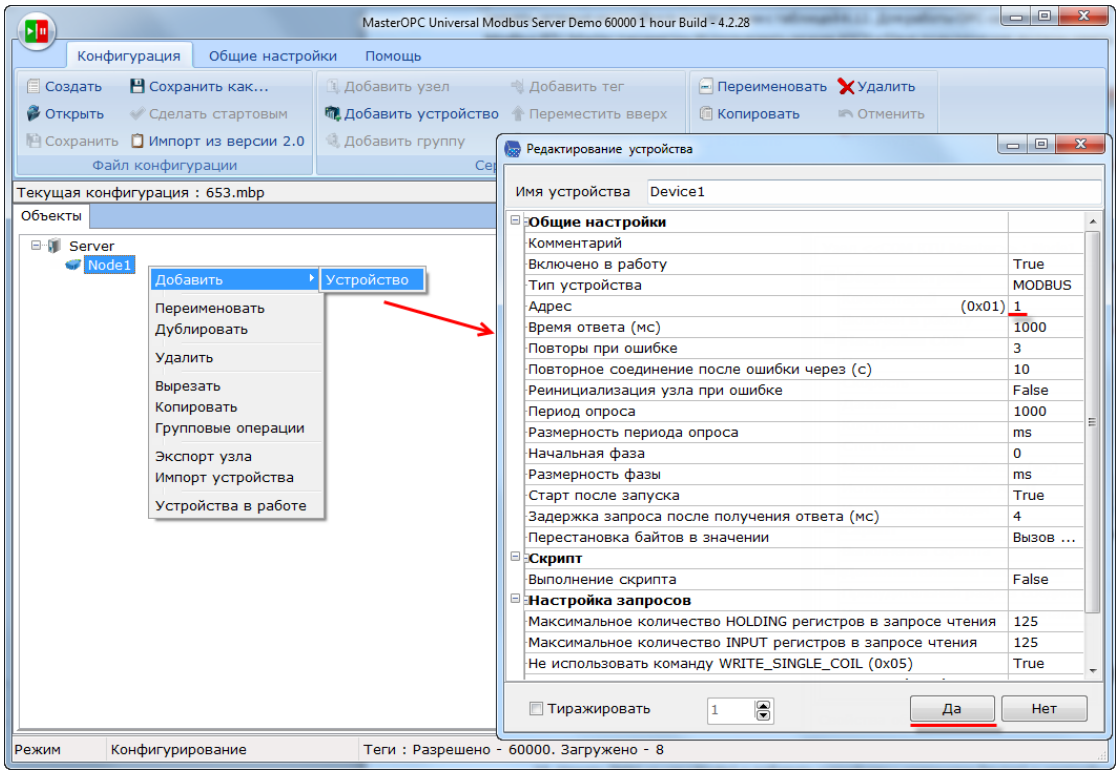


Рисунок 5.7.17 – Добавление устройства

10. Нажать **ПКМ** на устройство и добавить 4 тега. Число тегов соответствует числу переменных, считываемых/записываемых OPC-сервером. Настройки тегов приведены ниже.

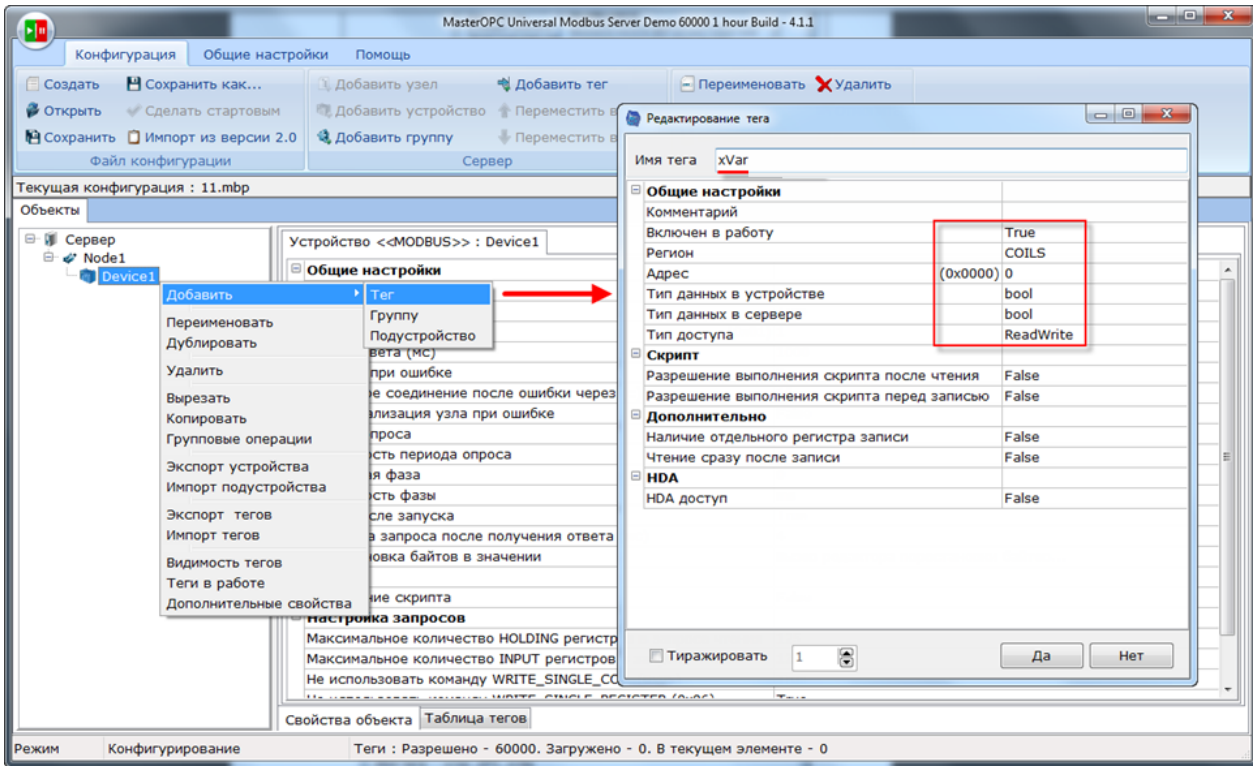
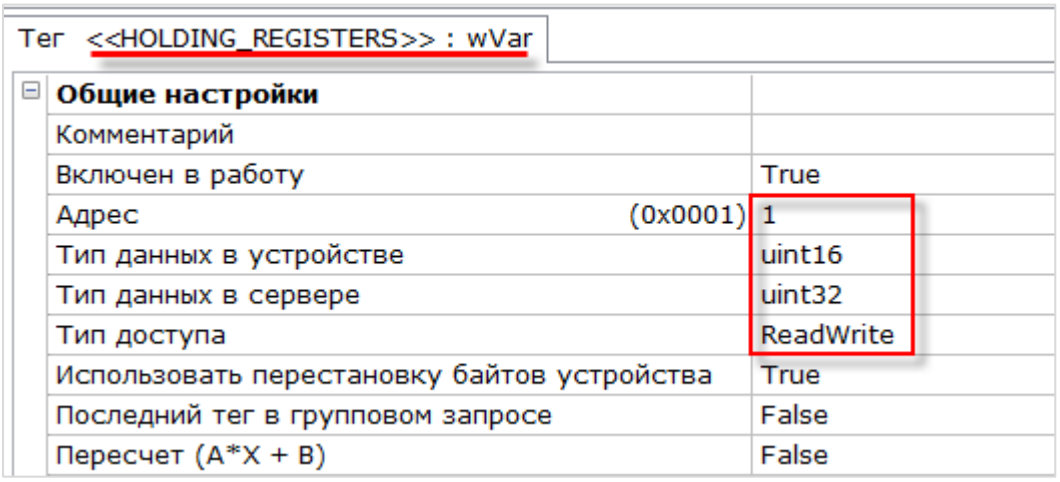
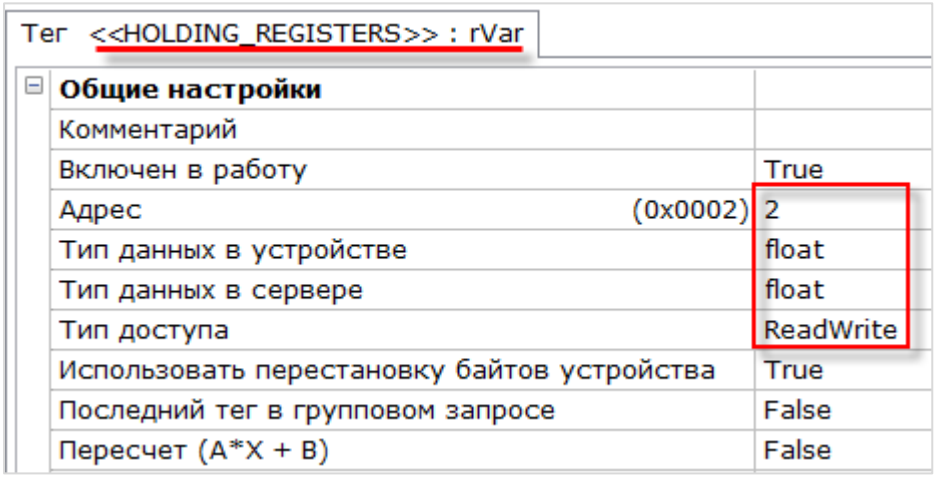


Рисунок 5.9.18 – Добавление тега xVar



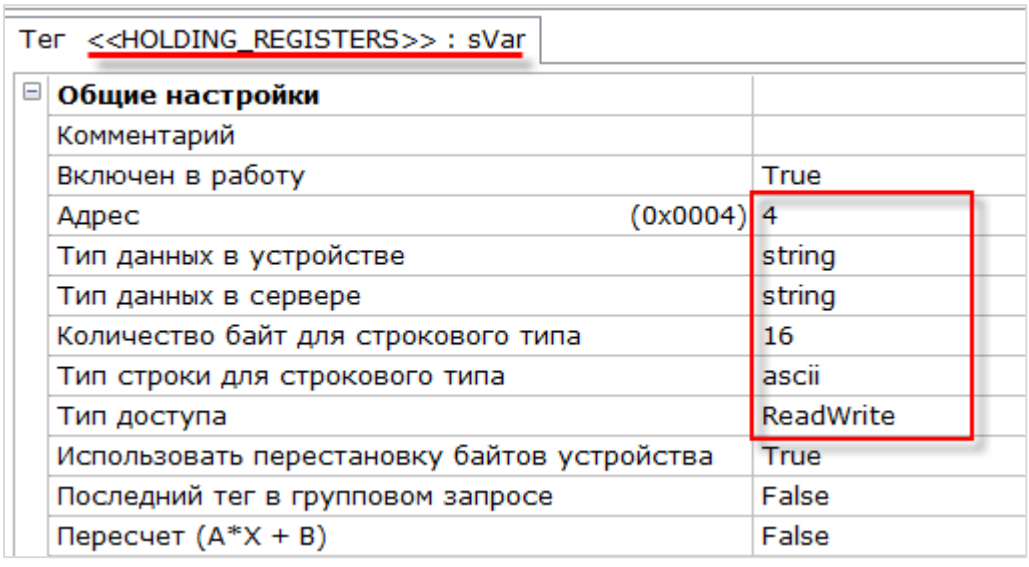
Тег <<HOLDING_REGISTERS>> : wVar		
Общие настройки		
Комментарий		
Включен в работу		True
Адрес	(0x0001)	1
Тип данных в устройстве		uint16
Тип данных в сервере		uint32
Тип доступа		ReadWrite
Использовать перестановку байтов устройства		True
Последний тег в групповом запросе		False
Пересчет (A*X + B)		False

Рисунок 5.9.19 – Добавление тега wVar



Тег <<HOLDING_REGISTERS>> : rVar		
Общие настройки		
Комментарий		
Включен в работу		True
Адрес	(0x0002)	2
Тип данных в устройстве		float
Тип данных в сервере		float
Тип доступа		ReadWrite
Использовать перестановку байтов устройства		True
Последний тег в групповом запросе		False
Пересчет (A*X + B)		False

Рисунок 5.9.20 – Добавление тега rVar



Тег <<HOLDING_REGISTERS>> : sVar		
Общие настройки		
Комментарий		
Включен в работу		True
Адрес	(0x0004)	4
Тип данных в устройстве		string
Тип данных в сервере		string
Количество байт для строкового типа		16
Тип строки для строкового типа		ascii
Тип доступа		ReadWrite
Использовать перестановку байтов устройства		True
Последний тег в групповом запросе		False
Пересчет (A*X + B)		False

Рисунок 5.9.21 – Добавление тега sVar

11. Загрузить проект в контроллер и запустить его. Запустить OPC-сервер для контроля значений переменных.

В OPC-сервере следует изменить значения тегов и наблюдать соответствующие значения в CODESYS. В CODESYS следует изменить значения **_Plc** переменных и сгенерировать импульс в переменной **xWrite** для записи значений в регистры slave'a. Записанные значения будут прочитаны OPC-сервером.

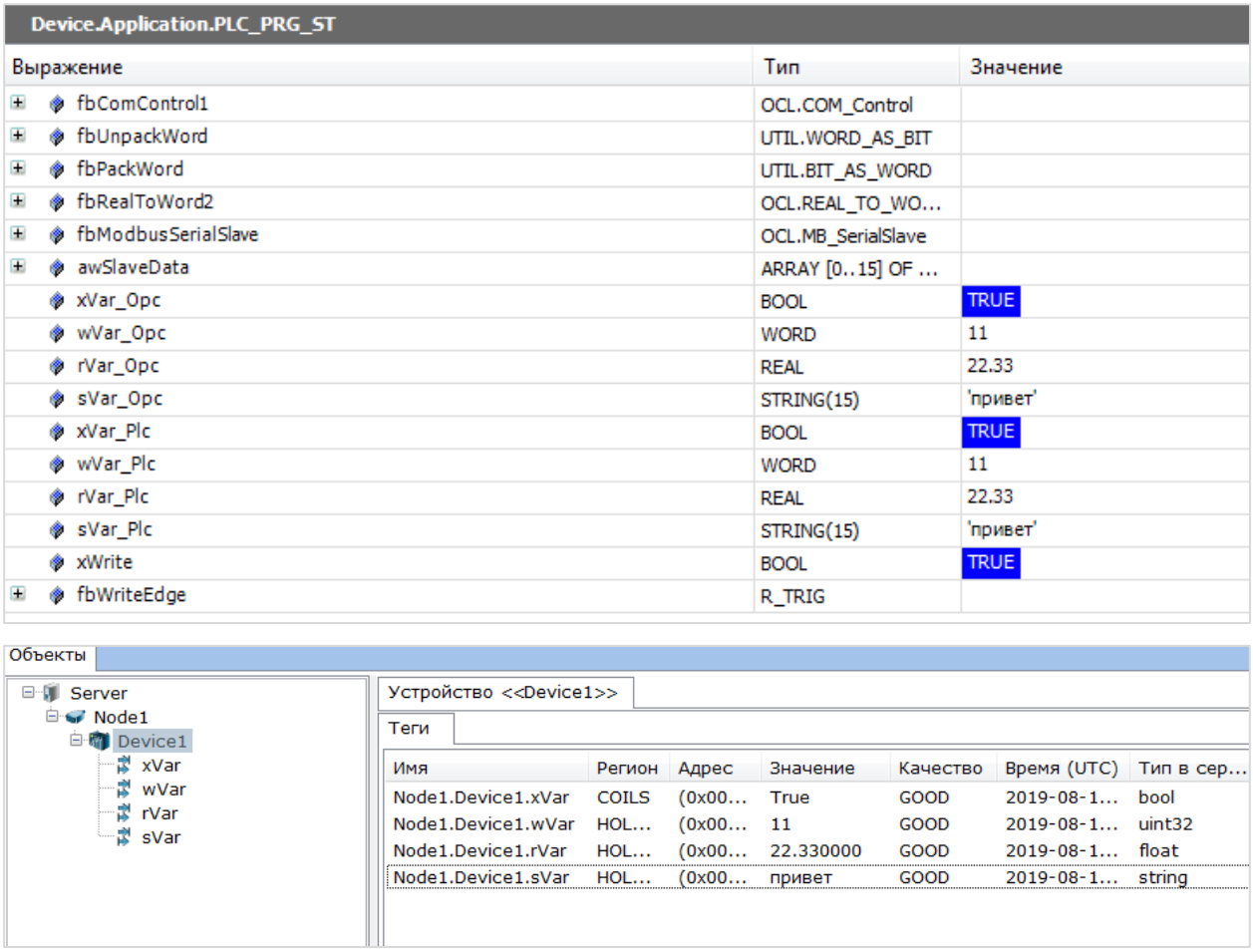


Рисунок 5.9.22 – Чтение и запись данных через OPC-сервер

5.9.3 СПК1xx [M01] (Modbus TCP Master) + модули Mx210

В качестве примера будет рассмотрена настройка обмена с модулями Mx210 (MB210-101 и МК210-301) с использованием библиотеки **OwenCommunication**. В примере используется библиотека версии **3.5.11.6**.

Реализуемый алгоритм: если значение первого аналогового входа модуля **MB210-101** превышает **30** и при этом первый дискретный вход модуля **МК210-301** имеет значение **TRUE** (замкнут), то первому дискретному выходу модуля **МК210-301** присваивается значение **TRUE** (замкнут). Во всех остальных случаях дискретному выходу присваивается значение **FALSE** (разомкнут).

