Протокол Modbus
в системах промышленной автоматизации

Оглавление

1. О курсе

1.1 Описание курса

Этот учебный курс посвящён протоколу Modbus, который используется для обмена данными в системах промышленной автоматизации и некоторых других областях.

Он основан на материалах внутреннего обучения компании ОВЕН, проведённого зимой 2024–2025 года. Поэтому в рассматриваемых в курсе примерах в основном используется оборудование ОВЕН.

Курс будет полезен:

- инженерам, разрабатывающим системы автоматизации, в которых используется протокол Modbus;
- инженерам, разрабатывающим устройства и ПО с поддержкой протокола Modbus;
- студентам профильных специальностей.

Курс состоит из следующих частей:

- обзор спецификации Modbus (*включая некоторые сведения об интерфейсе RS–485*), сопровождаемый примерами опроса конфигурируемых приборов ОВЕН;
- обзор реализации Modbus в программируемых устройствах ОВЕН;
- обзор реализации Modbus в панелях оператора ОВЕН;
- обзор использования Modbus совместно со шлюзами интерфейсов и протоколов;
- обзор реализации Modbus в ПО верхнего уровня на примере OPC–серверов и облачного сервиса OwenCloud;
- обзор примеров реальных приборов, в которых реализация Modbus не соответствует спецификации;
- рекомендации по реализации Modbus при разработке нового прибора.

Усвоение материалов курса будет затруднено при отсутствии:

- базовых навыков программирования;
- базового представления о компьютерных сетях;
- отсутствия возможности проводить эксперименты по настройке обмена с реальными приборами.

Если вы заметили опечатку/неточность или же у вас возник какой–то вопрос – то, пожалуйста, напишите об этом в комментариях к соответствующему уроку.

Желаем вам успехов!

1.2 Предварительная информация

Системы счисления

В рамках курса помимо десятичной системы счисления используются шестнадцатеричная (HEX) и двоичная.

Информация об этих системах приведена по ссылкам выше.

Шестнадцатеричная система используется при записи содержимого пакетов Modbus – как, например, ниже:

> *10 03 10 00 00 0D 83 8E*

В тексте уроков числа, записанные в шестнадцатеричной системе, обозначаются с префиксом **0x** (например: **0x10**) или постфиксом **h** (например: **10h**).

Двоичная система используется при работе с битовыми масками, которые в первый раз будут упомянуты в уроке 2.5.

1.3 Благодарности

Хотелось бы выразить признательность всем людям, которые помогли с вычиткой и доработкой данного курса:

- Александру Пинэко–Скворцову;
- Михаилу Троицкому;
- Электрошаману (Cs–Cs);
- Игорю Явкину;
- Арсению Виноградову;
- Андрею Цуканову;
- Никите Ларину.

2. Обзор спецификации Modbus

2.1 Флешфорвард

Изначально данный модуль должен был начинаться с урока 2.2 Протоколы и интерфейсы.

Но в процессе его подготовки быстро стало понятно, что изложенный там материал в первый момент времени может показаться очень «академическим», скучным и даже напрямую не связанным с темой курса.

Поэтому давайте лучше начнём с флешфорварда.

Флешфорвард – это кинематографический и литературный приём, суть которого сводится к следующему: в первой сцене произведения вы видите главного героя в активной (часто – экстремальной) фазе какой–либо ситуации, ещё не зная, как он в неё попал и как будет из неё выпутываться. Обычно это эмоционально цепляет гораздо больше, чем «хронологически первая» сцена, в которой протагонист проснулся, позавтракал и пошёл на работу.