Структурная схема примера приведена на рисунке ниже:

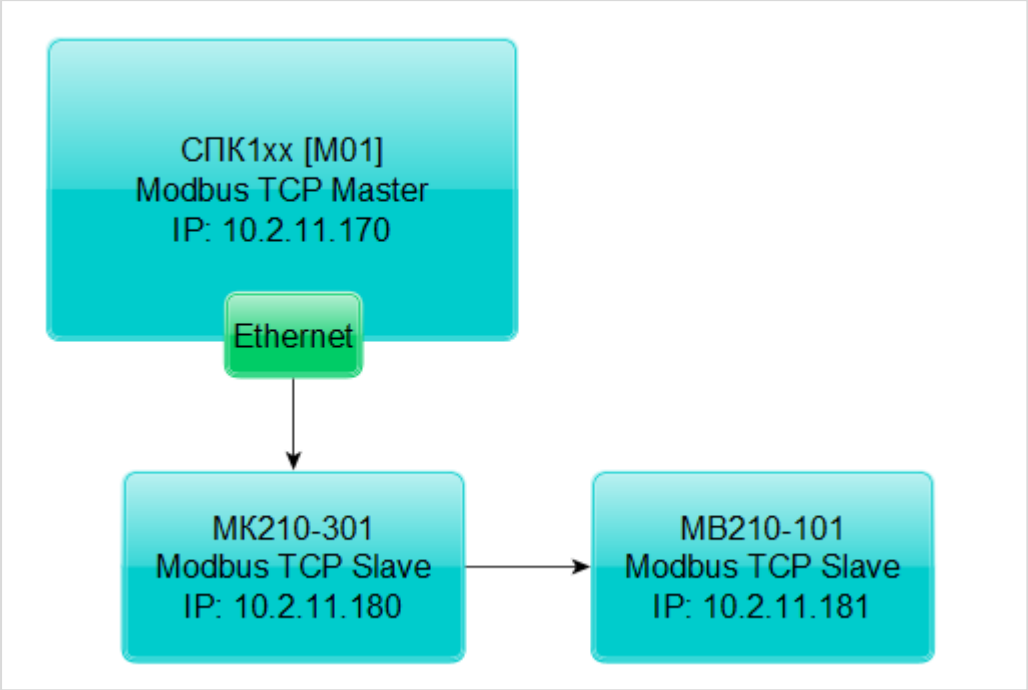


Рисунок 5.9.23 – Структурная схема примера

Пример создан в среде **CODESYS V3.5 SP17 Patch 3** и подразумевает запуск на **СПК1xx [M01]** с таргет-файлом **3.5.17.31.** 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_OwenCommunicationModbusTcpMaster_3517v1.projectarchive

Сетевые параметры устройств приведены в таблице ниже:

Таблица 5.9.4 – Сетевые параметры устройств

Параметр	СПК1xx [M01]	МК210-301	МВ210-101
Режим работы	master	slave	slave
IP-адрес	10.2.11.170	10.2.11.180	10.2.11.181
Маска подсети	255.255.0.0		
IP-адрес шлюза	10.2.1.1		
Порт	502		
Unit ID	-	1	1

Для настройки обмена следует:

1. Настроить модули **Mx210** с помощью программы **ОВЕН Конфигуратор** в соответствии с таблицей 5.9.4 (см. руководство **Mx210. Примеры настройки обмена**). Подключить модули к контроллеру.

2. Установить в CODESYS библиотеку **OwenCommunication** (см. п. 5.2).

3. Создать новый проект **CODESYS** с программой на языке **ST** или **CFC**:

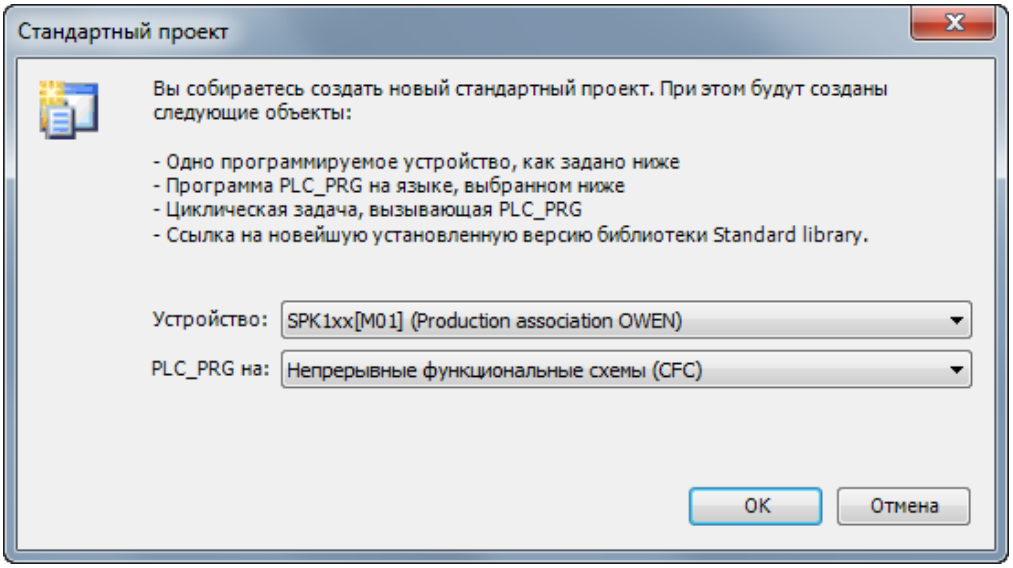


Рисунок 5.9.24 – Создание проекта CODESYS

ПРИМЕЧАНИЕ
Проект примера содержит программы на языках CFC и ST. По умолчанию используется программа на CFC (**PLC_PRG_CFC**). Для работы с программой на ST следует в конфигурации задач для задачи **MainTask** удалить вызов **PLC_PRG_CFC** и добавить вызов **PLC_PRG_ST**.

4. Добавить в проект библиотеки **OwenCommunication** и **Util** (**Менеджер библиотек – Добавить библиотеку**).

ПРИМЕЧАНИЕ
Библиотека **Util** используется только в программе на языке CFC.

5. Объявить в программе следующие переменные:

PLC_PRG_CFC

```
1   PROGRAM PLC_PRG_CFC
2   VAR
3       fbTcpClientMV210:   OCL.TCP_Client;         // ФБ TCP-подключения к модулю MB210-101
4       fbTcpClientMK210:   OCL.TCP_Client;         // ФБ TCP-подключения к модулю MK210-301
5       fbMV210_101_AI1:    OCL.MB_TcpRequest;      // ФБ опроса модуля MB210-101
6       fbMK210_301_DI:     OCL.MB_TcpRequest;      // ФБ опроса входов модуля MK210-301
7       fbMK210_301_DO:     OCL.MB_TcpRequest;      // ФБ опроса выходов модуля MK210-301
8
9       rAI1:               REAL;                   // значение 1-го входа модуля MB210-101
10      wDiMask:            WORD;                   // битовая маска входов модуля MK210-301
11      wDoMask:            WORD;                   // битовая маска выходов модуля MK210-301
12      xDi0:               BOOL;                   // значение 1-го входа модуля MK210-301
13      xDo0:               BOOL;                   // значение 1-го выхода модуля MK210-301
14
15      awAI1:              ARRAY [0..1] OF WORD;   // регистры, считанные с модуля MB210-101
16
17      fbUnpackWord:       UTIL.WORD_AS_BIT;       // ФБ распаковки битовой маски
18      fbPackWord:         UTIL.BIT_AS_WORD;       // ФБ упаковки битовой маски
19
20      iStateMV210:            INT;                // шаг опроса модуля MB210-101
21      iStateMK210:            INT;                // шаг опроса модуля MK210-301
22  END_VAR
```

Рисунок 5.9.25 – Объявление переменных в программе

ПРИМЕЧАНИЕ

Переменные **fbUnpackWord** и **fbPackWord** используются только в программе на языке CFC. Переменные **iStateMV210** и **iStateMK210** используется только в программе на языке ST.

6. Нажать **ПКМ** на программу, выбрать команду **Добавление объекта – Действие** и добавить действия с названиями **MV210_101** и **MK210_301** (язык реализации действий соответствует языку программы). В рамках примера действия используются для повышения читабельности кода.

7. Код действий и программы на языке CFC будет выглядеть следующим образом:

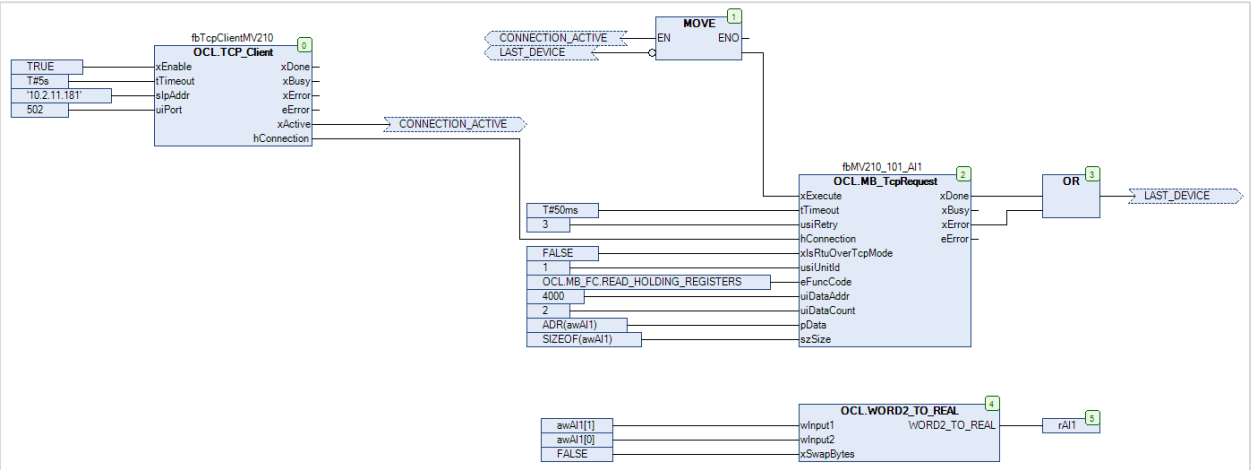


Рисунок 5.9.26 – Код действия MV210_101

При первом вызове действия происходит установка соединения с модулем по заданному IP-адресу (**sIpAddr**) и порту (**uiPort**) с помощью экземпляра ФБ TCP_Client (**блок 0**). После установки соединения выход **xActive** принимает значение **TRUE**, что приводит к вызову экземпляра ФБ MB_TcpRequest (**блок 2**), который производит опрос первого аналогового входа модуля **MB210-101** с адресом **1** (**usUnitId**). Требуемый код функции (**eFunc**), начальный адрес регистра (**uiDataAddr**) и их количество

(**uiDataCount**) приведены в РЭ на модуль. Полученные данные помещаются в переменную **awAI1** типа **ARRAY [0..1] OF WORD**. С помощью функции WORD2_TO_REAL (**блоки 4–5**) этот массив преобразуется в переменную типа **rAI1** типа **REAL**.

После завершения работы экземпляра ФБ MB_TcpRequest один из его выходов принимает значение **TRUE**: если опрос произведен успешно, то значение **TRUE** принимает выход **xDone**, если ответ не получен или получен ответ с кодом ошибки Modbus – то значение **TRUE** принимает выход **xError**. В любой из этих ситуаций с помощью **блока 1** происходит вызов экземпляра ФБ MB_TcpRequest со значением **FALSE** на входе **xExecute** (сброс блока), а потом – его следующий вызов, что приводит к началу следующего сеанса опроса.

Если соединение разорвано или экземпляр ФБ TCP_Client находится в состоянии ошибки (выход **xActive** имеет значение **FALSE**), то опрос прекращается.

ПРИМЕЧАНИЕ

В рамках примера для повышения читабельности схемы вместо некоторых линий связи использованы **метки соединения**.

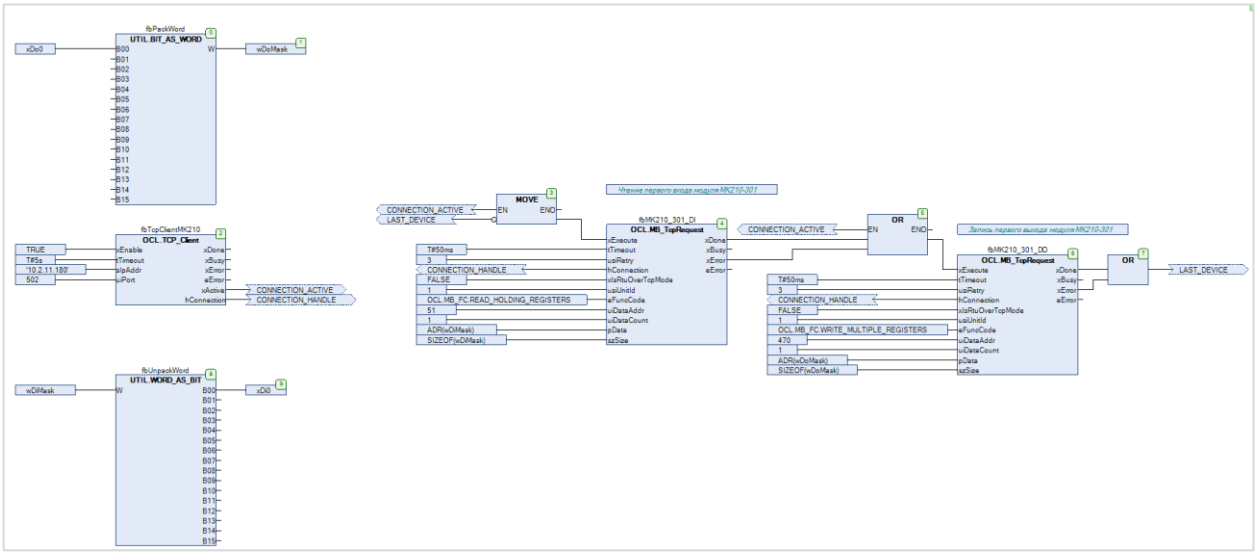


Рисунок 5.9.27 – Код действия MK210_301

При первом вызове действия происходит установка соединения с модулем по заданному IP-адресу (**sIpAddr**) и порту (**uiPort**) с помощью экземпляра ФБ TCP_Client (**блок 0**). После установки соединения выход **xActive** принимает значение **TRUE**, что приводит к вызову экземпляра ФБ MB_TcpRequest (**блок 4**), который производит опрос битовой маски дискретных входов модуля **МК210-301** с адресом **1** (**usiUnitId**). Требуемый код функции (**eFunc**), начальный адрес регистра (**uiDataAddr**) и их количество (**uiDataCount**) приведены в РЭ на модуль. Полученные данные помещаются в переменную **wDiMask** типа **WORD** и разделяются на отдельные переменные типа **BOOL** с помощью экземпляра ФБ **WORD_AS_BIT** из библиотеки **Util**.

После завершения работы ФБ MB_TcpRequest один из его выходов принимает значение **TRUE**: если опрос произведен успешно, то значение **TRUE** принимает выход **xDone**, если ответ не получен или получен ответ с кодом ошибки Modbus – то значение **TRUE** принимает выход **xError**. В любой из этих ситуаций начинается работа следующего блока MB_TcpRequest (**блоки 5–6**), который производит запись переменной **wDoMask** в качестве битовой маски дискретных выходов модуля. Требуемый код функции (**eFunc**), начальный адрес регистра (**uiDataAddr**) и их количество (**uiDataCount**) приведены в РЭ на модуль. Экземпляр блока **BIT_AS_WORD** (**блоки 0–1**) позволяет упаковать отдельные переменные типа **BOOL** в битовую маску типа **WORD**.

После окончания последнего блока MB_TcpRequest с помощью **блока 3** происходит вызов первого ФБ MB_TcpRequest со значением **FALSE** на входе **xExecute** (сброс блока), а потом – его следующий вызов, что приводит к началу следующего сеанса опроса.

Если соединение разорвано или экземпляр ФБ TCP_Client находится в состоянии ошибки (выход **xActive** имеет значение **FALSE**), то опрос прекращается.

ПРИМЕЧАНИЕ
Для переменных битовых масок используется тип **WORD** (хотя из-за небольшого числа входов/выходов подошел бы и **BYTE**), так как он соответствует по размеру регистру Modbus.

ПРИМЕЧАНИЕ
В рамках примера для повышения читабельности схемы вместо некоторых линий связи использованы **метки соединения**.

В программе **PLC_PRG_CFC** производится вызов действий **MV210_101** и **MK210_301**, а также выполнение алгоритма, описанного в условии примера.

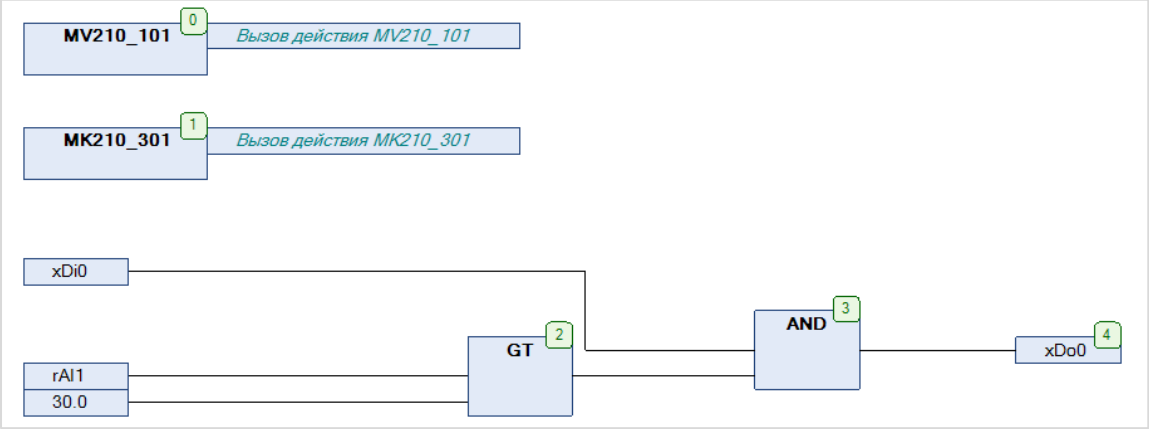
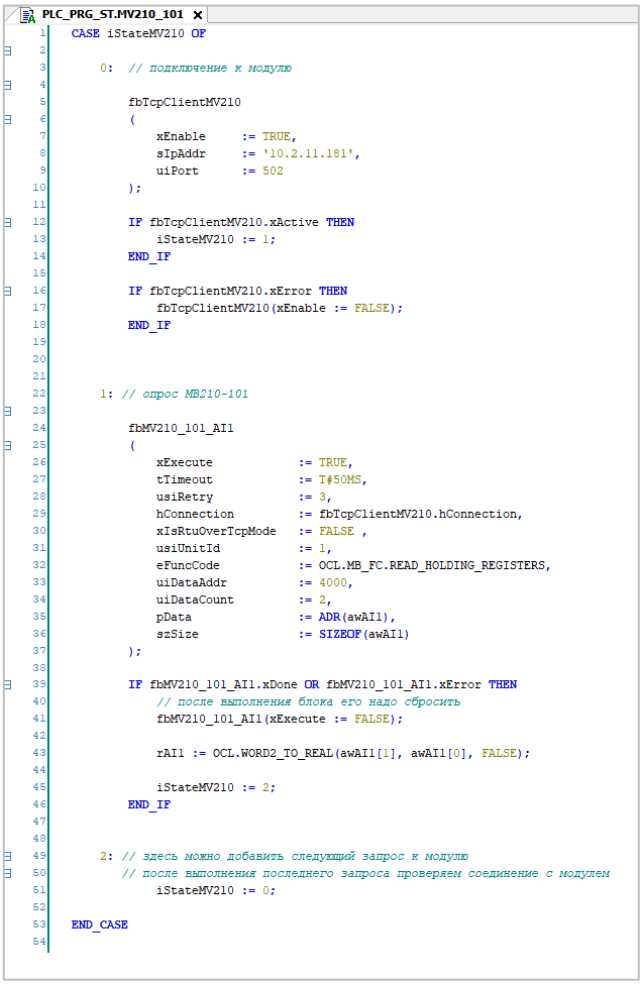


Рисунок 5.9.28 – Код программы PLC_PRG_CFC

8. Код действий и программы на языке ST будет выглядеть следующим образом:



```
PLC_PRG_ST.MV210_101
CASE iStateMV210 OF

    0:  // подключение к модулю

        fbTcpClientMV210
        (
            xEnable     := TRUE,
            sIpAddr     := '10.2.11.181',
            uiPort      := 502
        );

        IF fbTcpClientMV210.xActive THEN
            iStateMV210 := 1;
        END_IF

        IF fbTcpClientMV210.xError THEN
            fbTcpClientMV210(xEnable := FALSE);
        END_IF



    1: // опрос MB210-101

        fbMV210_101_AI1
        (
            xExecute            := TRUE,
            tTimeout            := T#50MS,
            usiRetry            := 3,
            hConnection         := fbTcpClientMV210.hConnection,
            xIsRtuOverTcpMode   := FALSE ,
            usiUnitId           := 1,
            eFuncCode           := OCL.MB_FC.READ_HOLDING_REGISTERS,
            uiDataAddr          := 4000,
            uiDataCount         := 2,
            pData               := ADR(awAI1),
            szSize              := SIZEOF(awAI1)
        );

        IF fbMV210_101_AI1.xDone OR fbMV210_101_AI1.xError THEN
            // после выполнения блока его надо сбросить
            fbMV210_101_AI1(xExecute := FALSE);

            rAI1 := OCL.WORD2_TO_REAL(awAI1[1], awAI1[0], FALSE);

            iStateMV210 := 2;
        END_IF


    2: // здесь можно добавить следующий запрос к модулю
       // после выполнения последнего запроса проверяем соединение с модулем
            iStateMV210 := 0;

END_CASE
```

Рисунок 5.9.29 – Код действия MV210_101

При первом вызове действия в **шаге 0** происходит установка соединения с модулем по заданному IP-адресу (**sIpAddr**) и порту (**uiPort**) с помощью экземпляра ФБ TCP_Client (**блок 0**). После установки соединения выход **xActive** принимает значение **TRUE**, что приводит к переходу на **шаг 1**. Если соединение не удается установить в течение заданного времени или при установке соединения происходит ошибка, то выполняется новая попытка соединения.

На **шаге 1** выполняется вызов экземпляра ФБ MB_TcpRequest, который производит опрос первого аналогового входа модуля **MB210-101** с адресом **1** (**usiSlaveId**). Требуемый код функции (**eFunc**), начальный адрес регистра (**uiDataAddr**) и их количество (**uiDataCount**) приведены в РЭ на модуль. Полученные данные помещаются в переменную **awAI1** типа **ARRAY [0..1] OF WORD**. С помощью функции WORD2_TO_REAL этот массив преобразуется в переменную типа **rAI1** типа **REAL**.

После завершения работы экземпляра ФБ MB_TcpRequest один из его выходов принимает значение **TRUE**: если опрос произведен успешно, то значение **TRUE** принимает выход **xDone**, если ответ не получен или получен ответ с кодом ошибки Modbus – то значение **TRUE** принимает выход **xError**.

В любой из этих ситуаций происходит вызов экземпляра ФБ MB_TcpRequest со значением **FALSE** на входе **xExecute** (сброс блока) и переход на **шаг 2**.

В рамках примера на **шаге 2** происходит переход на **шаг 0**, что приводит к проверке состояния соединения и началу следующего сеанса опроса. В случае опроса нескольких параметров устройства число шагов можно увеличить. После выполнения последнего шага должен происходить переход на шаг опроса первого устройства.

PLC_PRG_ST.MK210_301

```
CASE iStateMK210 OF

    0:  // подключение к модулю

        fbTcpClientMK210
        (
            xEnable     := TRUE,
            tTimeout    := T#10S,
            sIpAddr     := '10.2.11.180',
            uiPort      := 502
        );

        IF fbTcpClientMK210.xActive THEN
            iStateMK210 := 1;
        END_IF

        IF fbTcpClientMK210.xError THEN
            fbTcpClientMK210(xEnable := FALSE);
        END_IF

    1:  // опрос дискретных входов модуля MK210-301

        fbMK210_301_DI
        (
            xExecute            := TRUE,
            tTimeout            := T#50MS,
            usiRetry            := 3,
            hConnection         := fbTcpClientMK210.hConnection,
            xIsRtuOverTcpMode   := FALSE ,
            usiUnitId           := 1,
            eFuncCode           := OCL.MB_FC.READ_HOLDING_REGISTERS,
            uiDataAddr          := 51,
            uiDataCount         := 1,
            pData               := ADR(wDiMask),
            szSize              := SIZEOF(wDiMask)
        );

        IF fbMK210_301_DI.xDone OR fbMK210_301_DI.xError THEN
            // после выполнения блока его надо сбросить
            fbMK210_301_DI(xExecute := FALSE);

            xDi0 := wDiMask.0;

            iStateMK210 := 2;
        END_IF

    2:  // запись дискретных выходов модуля MK210-301

        wDoMask.0 := xDo0;

        fbMK210_301_DO
        (
            xExecute            := TRUE,
            tTimeout            := T#50MS,
            usiRetry            := 3,
            hConnection         := fbTcpClientMK210.hConnection,
            xIsRtuOverTcpMode   := FALSE ,
            usiUnitId           := 1,
            eFuncCode           := OCL.MB_FC.WRITE_MULTIPLE_REGISTERS,
            uiDataAddr          := 470,
            uiDataCount         := 1,
            pData               := ADR(wDoMask),
            szSize              := SIZEOF(wDoMask)
        );

        IF fbMK210_301_DO.xDone OR fbMK210_301_DO.xError THEN
            // после выполнения блока его надо сбросить
            fbMK210_301_DO(xExecute := FALSE);

            // перед следующим сеансом опроса проверяем соединение с модулем
            iStateMK210 := 0;
        END_IF

END_CASE
```

Рисунок 5.9.30 – Код действия MK210_301

При первом вызове действия в **шаге 0** происходит установка соединения с модулем по заданному IP-адресу (**sIpAddr**) и порту (**uiPort**) с помощью экземпляра ФБ TCP_Client (**блок 0**). После установки соединения выход **xActive** принимает значение **TRUE**, что приводит к переходу на **шаг 1**. Если соединение не удается установить в течение заданного времени или при установке соединения происходит ошибка, то выполняется новая попытка соединения.

На **шаге 1** выполняется вызов экземпляра ФБ MB_TcpRequest, который производит опрос битовой маски дискретных входов модуля **МК210-301** с адресом **1** (**usiSlaveId**). Требуемый код функции (**eFunc**), начальный адрес регистра (**uiDataAddr**) и их количество (**uiDataCount**) приведены в РЭ на модуль. Полученные данные помещаются в переменную **wDiMask** типа **WORD**, из которой происходит копирование данных в нужные переменные типа **BOOL** с помощью побитового доступа (*переменная.номер_бита*).

После завершения работы экземпляра ФБ MB_TcpRequest один из его выходов принимает значение **TRUE**: если опрос произведен успешно, то значение **TRUE** принимает выход **xDone**, если ответ не получен или получен ответ с кодом ошибки Modbus – то значение **TRUE** принимает выход **xError**. В любой из этих ситуаций происходит вызов экземпляра ФБ MB_TcpRequest со значением **FALSE** на входе **xExecute** (сброс блока) и переход на **шаг 2**.

На **шаге 2** выполняется вызов экземпляра ФБ MB_TcpRequest, который производит запись переменной **wDoMask** в качестве битовой маски дискретных выходов модуля. Требуемый код функции (**eFunc**), начальный адрес регистра (**uiDataAddr**) и их количество (**uiDataCount**) приведены в РЭ на модуль. Битовая маска может формироваться из отдельных переменных типа **BOOL** с помощью побитового доступа.

В рамках примера на **шаге 2** происходит переход на **шаг 0**, что приводит к проверке соединения и началу следующего сеанса опроса. В случае опроса нескольких параметров устройства число шагов можно увеличить. После выполнения последнего шага должен происходить переход на шаг опроса первого устройства.

Если соединение разорвано или экземпляр ФБ TCP_Client находится в состоянии ошибки (выход **xActive** имеет значение **FALSE**), то опрос прекращается.

В программе **PLC_PRG_ST** производится вызов действий **MV210_101** и **МК210_301**, а также выполнение алгоритма, описанного в условии примера.

```
1  // чтобы запустить пример на ST на контроллере требуется:
2  // 1. Удалить из задачи MainTask вызов программы PLC_PRG_CFC
3  // 2. Привязать к задаче MainTask вызов программы PLC_PRG_ST
4
5  MV210_101();
6  МК210_301();
7
8  xDo0 := xDi0 AND (rAI1 > 30.0);
```

Рисунок 5.9.31 – Код программы PLC_PRG_ST

9. Загрузить проект в контроллер и запустить его.

В переменной **rAI1** будет отображаться текущее значение первого аналогового входа модуля **MB210-101**. В переменной **xDi0** будет отображаться текущее значение первого дискретного входа модуля **МК210-301**.

Если значение **rAI1** превысит **30** и при этом значение **xDi0** будет равно **TRUE**, то в переменную **xDo0** будет записано значение **TRUE,** что приведет к замыканию первого дискретного выхода модуля **МК210-301**. Если одно из условий перестанет выполняться, то выход будет разомкнут.

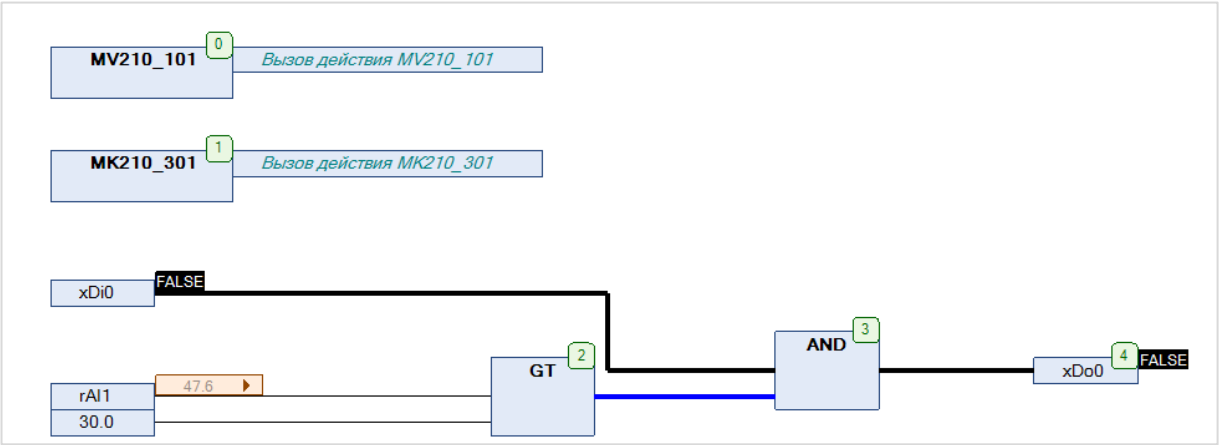


Рисунок 5.9.32 – Выполнение программы в режиме Online

5.9.4 СПК1xx [M01] (Modbus TCP Slave) + MasterOPC Universal Modbus Server

В качестве примера будет рассмотрена настройка обмена с OPC-сервером Insat MasterOPC Universal Modbus Server, который будет использоваться в режиме **Modbus TCP Master**, с помощью библиотеки **OwenCommunication**. В примере используется библиотека версии **3.5.11.6**.

Структурная схема примера приведена на рисунке ниже:

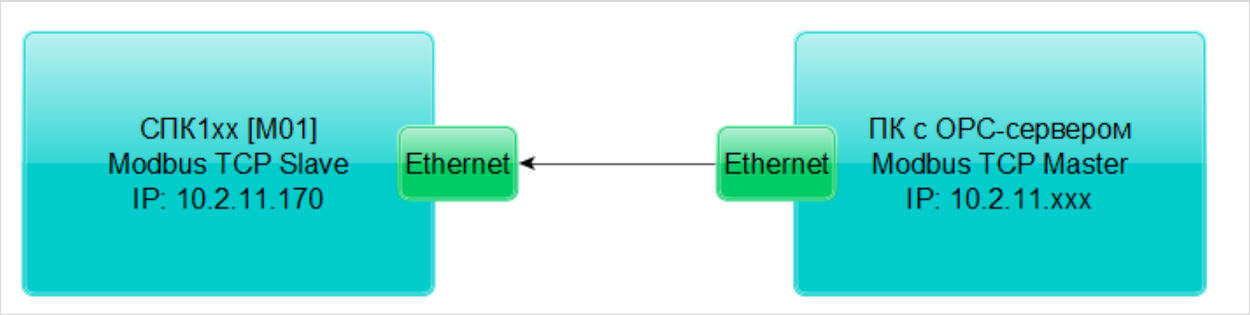


Рисунок 5.9.33 – Структурная схема примера

Пример создан в среде **CODESYS V3.5 SP17 Patch 3** и подразумевает запуск на **СПК1xx [M01]** с таргет-файлом **3.5.17.31.** 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_OwenCommunicationModbusTcpSlave_3517v1.zip

Сетевые параметры модулей приведены в таблице ниже:

Таблица 5.9.5 – Сетевые параметры устройств

Параметр	СПК1xx [M01]	ПК с OPC-сервером
IP-адрес	10.2.11.170	*любой адрес из сети, к которой подключен контроллер*
Порт	502	
Адрес (Unit ID)	1	-
Режим работы	slave	master

Переменные примера описаны в таблице ниже:

Таблица 5.9.6 – Список переменных примера

Имя	Тип	Область памяти Modbus	Адрес регистра/бита
xVar_Opc	BOOL	Coils	0/0
wVar_Opc	WORD	Holding регистры	1
rVar_Opc	REAL		2–3
sVar_Opc	STRING(15)		4–11

ПРИМЕЧАНИЕ

В рамках примера значения переменных slave'а могут быть изменены как из OPC, так и из программы контроллера (с помощью переменных с постфиксом _**Plc**).

Для настройки обмена следует:

1. Подключить контроллер и ПК к общей локальной сети.

2. Создать новый проект **CODESYS** с программой на языке **ST** или **CFC**:

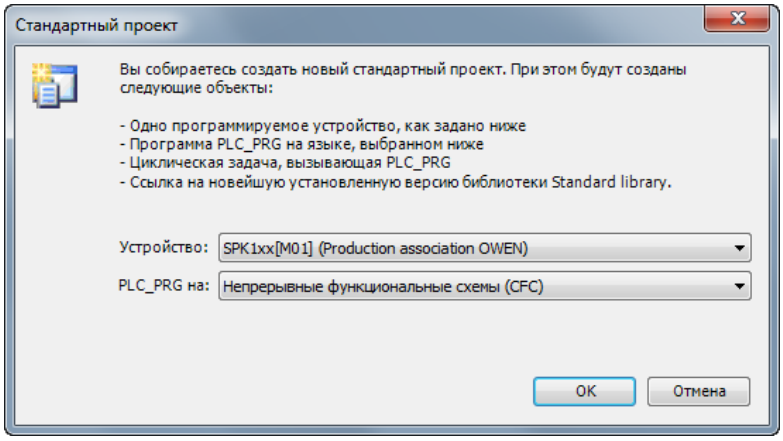


Рисунок 5.9.12 – Создание проекта CODESYS

ПРИМЕЧАНИЕ
Проект примера содержит программы на языках CFC и ST. По умолчанию используется программа на CFC (**PLC_PRG_CFC**). Для работы с программой на ST следует в конфигурации задач для задачи **MainTask** удалить вызов **PLC_PRG_CFC** и добавить вызов **PLC_PRG_ST**.

3. Добавить в проект библиотеки **OwenCommunication** и **Util** (**Менеджер библиотек – Добавить библиотеку**).

ПРИМЕЧАНИЕ
Библиотека **Util** используется только в программе на языке CFC.

4. Объявить в программе следующие переменные:

```
PROGRAM PLC_PRG_ST
VAR

    fbUnpackWord:           UTIL.WORD_AS_BIT;       // ФБ распаковки битовой маски
    fbPackWord:             UTIL.BIT_AS_WORD;       // ФБ упаковки битовой маски
    fbRealToWord2:          OCL.REAL_TO_WORD2;      // ФБ преобразования REAL в две переменные типа WORD


    fbModbusTcpSlave:       OCL.MB_TcpSlave;        // ФБ для реализации Modbus Slave

    awSlaveData:            ARRAY [0..15] OF WORD;  // буфер данных Modbus Slave

    (* значения, полученные от OPC *)
    xVar_Opc:               BOOL;
    wVar_Opc:               WORD;
    rVar_Opc:               REAL;
    sVar_Opc:               STRING(15);

    (* значения для передачи в OPC *)
    xVar_Plc:               BOOL;
    wVar_Plc:               WORD;
    rVar_Plc:               REAL;
    sVar_Plc:               STRING(15);

    xWrite:                 BOOL;                   // команда записи данных из программы в регистры Modbus Slave
    fbWriteEdge:            R_TRIG;                 // триггер для однократной записи
END_VAR
```

Рисунок 5.9.34 – Объявление переменных программы

ПРИМЕЧАНИЕ
Переменные **fbUnpackWord** и **fbPackWord** используются только в программе на языке CFC.

5. Код программы на языке CFC будет выглядеть следующим образом:

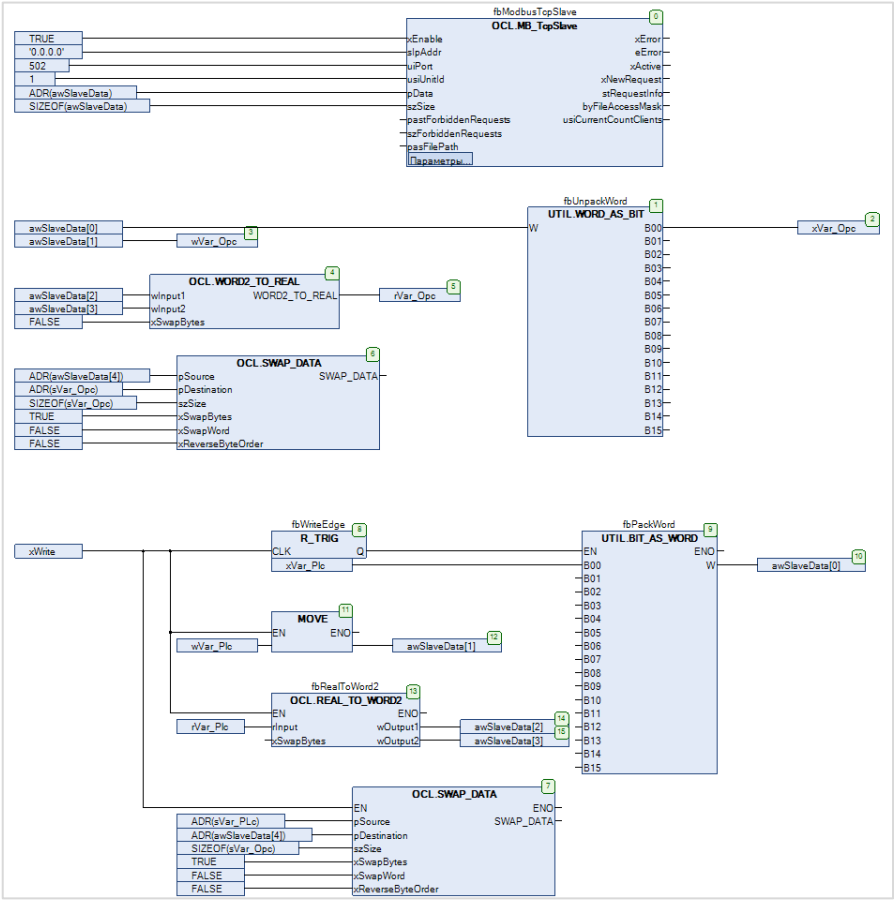


Рисунок 5.9.35 – Код программы на языке CFC

При запуске программы происходит вызов экземпляра ФБ MB_TcpSlave (**блок 0**), который выполняет функцию Modbus Slave. В качестве IP-адреса (**slpAddr**), на котором используется slave, применяется специальный адрес 0.0.0.0, который соответствует адресам всех интерфейсов контроллера (то есть slave доступен по всем TCP/IP интерфейсам). Значения входов **uiPort** и **usiUnitId** соответствуют таблице 5.9.5.

Данные slave'а хранятся в массиве **awSlaveData**. В экземпляр ФБ передается указатель на этот массив (**pData**) и его размер в байтах (**szSize**).

В **блоках 1–6** происходит копирование данных из регистров slave'а в переменные программы. Для выделения переменных типа **BOOL** из переменной типа **WORD** используется экземпляр ФБ **WORD_AS_BIT** из библиотеки **Util**. Для преобразования двух переменных типа **WORD** в переменную типа **REAL** используется функция WORD2_TO_REAL. Для преобразования набора переменных типа **WORD** в переменную типа **STRING** используется функция SWAP_DATA. Для обеспечения порядка байт, принятого в OPC-сервере, в процессе копирования происходит перестановка байт в регистрах (**xSwapBytes**=**TRUE**).

ПРИМЕЧАНИЕ

Для определения размера строки используется оператор **SIZEOF**, который учитывает терминирующий ноль. В рамках примера этот способ работает корректно, так как размер строки является нечетным (15 символов), и с учетом терминирующего нуля равен 16 байт (8 регистров Modbus).

По переднему фронту переменной **xWrite** (**блок 8**) происходит запись переменных программы в регистры slave'a. Для записи переменной типа **BOOL** используется экземпляр ФБ **BIT_AS_WORD** из библиотеки **Util**. Для записи переменной типа **REAL** используется экземпляр ФБ REAL_TO_WORD2. Для записи переменной типа **STRING** используется функция SWAP_DATA.

6. Код программы на языке ST будет выглядеть следующим образом:

```
// чтобы запустить пример на ST на контроллере требуется:
// 1. Удалить из задачи MainTask вызов программы PLC_PRG_CFC
// 2. Привязать к задаче MainTask вызов программы PLC_PRG_ST


fbModbusTcpSlave
(
    xEnable     := TRUE,
    sIpAddr     := '0.0.0.0',
    uiPort      := 502,
    usiUnitId   := 1,
    pData       := ADR(awSlaveData),
    szSize      := SIZEOF(awSlaveData)
);

// данные, полученные от OPC
xVar_Opc := awSlaveData[0].0;
wVar_Opc := awSlaveData[1];
rVar_Opc := OCL.WORD2_TO_REAL(awSlaveData[2], awSlaveData[3], FALSE);
OCL.SWAP_DATA( ADR(awSlaveData[4]), ADR(sVar_Opc), SIZEOF(sVar_Opc), TRUE, FALSE, FALSE );


// по команде записываем переменные из программы в регистры Modbus Slave
fbWriteEdge(CLK := xWrite);

IF fbWriteEdge.Q THEN

    awSlaveData[0].0 := xVar_Plc;
    awSlaveData[1]   := wVar_Plc;
    fbRealToWord2(rInput := rVar_Plc, wOutput1 => awSlaveData[2], wOutput2 => awSlaveData[3]);
    OCL.SWAP_DATA( ADR(sVar_Plc), ADR(awSlaveData[4]), SIZEOF(sVar_Plc), TRUE, FALSE, FALSE );

END_IF
```

Рисунок 5.9.36 – Код программы на языке ST

При запуске программы происходит вызов экземпляра ФБ MB_TcpSlave (**блок 0**), который выполняет функцию Modbus Slave. В качестве IP-адреса (**sIpAddr**), на котором используется slave, применяется специальный адрес 0.0.0.0, который соответствует адресам всех интерфейсов контроллера (то есть slave доступен по всем TCP/IP интерфейсам). Значения входов **uiPort** и **usiUnitId** соответствуют таблице 5.9.5.

Данные из регистров slave'a копируются в переменные программы. Для преобразования двух переменных типа **WORD** в переменную типа **REAL** используется функция WORD2_TO_REAL. Для преобразования набора переменных типа **WORD** в переменную типа **STRING** используется функция SWAP_DATA. Для обеспечения порядка байт, принятого в OPC-сервере, в процессе копирования происходит перестановка байт в регистрах (**xSwapBytes**=**TRUE**).

ПРИМЕЧАНИЕ

Для определения размера строки используется оператор **SIZEOF**, который учитывает терминирующий ноль. В рамках примера этот способ работает корректно, так как размер строки является нечетным (15 символов), и с учетом терминирующего нуля равен 16 байт (8 регистров Modbus).

По переднему фронту переменной **xWrite** происходит запись переменных программы в регистры slave'а. Для записи переменной типа **REAL** используется экземпляр ФБ REAL_TO_WORD2. Для записи переменной типа **STRING** используется функция SWAP_DATA.

7. Установить и запустить MasterOPC Universal Modbus Server.

8. Нажать **ПКМ** на узел **Server** и добавить коммуникационный узел типа **TCP/IP**. В узле следует указать сетевые настройки в соответствии с таблицей 5.9.5. Для работы OPC-сервера в режиме **Modbus TCP Master** параметры **Modbus поверх TCP** и **Slave подключение** должны иметь значение **FALSE**.

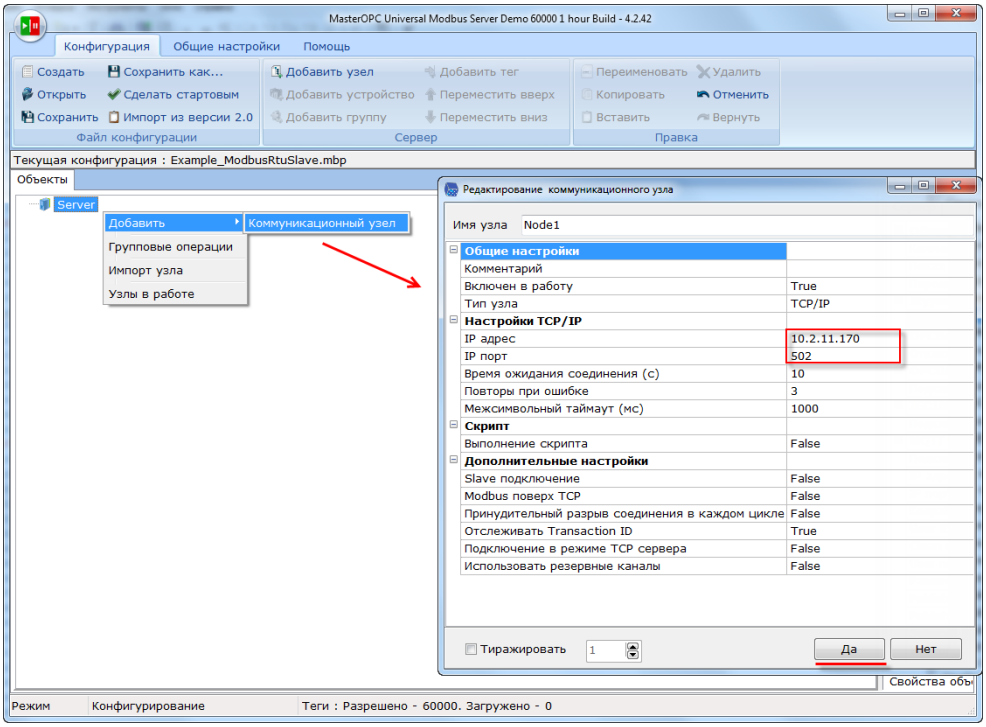


Рисунок 5.9.37 – Добавление коммуникационного узла

9. Нажать **ПКМ** на коммуникационный узел и добавить устройство с настройками по умолчанию.

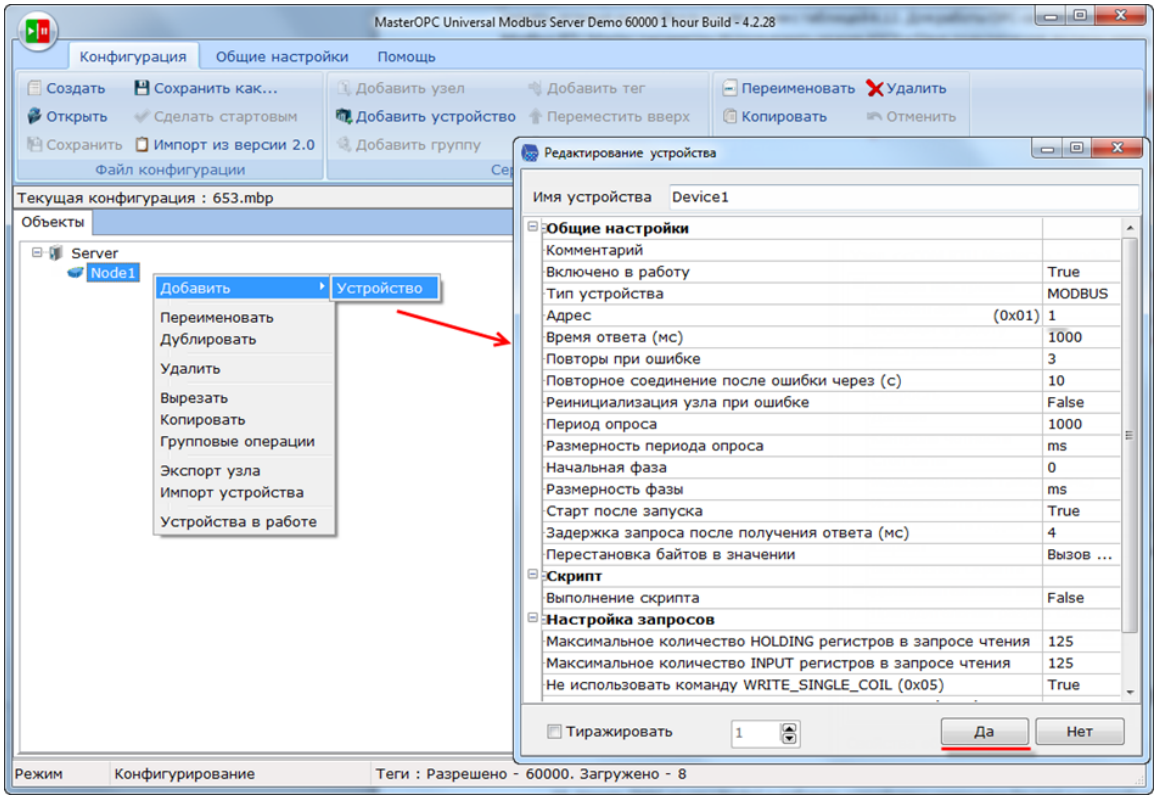


Рисунок 5.9.38 – Добавление устройства

10. Нажать **ПКМ** на устройство и добавить 4 тега. Число тегов соответствует числу переменных, считываемых/записываемых OPC-сервером. Настройки тегов приведены ниже.

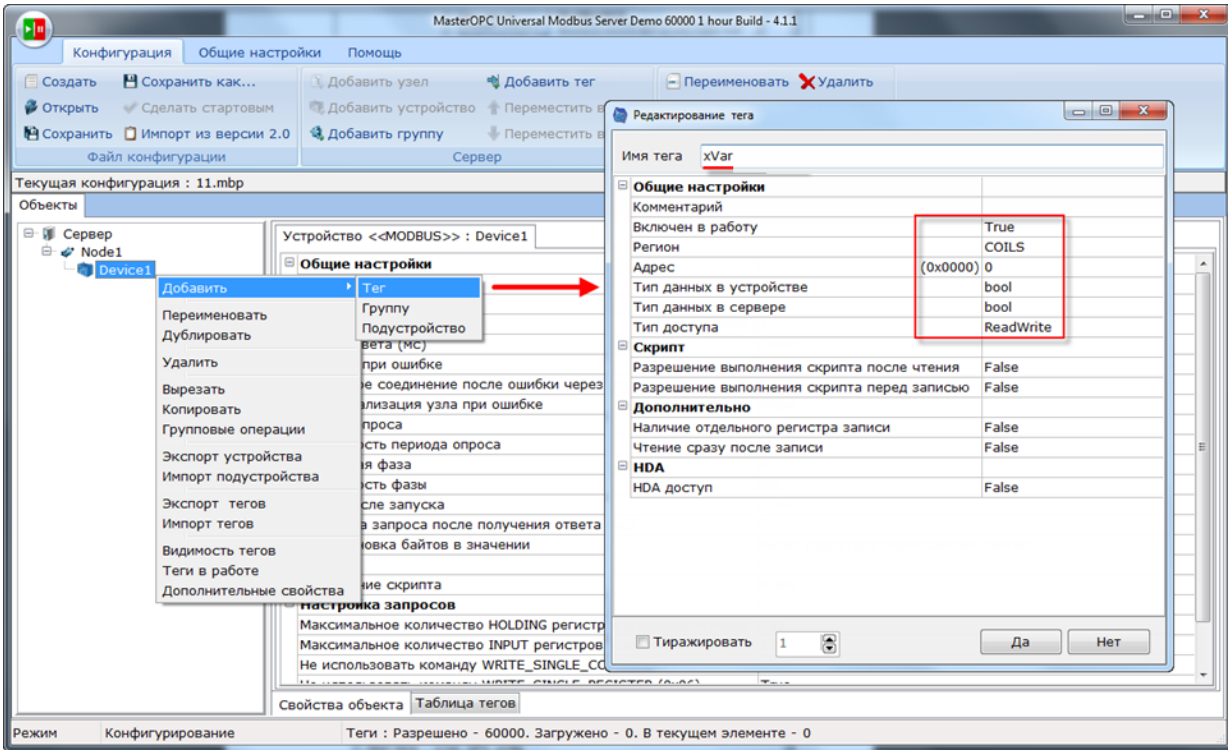


Рисунок 5.9.39 – Добавление тега xVar

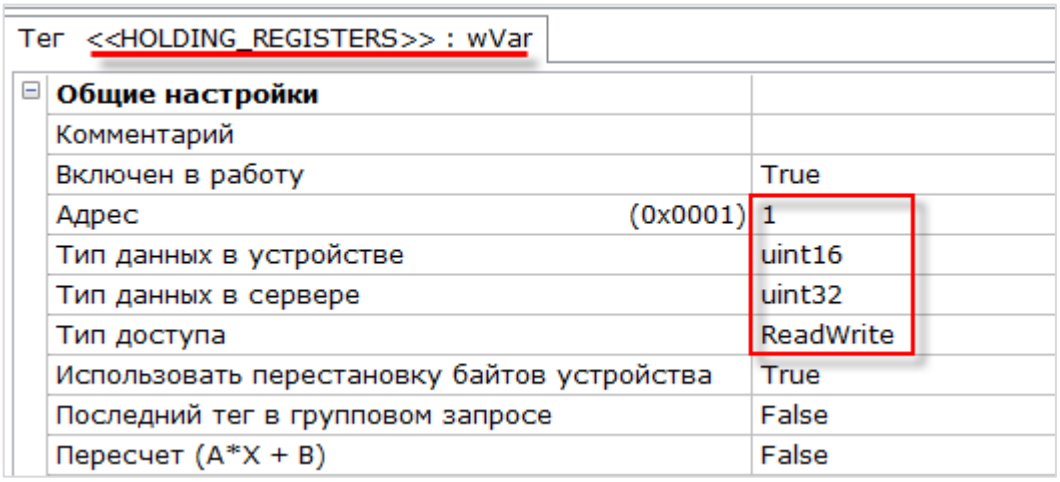


Рисунок 5.9.40 – Добавление тега wVar

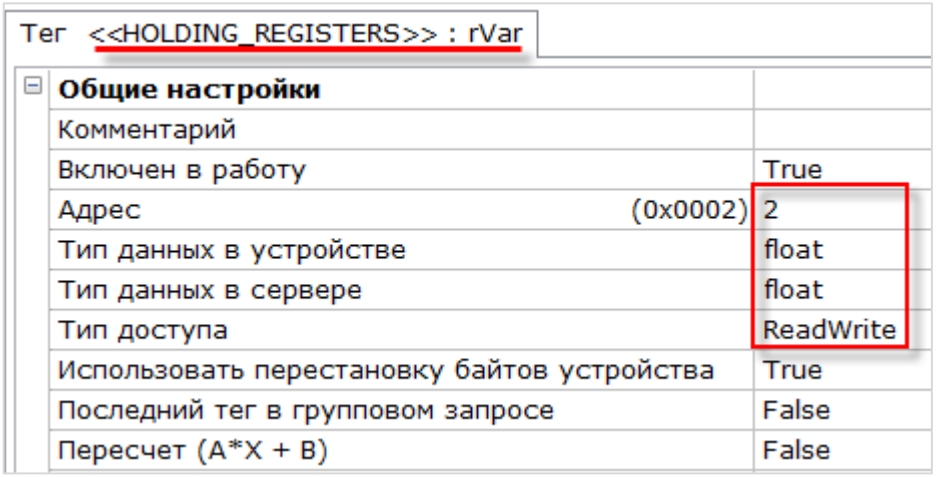


Рисунок 5.9.41 – Добавление тега rVar

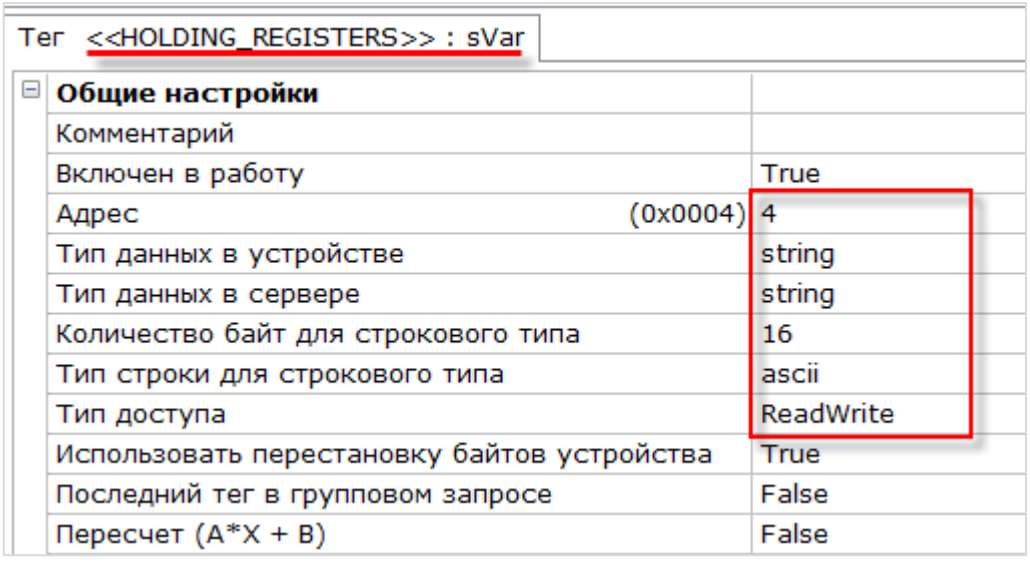


Рисунок 5.9.42 – Добавление тега sVar

11. Загрузить проект в контроллер и запустить его. Запустить OPC-сервер для контроля значений переменных.

В OPC-сервере следует изменить значения тегов и наблюдать соответствующие значения в CODESYS. В CODESYS следует изменить значения **_Plc** переменных и сгенерировать импульс в переменной **xWrite** для записи значений в регистры slave'а. Записанные значения будут прочитаны OPC-сервером.

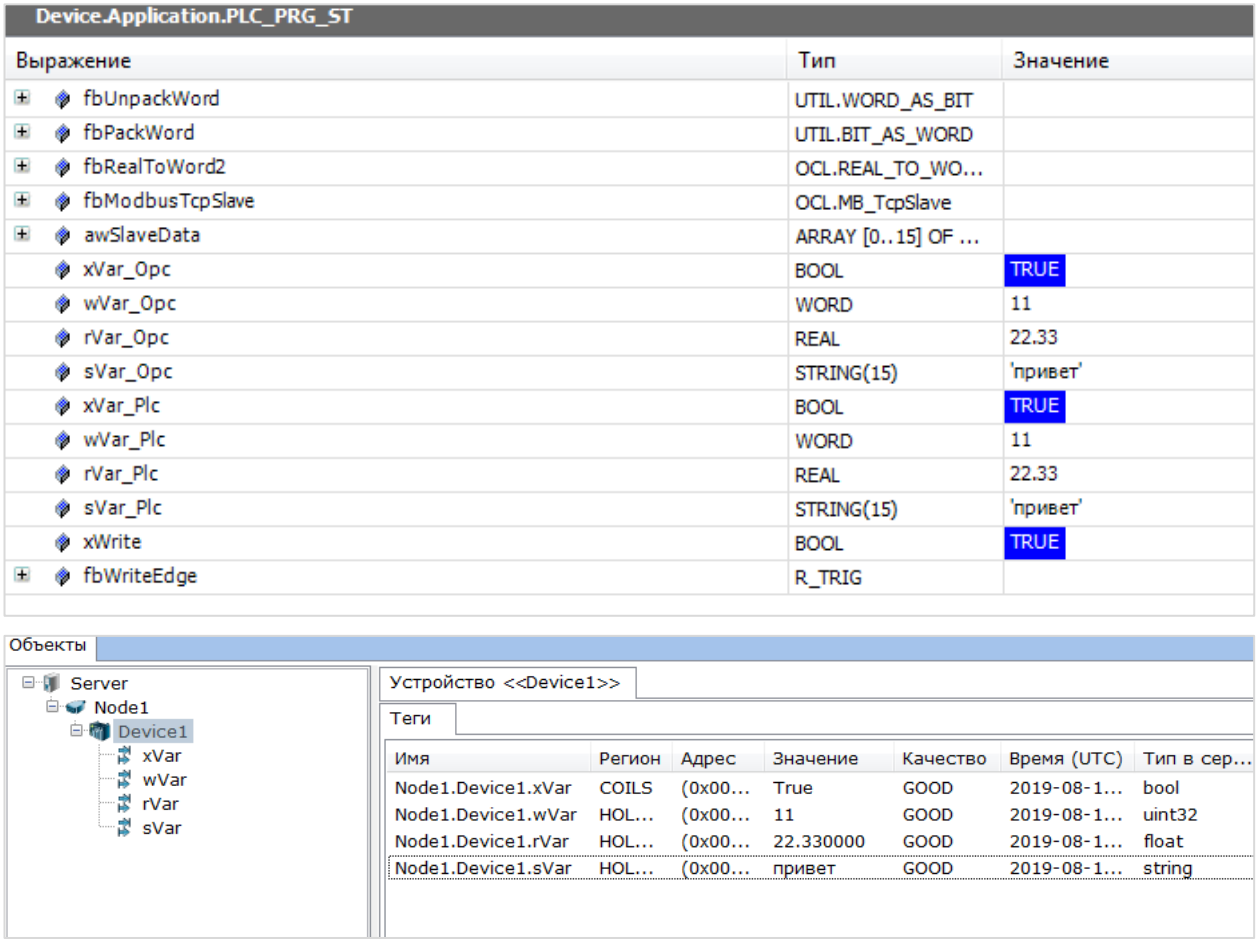


Рисунок 5.9.43 – Считывание и запись данных через OPC-сервер

5.9.5 СПК1xx [M01] (Modbus TCP Slave) – чтение файлов с помощью 20 функции Modbus

Функциональный блок MB_TcpSlave поддерживает 20 функцию Modbus (**Read File Record**), что позволяет считывать файлы контроллера. Устройство, которое является Modbus TCP Master'ом, должно также поддерживать эту функцию.

В качестве примера будет рассмотрено считывание файлов с контроллера OPC-сервером Insat MasterOPC Universal Modbus Server, который поддерживает 20 функцию Modbus. Считанные из файлов значения могут быть переданы в SCADA-систему с помощью технологии **OPC HDA**.

Данный OPC-сервер включает скрипт, который позволяет считывать только файлы определенного формата (формата архива, поддержанного в контроллерах ОВЕН ПЛК1xx, режим ASCII). Описание этого формата доступно по ссылке. Пользователь может отредактировать скрипт для поддержки другого формата. Скрипт имеет название **OwenPlcHDA.lua** и расположен в директории **C:\ProgramData\InSAT\MasterOPC Universal Modbus Server\MODULES**. Описание принципов работы со скриптами OPC-сервера доступно по ссылке.

В рамках примера считывается файл **Arc1.log** формата архива ПЛК1xx. Этот файл включен в состав файла примера. Файл включает в себя архив трех переменных с типами **UINT**, **UDINT** и **REAL** соответственно.

Структурная схема примера приведена на рисунке ниже:

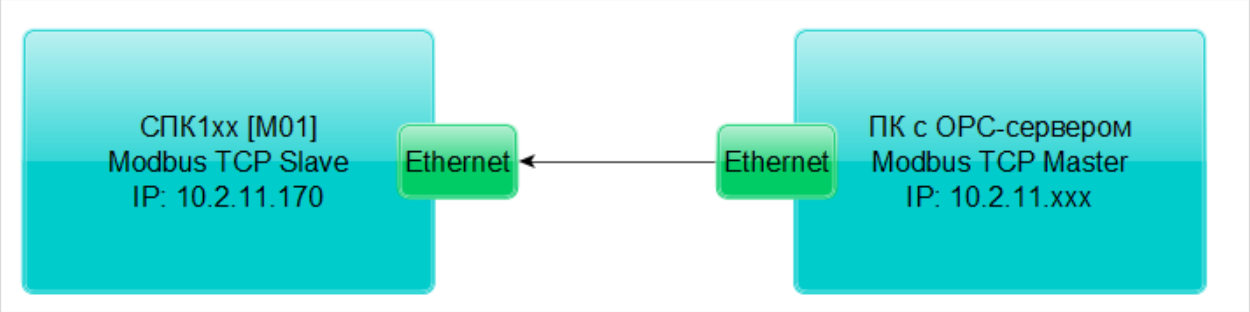


Рисунок 5.9.44 – Структурная схема примера

Пример создан в среде **CODESYS V3.5 SP17 Patch 3** и подразумевает запуск на **СПК1xx [M01]** с таргет-файлом **3.5.17.31.** 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_OwenCommunicationModbusTcpSlaveFiles_3517v1.zip

Сетевые параметры модулей приведены в таблице ниже:

Таблица 5.9.7 – Сетевые параметры устройств

Параметр	СПК1xx [M01]	ПК с OPC-сервером
IP-адрес	10.2.11.170	*любой адрес из сети, к которой подключен контроллер*
Порт	502	
Адрес (Unit ID)	1	-
Режим работы	slave	master

Для настройки обмена следует:

1. Подключить контроллер и ПК к общей локальной сети.

2. Убедиться, что в контроллере по пути **/mnt/ufs/home/root** находится файл **Arc1.log**. В рамках примера этот файл может быть загружен в контроллер вручную через утилиту WinSCP (см. руководство **CODESYS V3.5. FAQ**, п. 13.5).

3. Создать новый проект **CODESYS** с программой на языке **ST** или **CFC**:

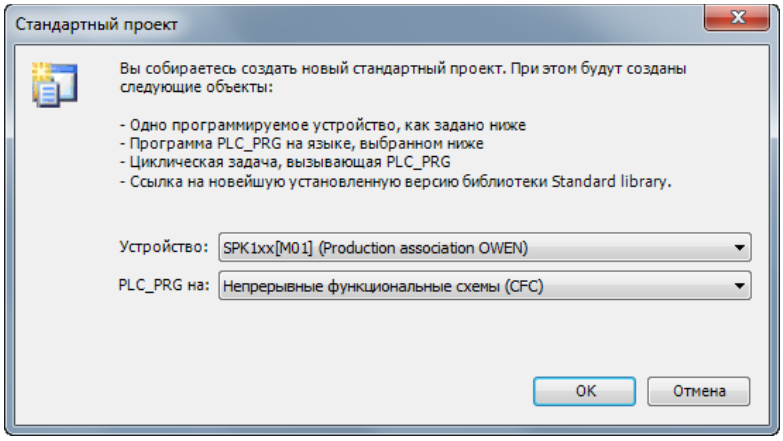


Рисунок 5.9.45 – Создание проекта CODESYS

ПРИМЕЧАНИЕ

Проект примера содержит программы на языках CFC и ST. По умолчанию используется программа на CFC (**PLC_PRG_CFC**). Для работы с программой на ST следует в конфигурации задач для задачи **MainTask** удалить вызов **PLC_PRG_CFC** и добавить вызов **PLC_PRG_ST**.

4. Объявить в программе следующие переменные:

```
PROGRAM PLC_PRG_CFC
VAR

    fbModbusTcpSlave:       OCL.MB_TcpSlave;        // ФБ для реализации Modbus Slave
    awSlaveData:            ARRAY [0..15] OF WORD;  // буфер данных Modbus Slave

    // путь к файлам архивов
    asFilePath:             ARRAY [1..8] OF STRING := ['/mnt/ufs/home/root/Arc1.log', 7('')];
END VAR
```

Рисунок 5.9.46 – Объявление переменных программы

5. Код программы будет выглядеть следующим образом:

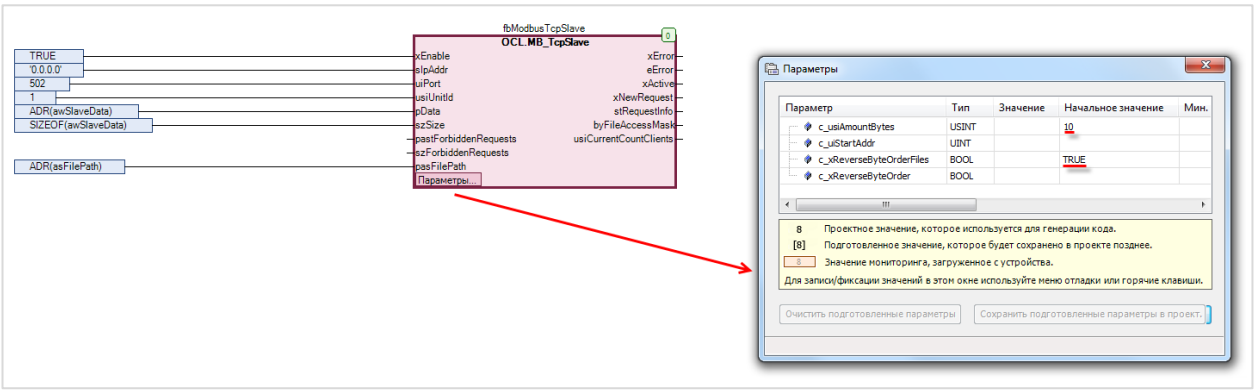


Рисунок 5.9.47 – Код программы на языке CFC

```
fbModbusTcpSlave
(
    xEnable                 := TRUE,
    sIpAddr                 := '0.0.0.0',
    uiPort                  := 502,
    usiUnitId               := 1,
    pData                   := ADR(awSlaveData),
    szSize                  := SIZEOF(awSlaveData),
    pasFilePath             := ADR(asFilePath),
    c_xReverseByteOrderFiles := TRUE,
    c_usiAmountBytes        := 10
);
```

Рисунок 5.9.48 – Код программы на языке ST

При запуске программы происходит вызов экземпляра ФБ MB_TcpSlave (**блок 0**), который выполняет функцию Modbus Slave. В качестве IP-адреса (**sIpAddr**), на котором используется slave, применяется специальный адрес 0.0.0.0, который соответствует адресам всех интерфейсов контроллера (то есть slave доступен по всем TCP/IP интерфейсам). Значения входов **uiPort** и **usiUnitId** соответствуют таблице 5.9.7.

Данные slave'а хранятся в массиве **awSlaveData**. В экземпляр ФБ передается указатель на этот массив (**pData**) и его размер в байтах (**szSize**).

Пути к файлам архивов хранятся в массиве строк **asFilePath**. В блок передается указатель на эту переменную. ФБ поддерживает до 8 файлов. Нумерация файлов ведется с **1**. В рамках примера используется только один (первый) файл.

Для работы с Insat MasterOPC Universal Modbus Server параметр **c_xReverseByteOrderFiles** (порядок байт при передаче файлов) должен иметь значение **TRUE**, а параметр **c_usiAmountBytes** (размер записи в файле) – значение **10**.

6. Установить и запустить MasterOPC Universal Modbus Server.

7. Нажать **ПКМ** на узел **Server** и добавить коммуникационный узел типа **TCP/IP**. В узле следует указать сетевые настройки в соответствии с таблицей 5.9.7. Для работы OPC-сервера в режиме **Modbus TCP Master** параметры **Modbus поверх TCP** и **Slave подключение** должны иметь значение **FALSE**.

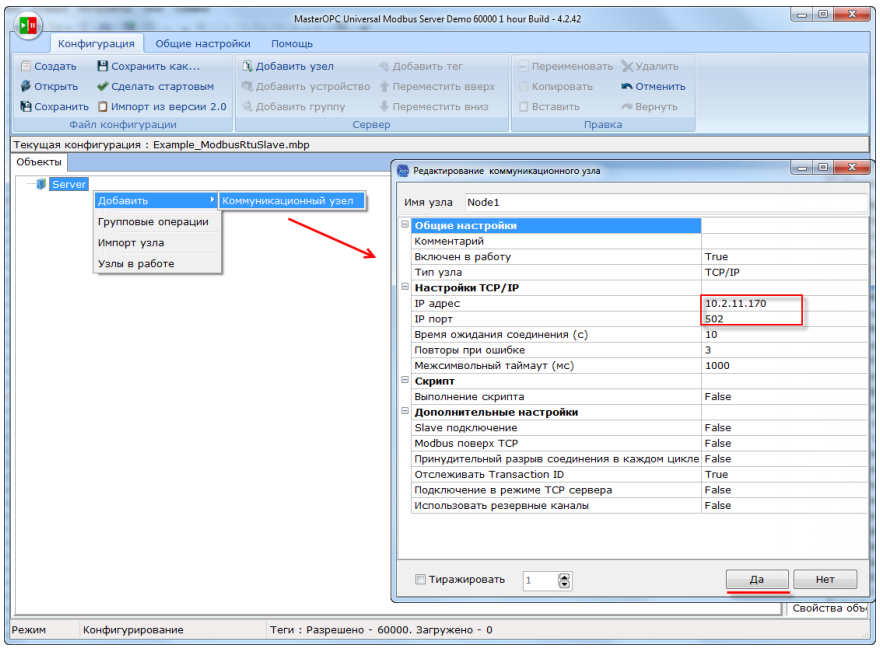


Рисунок 5.9.49 – Добавление коммуникационного узла

8. Нажать **ПКМ** на коммуникационный узел и добавить устройство с настройками по умолчанию.

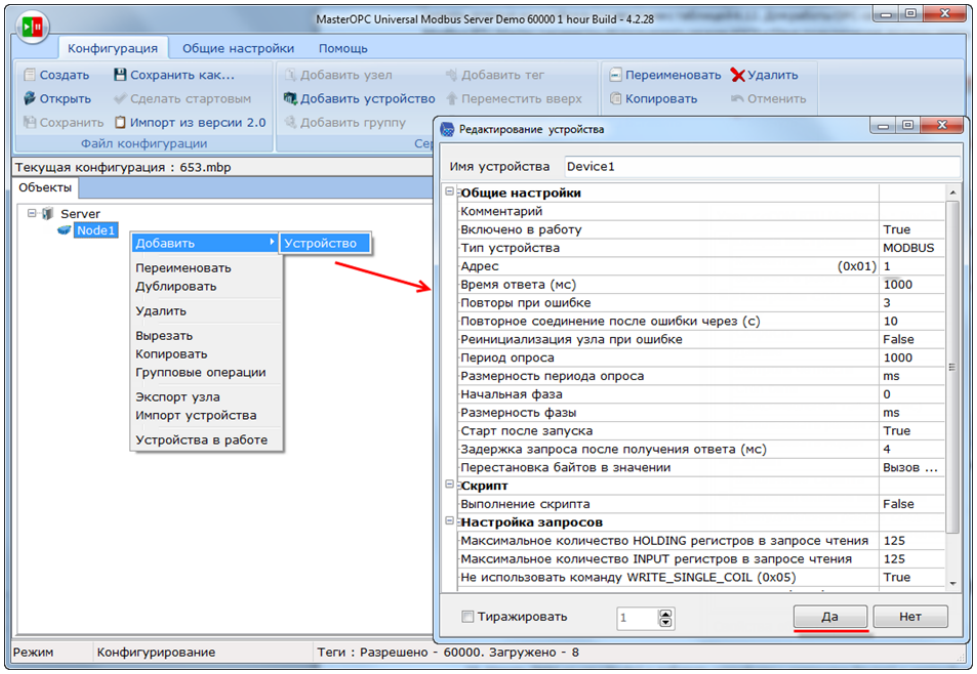


Рисунок 5.9.50 – Добавление устройства

9. Нажать **ПКМ** на устройство и импортировать подустройство **Owen History HDA.ssd** (по умолчанию оно находится в директории **C:\ProgramData\InSAT\MasterOPC Universal Modbus Server\SERVEREXPORT\SUBDEVICE_LIBRARY\OWEN**).

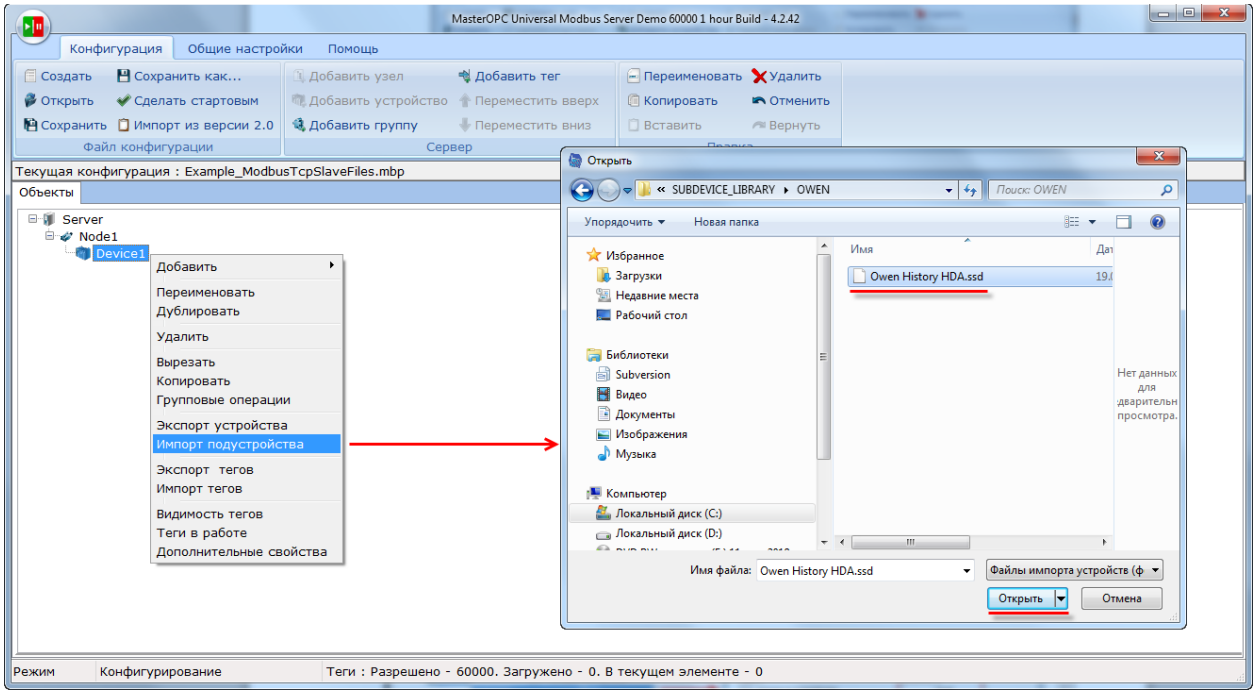


Рисунок 5.9.51 – Импорт подустройства

10. В появившемся подустройстве **Архив** указать номер файла. Нумерация файлов ФБ ведется с **1**, и в примере используется только один файл – поэтому следует установить значение **1**. В случае необходимости можно изменить другие настройки подустройства.

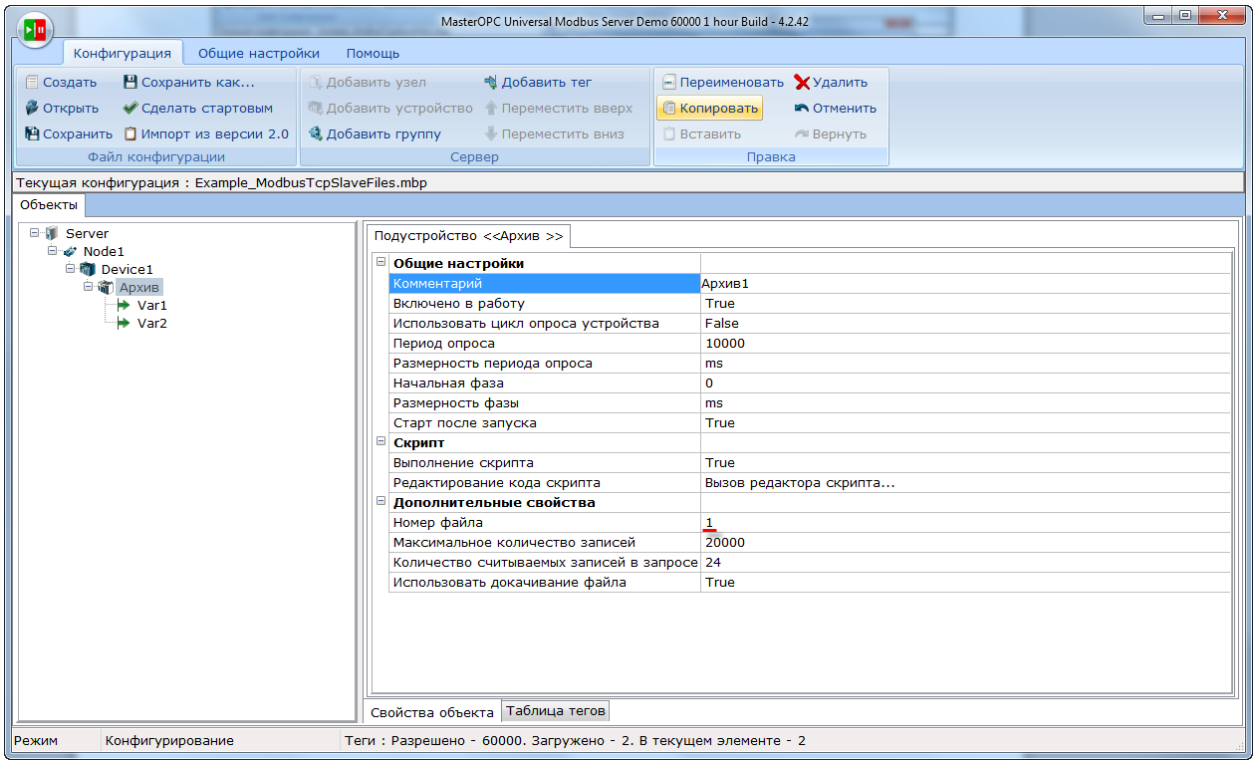


Рисунок 5.9.52 – Настройки подустройства

11. В рамках примера считывается файл **Arc1.log** формата архива ПЛК1xx. Этот файл включен в состав проекта примера. Файл содержит архив трех переменных с типами **UINT**, **UDINT** и **REAL** соответственно.

По умолчанию подустройство **Архив** содержит только два тега – типа **Uint32** (UDINT) и типа **Float** (REAL). Следует нажать **ПКМ** на подустройство и добавить тег с названием **Var0** (используется тип **Uint32**, так как в OPC-сервере нет типа **Uint16**) :

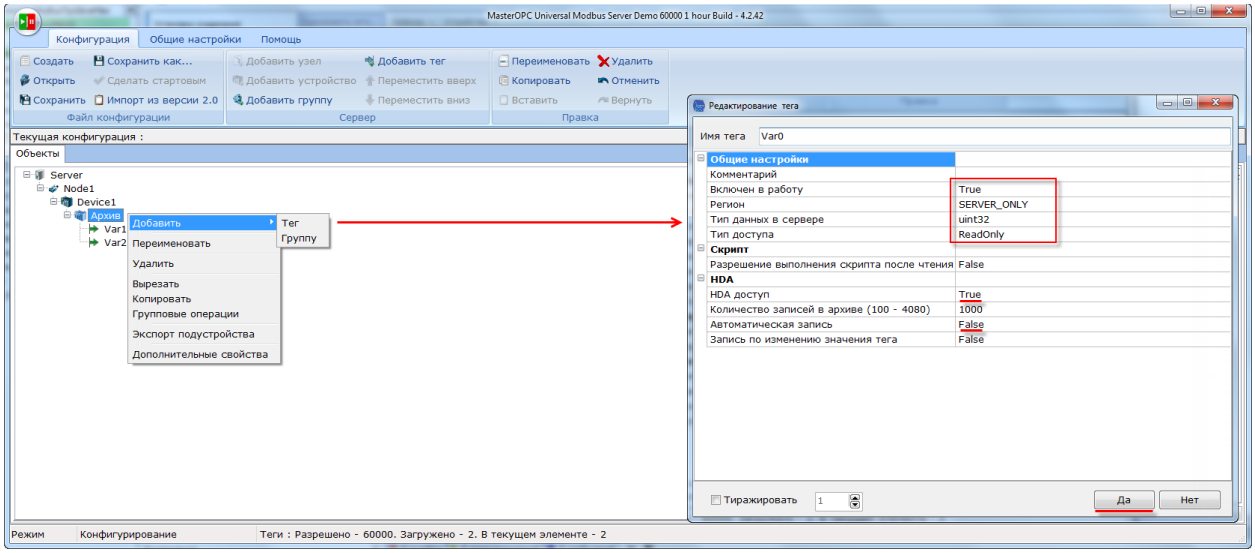


Рисунок 5.9.53 – Добавление тега в подустройство

12. Переместить тег **Var0** в дереве OPC-сервера с помощью кнопки **Переместить вверх:**

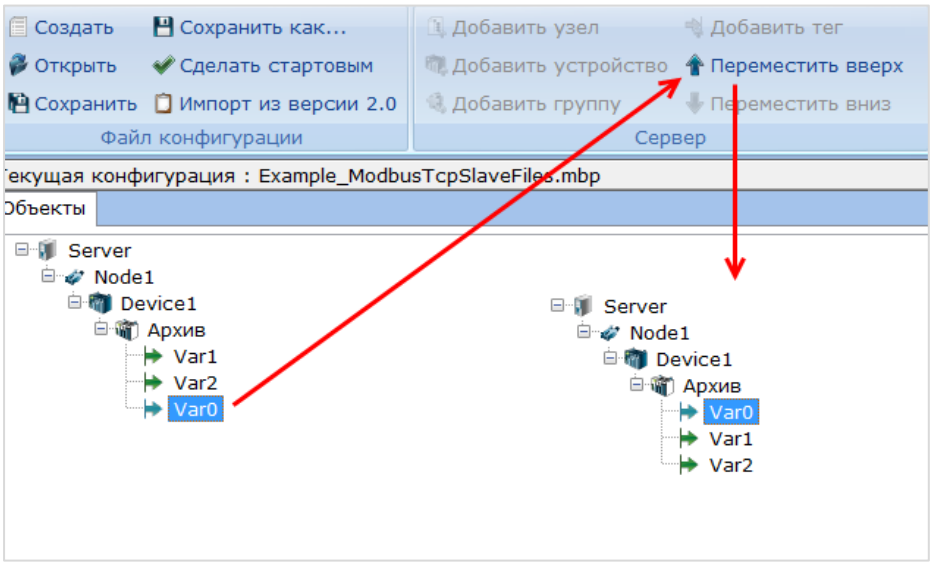


Рисунок 5.9.54 – Перемещение тегов в дереве OPC-сервера

13. Загрузить проект в контроллер и запустить его. Запустить OPC-сервер для контроля значений переменных. В тегах будут отображены последние значения, считанные из файла архива.

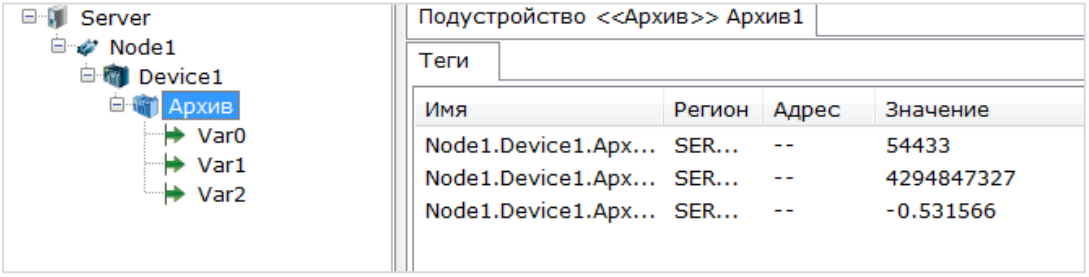


Рисунок 5.9.55 – Считывание и запись данных через OPC-сервер

Для просмотра истории тега следует выделить его в дереве проекта и перейти на вкладку **HDA** (в OPC формат отображения **DEC**, порядок – от новых записей к старым. В файле архива формат отображения **HEX**, порядок – от старых записей к новым):

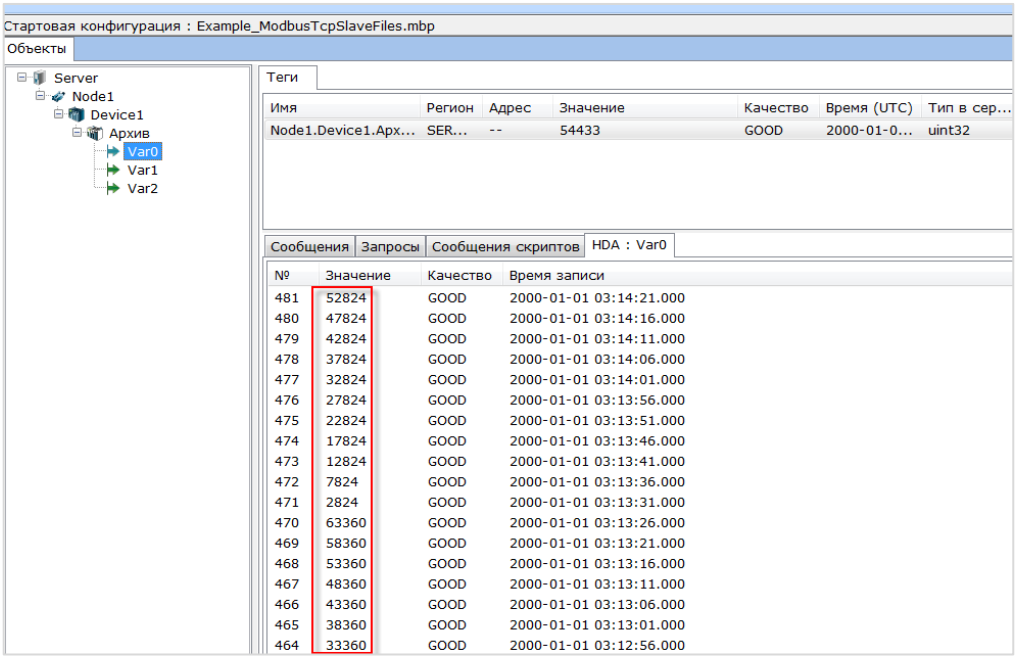


Рисунок 5.9.56 – Считывание архивных данных через OPC-сервер

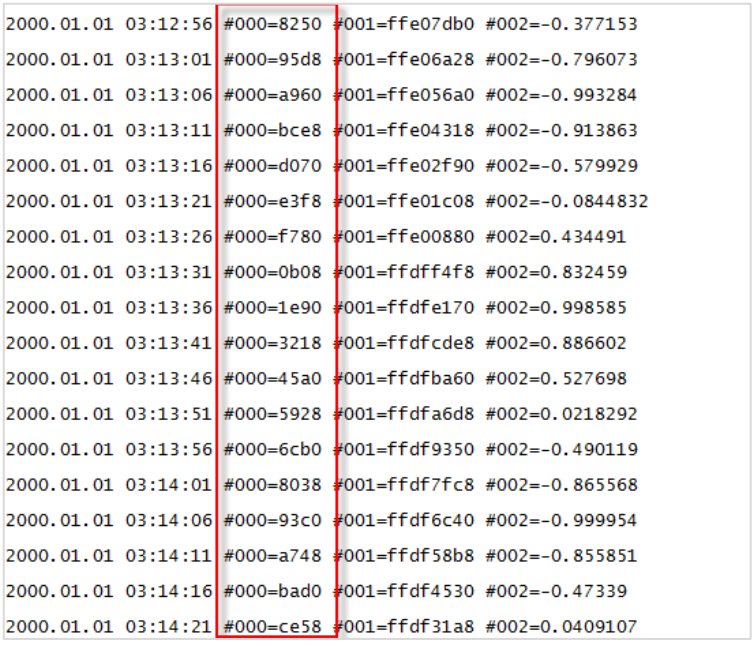


Рисунок 5.9.57 – Содержимое файла архива

Считанные исторические данные могут быть переданы в SCADA-систему с помощью технологии **OPC HDA**.

6 FAQ

6.1 Что делать, если не удается наладить обмен по Modbus?

В случае возникновения проблем при настройке обмена рекомендуется:

1. Проверить (прозвонить) линию связи. Проверить распиновку кабеля (в случае использования кабелей с разъемом DB9). Проверить, что контакты **A** и **B** (или **RXD** и **TXD**) не перепутаны местами.

2. Проверить используемый номер COM-порта в **CODESYS**.

3. Проверить соответствие сетевых настроек контроллера и подключаемых приборов (скорость обмена, количество стоп-бит, адреса slave-устройств и т. д.).

4. Проверить настройки опроса регистров: используемые коды функций, адреса регистров, типы данных и т. д.

5. Проверить, нет ли разрывов в карте регистров slave-устройства (в случае использования групповых запросов).

6. Проверить, что на шине находится только одно master-устройство (для **Modbus RTU/ASCII**).

7. Проверить, что в сети нет slave-устройств с одинаковыми адресами.

8. В случае опроса модулей **Mx110** – с помощью программы Конфигуратор Mx110 проверить, что модулям заданы корректные сетевые настройки.

9. В случае использования стандартных средств конфигурирования – проверить, что на вкладке привязки переменных для параметра **Всегда обновлять переменные** установлено значение **Вкл. 2 (Всегда в задаче цикла шины)**.

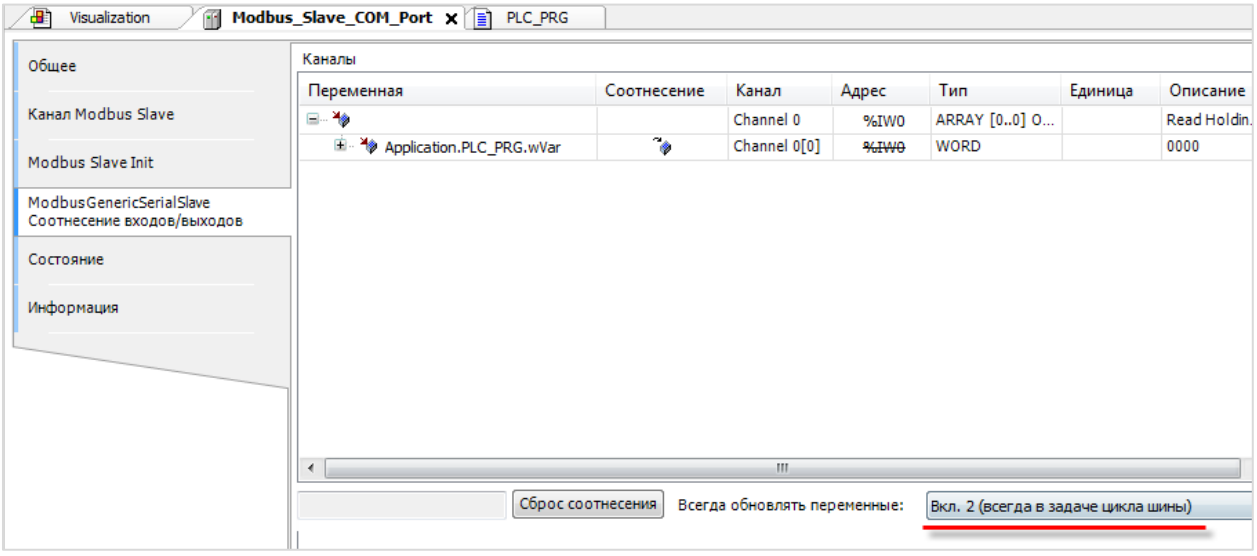


Рисунок 6.1 – Настройка параметра Всегда обновлять переменные

Если успешное выполнение всех вышеперечисленных пунктов не приведет к настройке обмена, то следует обратиться в техническую поддержку компании Овен. При обращении следует предоставить указанную ниже информацию и материалы:

- модель и модификацию контроллера, версию прошивки, таргет-файла, среды CODESYS и используемых компонентов Modbus;
- **подробное** описание проблемы;
- структурную схему сети с указанием используемых портов и адресов;

- маркировку используемых кабелей, информацию об изоляции и заземлении, наличии согласующих резисторов (терминаторов) и повторителей, а также условиях, в которых находятся приборы (например, о присутствии в шкафу автоматики силового оборудования);
- архивы проектов для программируемого оборудования Овен (СПК, ПЛК, панели оператора и т. д.), скриншоты сетевых настроек конфигурируемых приборов (модули Mx110, ТРМ и т. д.) и приборов других производителей (а также карты регистров этих устройств).

6.2 Каким образом считать/передать значение с плавающей точкой (REAL)?

См. п. 4.8 (перечисления, указатели) и п. 5.8 (функции библиотеки **OwenCommunication**).

6.3 Каким образом считать/передать отрицательное значение (INT)?

Если необходимо считать отрицательное число, то после получения соответствующей переменной типа **WORD** следует преобразовать ее в **INT** с помощью стандартного оператора **WORD_TO_INT**.

Если необходимо записать значение типа **INT**, то следует преобразовать его в **WORD** с помощью стандартного оператора **INT_TO_WORD**. На устройстве, которое получит эти данные, необходимо будет произвести обратную операцию.

6.4 Вопросы по стандартным средствам конфигурирования

6.4.1 Какие версии компонентов рекомендуются к использованию?

Таблица рекомендуемых версий компонентов Modbus приведена в приложении А.

Узнать используемую версию компонента можно на вкладке **Информация**:

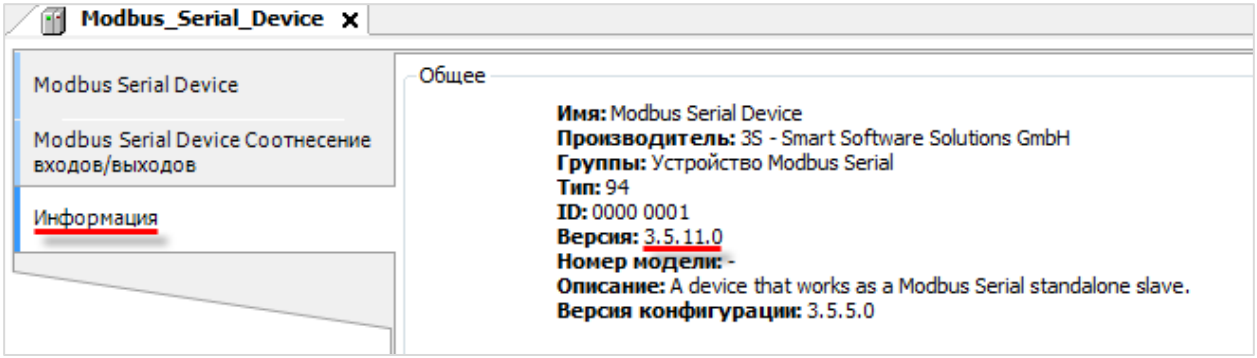


Рисунок 6.2 – Версия компонента Modbus Slave Serial Device

Чтобы изменить версию компонента следует нажать на нем **ПКМ** и выбрать команду **Обновить устройство**.

6.4.2 Modbus Serial Master: как реализовать чтение/запись по триггеру?

В настройках канала для параметра **Триггер** следует выбрать значение **Передний фронт**.

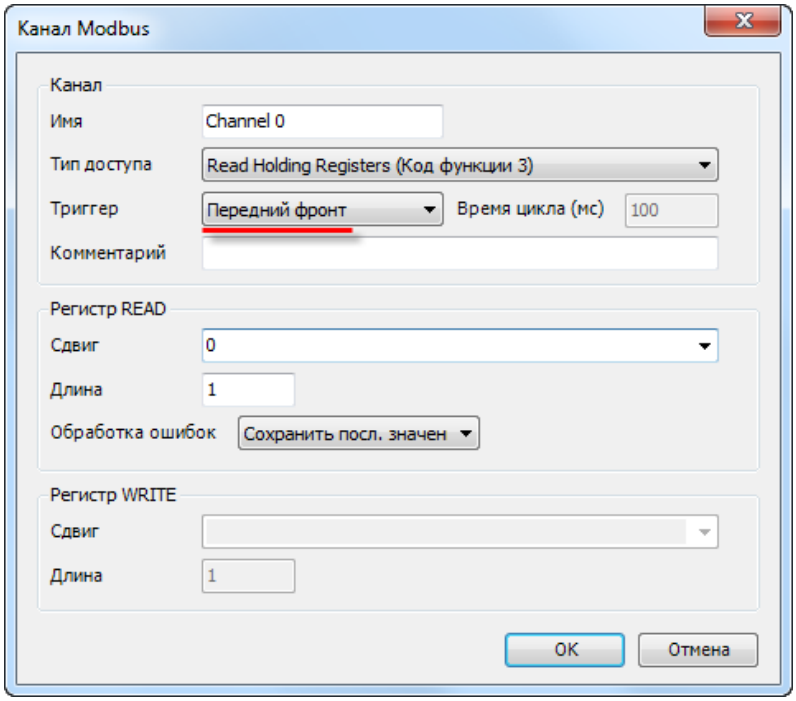


Рисунок 6.3 – Настройка опроса канала по триггеру

После этого на вкладке привязки переменных к каналу появится строка для триггерной переменной. Чтение/запись будет происходить по переднему фронту этой переменной.

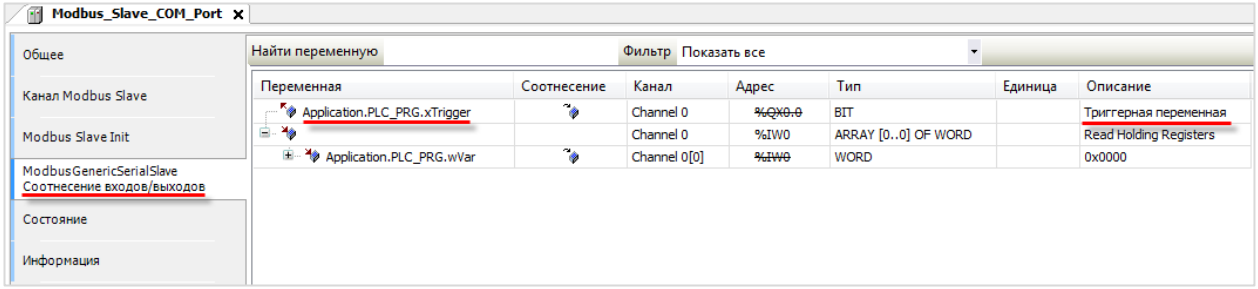


Рисунок 6.4 – Привязка триггерной переменной

6.4.3 Modbus Serial/TCP Device: почему принятые значения сбрасываются в 0?

Если контроллер используется в режиме **Modbus RTU Slave**, а master-устройство выполняет запись по триггеру (например, панель оператора записывает введенное значение однократно, а не циклически), то следует отключить галочку **Сторожевой таймер**. В противном случае регистры slave'а будут обнуляться, если в течение заданного времени не будет получено ни одного запроса от master-устройства.

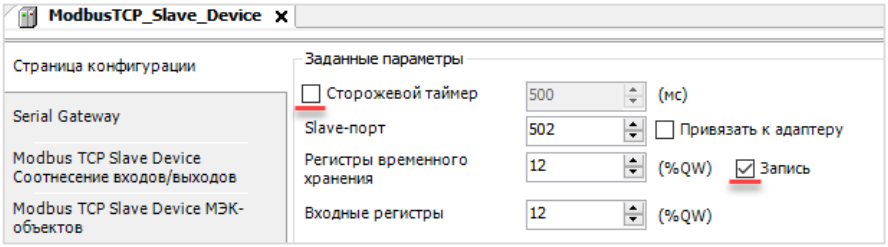


Рисунок 6.5 – Настройки компонента Modbus Serial Device

6.4.4 Modbus Serial/TCP Device: Можно ли менять данные holding регистров из программы?

Начиная с версии **3.5.16.0** это возможно с помощью установки галочки **Запись** (см. рис. 6.5). В более ранних версиях CODESYS следует использовать ФБ MB_SerialSlave или MB_TcpSlave из библиотеки OwenCommunication.

6.4.5 Как произвести диагностику обмена в программе?

См. п. 4.6. В случае использования шаблонов модулей Mx110 и 210 см. п. 3.4.

6.4.6 Modbus Serial/TCP Master: Можно ли обойти ограничение на 100 каналов опроса?

Компоненты **Modbus Serial Master** и **Modbus TCP Master** имеют ограничение на **100** каналов опроса. Некоторые slave-устройства имеют много параметров, и этого числа каналов оказывается недостаточно. В этом случае в компонент **Modbus TCP Master** можно добавить несколько компонентов **Modbus TCP Slave** с одинаковыми IP-адресами, чтобы обойти ограничение на число каналов (если slave-устройство поддерживает несколько одновременных TCP-подключений). Начиная с версии **3.5.16.0** в компонент **Modbus Serial Master** можно добавить несколько компонентов **Modbus Slave COM Port** с одинаковыми адресами устройств (**Slave ID**). В сообщениях компиляции будет отображаться соответствующее предупреждение, но оно не помешает загрузке проекта. В более ранних версиях CODESYS следует использовать ФБ MB_SerialRequest из библиотеки OwenCommunication.

6.4.7 Как расшифровываются пиктограммы статуса обмена?

Расшифровка пиктограмм статуса обмена приведена в таблице 6.1:

Таблица 6.1 – Расшифровка пиктограмм статуса обмена

Пиктограмма	Описание для компонентов Modbus Serial Master / Modbus TCP Master	Описание для остальных компонентов
	На запрос получен корректный ответ	Компонент работает корректно
	Ошибка в дочернем компоненте	
	Отсутствует лицензия на компонент	
	Ожидание соединения	Отсутствие запросов от master-устройства
	На запрос получен ответ с кодом ошибки Modbus	Компонент работает некорректно (например, IP-адрес в компоненте Ethernet отличается от реального IP-адреса контроллера)
	Информация о прошедшей ошибке (сквитировать можно на вкладке **Состояние**)	
	Ответ не получен	Ошибка инициализации компонента (например, не удалось выделить память)

6.5 Вопросы по библиотеке OwenCommunication

6.5.1 В примерах работы с библиотекой используются действия. Как добавить их в проект?

Для добавления действия следует нажать **ПКМ** на нужный компонент (например, программу) и выбрать команду **Добавление объекта – Действие**.

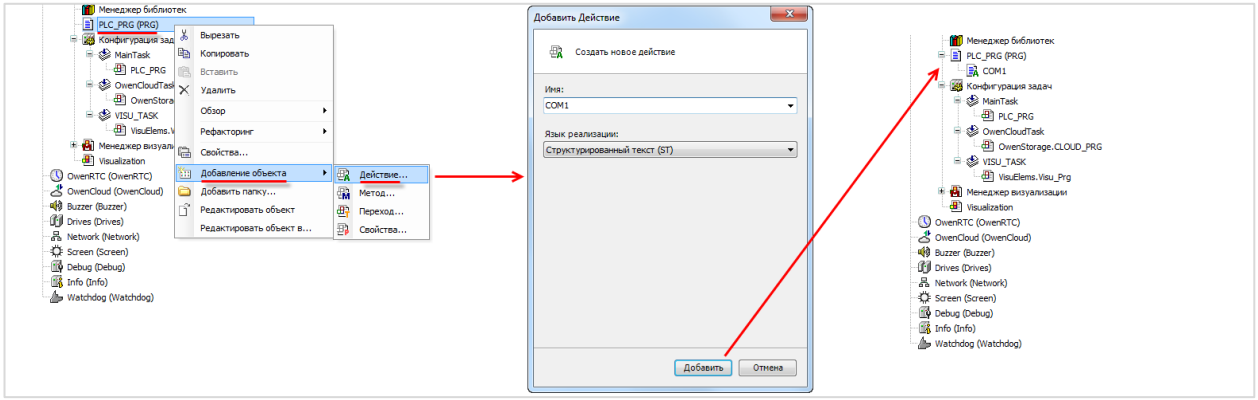


Рисунок 6.6 – Добавление действия в проект CODESYS

6.5.2 Позволяет ли библиотека организовать опрос с более высокой частотой по сравнению со стандартными средствами конфигурирования?

Нет. ФБ библиотеки работают асинхронно и построены на асинхронных блоках обмена из системных библиотек. Преимуществами библиотеки **OwenCommunication** являются расширенный функционал и возможность управления обменом из кода программы, но в большинстве случаев частота опроса будет ниже по сравнению со стандартными средствами конфигурирования.

6.5.3 Ограничения, связанные с библиотекой CAA AsyncManager

Библиотеки, включающие в себя асинхронно выполняемые функциональные блоки (к таким библиотекам, например, относятся **CAA SerialCom**, **CAA NetBaseServices**, **CAA File** и др.), используют в своей реализации библиотеку **CAA AsyncManager**.

Данная библиотека имеет следующее ограничение: ***в каждый момент времени (в пределах одного цикла контроллера) в программе может вызываться не более 20 экземпляров асинхронно выполняемых функциональных блоков***.

Под «вызовом» подразумевается исполнение кода, в котором экземпляру асинхронно выполняемого блока передается на вход **xExecute** значение **TRUE** или в котором выход **xBusy** имеет значение **TRUE**.

Библиотека OwenCommunication построена на блоках библиотеки **CAA SerialCom**, и также подвержена этим ограничениям, причем ФБ COM_Control, TCP_Client, MB_SerialSlave и MB_TcpSlave включают в себя один экземпляр асинхронно выполняемого блока, а все остальные ФБ библиотеки (MB_SerialRequest и т. д.) – два.

На практике это означает следующее: если одновременно (в пределах одного цикла контроллера), вызывать 3 экземпляра ФБ **COM_Control**, 3 экземпляра ФБ **UNM_SerialRequest** и, например, 11 экземпляров ФБ **FILE.Write** из библиотеки **CAA File**, то лимит для асинхронно выполняемых блоков окажется исчерпанными – при попытке вызова в том же цикле ПЛК еще одного экземпляра он вернет код ошибки (для разных библиотек этот код будет разным, например, для **OwenCommunicaton** – **WRONG_PARAMETER**, а для CAA File – 5802).

На практике это ограничение обычно не создает существенных проблем при программировании на языке ST – общепринятый подход подразумевает «сброс» (вызов со значением **FALSE** на входе **xExecute**) блока после его выполнения; этот подход используется и в примерах документа.

Но в языке CFC цепочка блоков на холсте обрабатывается более специфическим образом – даже при визуальном значении **FALSE** на входе **xExecute** какого-либо блока его «внутренние» экземпляры асинхронно выполняемых блоков могут сохранять значение **TRUE** на своих входах **xExecute** до следующего «прохода» данного ФБ на холсте и учитываться в упомянутом выше ограничении. Реализовать «сброс» каждого экземпляра на холсте после его выполнения в графическом языке программирования достаточно трудоемко; поэтому рекомендуется для решения сложных задач (реализация протоколов обмена, работа с файлами и т. д.) использовать исключительно язык ST.

Ниже приведен пример проявления ограничения на число одновременно вызываемых асинхронно выполняемых функциональных блоков в языке CFC: на холсте расположены экземпляр **COM_Control** (включает 1 экземпляр асинхронно выполняемого блока) и 10 экземпляров **MB_SerialRequest** (включает 2 экземпляра асинхронно выполняемых блоков). Из-за особенностей языка CFC «сброс» блоков происходит только перед началом нового «прохода» холста, и поэтому 10-й экземпляр **MB_SerialRequest** возвращает ошибку **WRONG_PARAMETER** (она возникает на этапе вызова в нем экземпляра ФБ **COM.Read**, потому что он является 21-м экземпляром асинхронно выполняемых блоков на холсте).

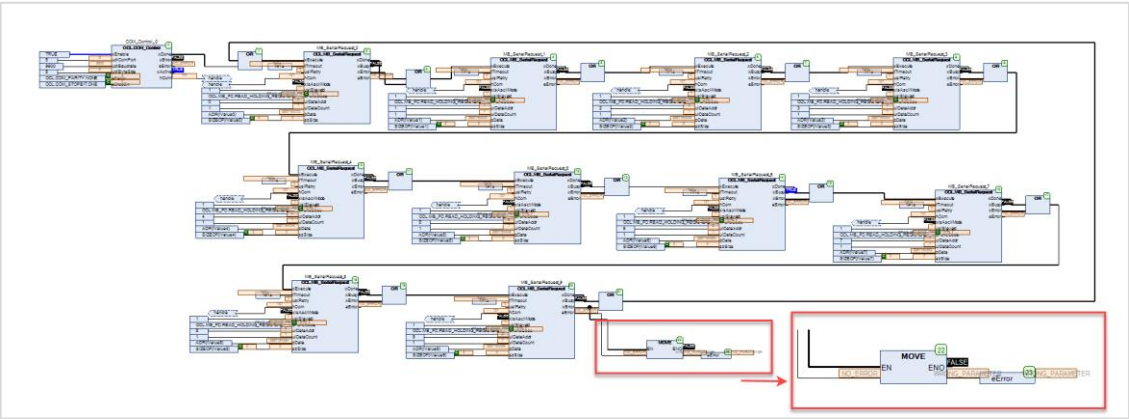


Рисунок 6.7 – Пример проявления ограничения на число одновременно вызываемых асинхронно выполняемых функциональных блоков в языке CFC

Приложение A. Рекомендуемые версии компонентов Modbus

Различные версии **CODESYS** включают в себя разные версии Modbus-компонентов. В таблице ниже приведены рекомендуемые версии компонентов в зависимости от используемой в контроллере системы исполнения. Информация о соответствии версий прошивок контроллеров, таргет-файлов и CODESYS приведена в документе **CODESYS V3.5. FAQ**.

Таблица A.1 – Рекомендуемые версии компонентов Modbus

Система исполнения	**3.5.17.30**	**3.5.16.30**	**3.5.14.30**	**3.5.11.50**	**3.5.4.30**
Среда программирования	3.5.17.30	3.5.16.30	3.5.14.30	3.5.11.54	3.5.5.5
Modbus COM	3.5.16.0	3.5.16.0	3.5.11.20	3.5.11.20	3.4.0.0
Modbus RTU Master	3.5.17.0	3.5.16.0	3.5.14.0	3.5.11.20	3.5.5.0
Modbus RTU Slave	3.5.16.0	3.5.16.0	3.5.10.30	3.5.10.30	3.5.4.0
Шаблоны Mx110	3.5.11.x	3.5.11.x	3.5.11.x	3.5.11.x	3.5.4.13
Modbus Serial Device	3.5.17.0	3.5.16.10	3.5.14.0	3.5.11.0	3.5.5.0
Ethernet-адаптер	3.5.17.0	3.5.16.0	3.5.14.0	3.5.11.0	3.4.2.0
Modbus TCP Master	3.5.17.0	3.5.16.0	3.5.14.0	3.5.11.30	3.5.5.0
Modbus TCP Slave	3.5.16.0	3.5.16.0	3.5.12.0	3.5.11.30	3.5.4.0
Шаблоны Mx210	3.5.11.x	3.5.11.x	3.5.11.x	3.5.11.x	-
Modbus TCP Device	3.5.17.0	3.5.16.0	3.5.14.0	3.5.11.0	3.5.2.0

Приложение Б. Листинги примеров

Б1 Листинг примера из п. 5.9.1

Б.1.1 Код программы PLC_PRG_ST

```
PROGRAM PLC_PRG_ST
VAR
	fbComControl1:  OCL.COM_Control;    // ФБ управления портом COM1
	fbComControl2:  OCL.COM_Control;    // ФБ управления портом COM2
	fbMV110_8A_AI1:   OCL.MB_SerialRequest;   // ФБ опроса модуля MB110-8A
	fbMV110_16D_DI:   OCL.MB_SerialRequest;   // ФБ опроса модуля MB110-16Д
	fbMU110_16R_DO: OCL.MB_SerialRequest;   // ФБ опроса модуля МУ110-16Р

	rAI1:		REAL;			// значение 1-го входа модуля MB110-8A
	wDiMask:	WORD;			// битовая маска входов модуля MB110-16Д
	wDoMask:	WORD;			// битовая маска выходов модуля МУ110-16Р
	xDi0:		BOOL;			// значение 1-го входа модуля MB110-16Д
	xDo0:		BOOL;			// значение 1-го выхода модуля МУ110-16Р

	awAI1:		ARRAY [0..1] OF WORD;	// регистры, считанные с модуля MB110-8A

	iStateCom1:	INT;			// шаг опроса по порту COM1
	iStateCom2:	INT;			// шаг опроса по порту COM2
END_VAR


// чтобы запустить пример на ST на контроллере требуется:
// 1. Удалить из задачи MainTask вызов программы PLC_PRG_CFC
// 2. Привязать к задаче MainTask вызов программа PLC_PRG_ST

COM1();
COM2();

xDo0 := xDi0 AND (rAI1 > 30.0);
```

Б.1.2 Код действия COM1

```
CASE iStateCom1 OF

    0:      // открытие COM-порта COM1

            fbComControl1
            (
                    xEnable := TRUE,
                    usiComPort  := 1,
                    udiBaudrate := 115200,
                    udiByteSize := 8,
                    eParity   := OCL.COM_PARITY.NONE,
                    eStopBit    := OCL.COM_STOPBIT.ONE
            );

            IF fbComControl1.xDone THEN
                    iStateCom1 := 1;
            END_IF

    1: // опрос MB110-8A

            fbMV110_8A_AI1
            (
                    xExecute                := fbComControl1.xActive,
                    tTimeout                := T#200MS,
                    usiRetry                := 3,
                    hCom                    := fbComControl1.hCom,
                    xIsAsciiMode            := FALSE ,
                    usiSlaveId              := 1,
                    eFuncCode               := OCL.MB_FC.READ_HOLDING_REGISTERS,
                    uiDataAddr              := 4,
                    uiDataCount             := 2,
                    pData                   := ADR(awAI1),
                    szSize                  := SIZEOF(awAI1)
            );

            IF fbMV110_8A_AI1.xDone OR fbMV110_8A_AI1.xError THEN
                    // после выполнения блока его надо сбросить
                    fbMV110_8A_AI1(xExecute := FALSE);
                    rAI1 := OCL.WORD2_TO_REAL(awAI1[0], awAI1[1], FALSE);
                    iStateCom1 := 2;
            END_IF

    2: // здесь можно добавить опрос следующего устройства
       // после опроса последнего устройства возвращаемся к опросу первого
                    iStateCom1 := 1;

END_CASE
```

Б.1.3 Код действия COM2

```
CASE iStateCom2 OF

    0:  // открытие COM-порта COM2

        fbComControl2
        (
            xEnable  := TRUE,
            usiComPort  := 2,
            udiBaudrate := 115200,
            udiByteSize := 8,
            eParity   := OCL.COM_PARITY.NONE,
            eStopBit    := OCL.COM_STOPBIT.ONE
        );

        IF fbComControl2.xDone THEN
            iStateCom2 := 1;
        END_IF

    1:  // опрос модуля MB110-16Д

        fbMV110_16D_DI
        (
            xExecute            := fbComControl2.xActive,
            tTimeout            := T#200MS,
            usiRetry            := 3,
            hCom                := fbComControl2.hCom,
            xIsAsciiMode        := FALSE ,
            usiSlaveId          := 1,
            eFuncCode           := OCL.MB_FC.READ_HOLDING_REGISTERS,
            uiDataAddr          := 51,
            uiDataCount         := 1,
            pData               := ADR(wDiMask),
            szSize              := SIZEOF(wDiMask)
        );

        IF fbMV110_16D_DI.xDone OR fbMV110_16D_DI.xError THEN
            // после выполнения блока его надо сбросить
            fbMV110_16D_DI(xExecute := FALSE);

            xDi0 := wDiMask.0;

            iStateCom2 := 2;
        END_IF
```

```
2:  // опрос модуля МУ110-16Р

        wDoMask.0 := xDo0;


        fbMU110_16R_DO
        (
                xExecute            := fbComControl2.xActive,
                tTimeout            := T#200MS,
                usiRetry            := 3,
                hCom                := fbComControl2.hCom,
                xIsAsciiMode        := FALSE ,
                usiSlaveId          := 17,
                eFuncCode           := OCL.MB_FC.WRITE_MULTIPLE_REGISTERS,
                uiDataAddr          := 50,
                uiDataCount         := 1,
                pData               := ADR(wDoMask),
                szSize              := SIZEOF(wDoMask)
        );

        IF fbMU110_16R_DO.xDone OR fbMU110_16R_DO.xError THEN
                // после выполнения блока его надо сбросить
                fbMU110_16R_DO(xExecute := FALSE);

                // возвращаемся к опросу первого модуля
                iStateCom2 := 1;
        END_IF

END_CASE
```

Б2 Листинг примера из п. 5.9.2

```
PROGRAM PLC_PRG_ST
VAR
	fbComControl1:  OCL.COM_Control;		// ФБ управления портом COM1

	fbRealToWord2:  OCL.REAL_TO_WORD2;		// ФБ преобразования REAL в две…
							// …переменные типа WORD

	fbModbusSerialSlave:	OCL.MB_SerialSlave;		// ФБ для реализиации Modbus Slave

	awSlaveData:		ARRAY [0..15] OF WORD;  // буфер данных Modbus Slave

	(* значения, полученные от OPC *)
	xVar_Opc:				BOOL;
	wVar_Opc:				WORD;
	rVar_Opc:				REAL;
	sVar_Opc:				STRING(15);

	(* значения для передачи в OPC *)
	xVar_Plc:				BOOL;
	wVar_Plc:				WORD;
	rVar_Plc:				REAL;
	sVar_Plc:				STRING(15);

	xWrite:		BOOL;	// команда записи данных из программы в регистры Modbus Slave
	fbWriteEdge:	R_TRIG; // триггер для однократной записи
END_VAR


// чтобы запустить пример на ST на контроллере требуется:
// 1. Удалить из задачи MainTask вызов программа PLC_PRG_CFC
// 2. Привязать к задаче MainTask вызов программы PLC_PRG_ST


fbComControl1
(
	xEnable := TRUE,
	usiComPort  := 1,
	udiBaudrate := 115200,
	udiByteSize := 8,
	eParity   := OCL.COM_PARITY.NONE,
	eStopBit    := OCL.COM_STOPBIT.ONE
);
```

```
fbModbusSerialSlave
(
        xEnable         :=fbComControl1.xActive,
        hCom            := fbComControl1.hCom,
        usiSlaveId      := 1,
        pData           := ADR(awSlaveData),
        szSize          := SIZEOF(awSlaveData)
);

// данные, полученные от OPC
xVar_Opc := awSlaveData[0].0;
wVar_Opc := awSlaveData[1];
rVar_Opc := OCL.WORD2_TO_REAL(awSlaveData[2], awSlaveData[3], FALSE);
OCL.SWAP_DATA( ADR(awSlaveData[4]), ADR(sVar_Opc), SIZEOF(sVar_Opc), TRUE, FALSE, FALSE );


// по команде записываем переменные из программы в регистры Modbus Slave
fbWriteEdge(CLK := xWrite);

IF fbWriteEdge.Q THEN

        awSlaveData[0].0 := xVar_Plc;
        awSlaveData[1]   := wVar_Plc;
        fbRealToWord2(rInput := rVar_Plc, wOutput1 => awSlaveData[2], wOutput2 => awSlaveData[3]);
        OCL.SWAP_DATA( ADR(sVar_Plc), ADR(awSlaveData[4]), SIZEOF(sVar_Plc), TRUE, FALSE, FALSE );

END_IF
```

Б3 Листинг примера из п. 5.9.3

Б.3.1 Код программы PLC_PRG_ST

```
PROGRAM PLC_PRG_ST
VAR
	fbTcpClientMV210:	OCL.TCP_Client;  // ФБ TCP-подключения к модулю MB210-101
	fbTcpClientMK210:	OCL.TCP_Client;  // ФБ TCP-подключения к модулю МК210-301
	fbMV210_101_AI1:	OCL.MB_TcpRequest;	// ФБ опроса модуля MB210-101
	fbMK210_301_DI:	OCL.MB_TcpRequest;	// ФБ опроса входов модуля МК210-301
	fbMK210_301_DO:	OCL.MB_TcpRequest;	// ФБ опроса выходов модуля МК210-301

	rAI1:			REAL;		// значение 1-го входа модуля MB210-101
	wDiMask:		WORD;		// битовая маска входов модуля МК210-301
	wDoMask:		WORD;		// битовая маска выходов модуля МК210-301
	xDi0:			BOOL;		// значение 1-го входа модуля МК210-301
	xDo0:			BOOL;		// значение 1-го выхода модуля МК210-301

	awAI1:		ARRAY [0..1] OF WORD;	// регистры, считанные с модуля MB210-101

	iStateMV210:	INT;			// шаг опроса модуля MB210-101
	iStateMK210:	INT;			// шаг опроса модуля МК210-301
END_VAR


// чтобы запустить пример на ST на контроллере требуется:
// 1. Удалить из задачи MainTask вызов программы PLC_PRG_CFC
// 2. Привязать к задаче MainTask вызов программы PLC_PRG_ST

MV210_101();
MK210_301();

xDo0 := xDi0 AND (rAI1 > 30.0);
```

Б.3.2 Код действия MV210_101

```
CASE iStateMV210 OF

    0:      // подключение к модулю

            fbTcpClientMV210
            (
                    xEnable  := TRUE,
                    tTimeout         := T#5S,
                    sIpAddr          := '10.2.11.181',
                    uiPort           := 502
            );

            IF fbTcpClientMV210.xActive THEN
                    iStateMV210 := 1;
            END_IF

            IF fbTcpClientMV210.xError THEN
                    fbTcpClientMV210(xEnable := FALSE);
            END_IF

    1: // опрос MB210-101

            fbMV210_101_AI1
            (
                    xExecute                 := fbTcpClientMV210.xActive,
                    tTimeout                 := T#50MS,
                    usiRetry                 := 3,
                    hConnection              := fbTcpClientMV210.hConnection,
                    xIsRtuOverTcpMode        := FALSE ,
                    usiUnitId                := 1,
                    eFuncCode                := OCL.MB_FC.READ_HOLDING_REGISTERS,
                    uiDataAddr               := 4000,
                    uiDataCount              := 2,
                    pData                    := ADR(awAI1),
                    szSize                   := SIZEOF(awAI1)
            );

            IF fbMV210_101_AI1.xDone OR fbMV210_101_AI1.xError THEN
                    // после выполнения блока его надо сбросить
                    fbMV210_101_AI1(xExecute := FALSE);
                    rAI1 := OCL.WORD2_TO_REAL(awAI1[1], awAI1[0], FALSE);
                    iStateMV210 := 2;
            END_IF
```

```
    2: // здесь можно добавить следующий запрос к модулю
       // после выполнения возвращаемся на шаг 0 для проверки состояния соединения
                    iStateMV210 := 0;

END_CASE
```

Б.3.3 Код действия MK210_301

```
CASE iStateMK210 OF

    0:      // подключение к модулю

            fbTcpClientMK210
            (
                    xEnable  := TRUE,
                    tTimeout        := T#5S,
                    sIpAddr         := '10.2.11.180',
                    uiPort          := 502
            );

            IF fbTcpClientMK210.xActive THEN
                    iStateMK210 := 1;
            END_IF

            IF fbTcpClientMK210.xError THEN
                    fbTcpClientMK210(xEnable := FALSE);
            END_IF


    1:  // опрос дискретных входов модуля MK210-301

            fbMK210_301_DI
            (
                    xExecute                := fbTcpClientMK210.xActive,
                    tTimeout                := T#50MS,
                    usiRetry                := 3,
                    hConnection             := fbTcpClientMK210.hConnection,
                    xIsRtuOverTcpMode       := FALSE ,
                    usiUnitId               := 1,
                    eFuncCode               := OCL.MB_FC.READ_HOLDING_REGISTERS,
                    uiDataAddr              := 51,
                    uiDataCount             := 1,
                    pData                   := ADR(wDiMask),
                    szSize                  := SIZEOF(wDiMask)
            );
```

```
            IF fbMK210_301_DI.xDone OR fbMK210_301_DI.xError THEN
                // после выполнения блока его надо сбросить
                fbMK210_301_DI(xExecute := FALSE);

                xDi0 := wDiMask.0;

                iStateMK210 := 2;
            END_IF


        2:  // запись дискретных выходов модуля МК210-301

            wDoMask.0 := xDo0;

            fbMK210_301_DO
            (
                xExecute                := fbTcpClientMK210.xActive,
                tTimeout                := T#50MS,
                usiRetry                := 3,
                hConnection             := fbTcpClientMK210.hConnection,
                xIsRtuOverTcpMode       := FALSE ,
                usiUnitId               := 1,
                eFuncCode               := OCL.MB_FC.WRITE_MULTIPLE_REGISTERS,
                uiDataAddr              := 470,
                uiDataCount             := 1,
                pData                   := ADR(wDoMask),
                szSize                  := SIZEOF(wDoMask)
            );

            IF fbMK210_301_DO.xDone OR fbMK210_301_DO.xError THEN
                // после выполнения блока его надо сбросить
                fbMK210_301_DO(xExecute := FALSE);

                // возвращаемся на шаг 0 для проверки состояния соединения
                iStateMK210 := 0;
            END_IF

END_CASE
```

Б4 Листинг примера из п. 5.9.4

```
PROGRAM PLC_PRG_ST
VAR

        fbRealToWord2: OCL.REAL_TO_WORD2;           // ФБ преобразования REAL в две…
                                                    // …переменные типа WORD

        fbModbusTcpSlave:   OCL.MB_TcpSlave;        // ФБ для реализиации Modbus Slave
        awSlaveData:        ARRAY [0..15] OF WORD;  // буфер данных Modbus Slave

        (* значения, полученные от OPC *)
        xVar_Opc:                       BOOL;
        wVar_Opc:                       WORD;
        rVar_Opc:                       REAL;
        sVar_Opc:                       STRING(15);

        (* значения для передачи в OPC *)
        xVar_Plc:                       BOOL;
        wVar_Plc:                       WORD;
        rVar_Plc:                       REAL;
        sVar_Plc:                       STRING(15);

        xWrite:         BOOL;   // команда записи данных из программы в регистры Modbus Slave
        fbWriteEdge:    R_TRIG; // триггер для однократной записи
END_VAR



// чтобы запустить пример на ST на контроллере требуется:
// 1. Удалить из задачи MainTask вызов программы PLC_PRG_CFC
// 2. Привязать к задаче MainTask вызов программы PLC_PRG_ST


fbModbusTcpSlave
(
        xEnable         := TRUE,
        sIpAddr         := '0.0.0.0',
        uiPort          := 502,
        usiUnitId       := 1,
        pData           := ADR(awSlaveData),
        szSize          := SIZEOF(awSlaveData)
);

// данные, полученные от OPC
xVar_Opc := awSlaveData[0].0;
wVar_Opc := awSlaveData[1];
rVar_Opc := OCL.WORD2_TO_REAL(awSlaveData[2], awSlaveData[3], FALSE);
```

```
OCL.SWAP_DATA( ADR(awSlaveData[4]), ADR(sVar_Opc), SIZEOF(sVar_Opc), TRUE, FALSE, FALSE );
// по команде записываем переменные из программы в регистры Modbus Slave
fbWriteEdge(CLK := xWrite);

IF fbWriteEdge.Q THEN

        awSlaveData[0].0 := xVar_Plc;
        awSlaveData[1]   := wVar_Plc;
        fbRealToWord2(rInput := rVar_Plc, wOutput1 => awSlaveData[2], wOutput2 => awSlaveData[3]);
        OCL.SWAP_DATA( ADR(sVar_Plc), ADR(awSlaveData[4]), SIZEOF(sVar_Plc), TRUE, FALSE, FALSE );

END_IF
```

Б5 Листинг примера из п. 5.9.5

```
PROGRAM PLC_PRG_ST
VAR
	fbModbusTcpSlave:	OCL.MB_TcpSlave;			// ФБ для реализации Modbus Slave
	awSlaveData:		ARRAY [0..15] OF WORD;  // буфер данных Modbus Slave

	// пути к файлам архивов
	asFilePath:		ARRAY [1..8] OF STRING := ['/mnt/ufs/home/root/Arc1.log', 7('')];
END_VAR


// чтобы запустить пример на ST на контроллере требуется:
// 1. Удалить из задачи MainTask вызов программы PLC_PRG_CFC
// 2. Привязать к задаче MainTask вызов программы PLC_PRG_ST

fbModbusTcpSlave
(
	xEnable				:= TRUE,
	sIpAddr				:= '0.0.0.0',
	uiPort				:= 502,
	usiUnitId			:= 1,
	pData				:= ADR(awSlaveData),
	szSize				:= SIZEOF(awSlaveData),
	pasFilePath			:= ADR(asFilePath),
	c_xReverseByteOrderFiles	:= TRUE,
	c_usiAmountBytes		:= 10
);
```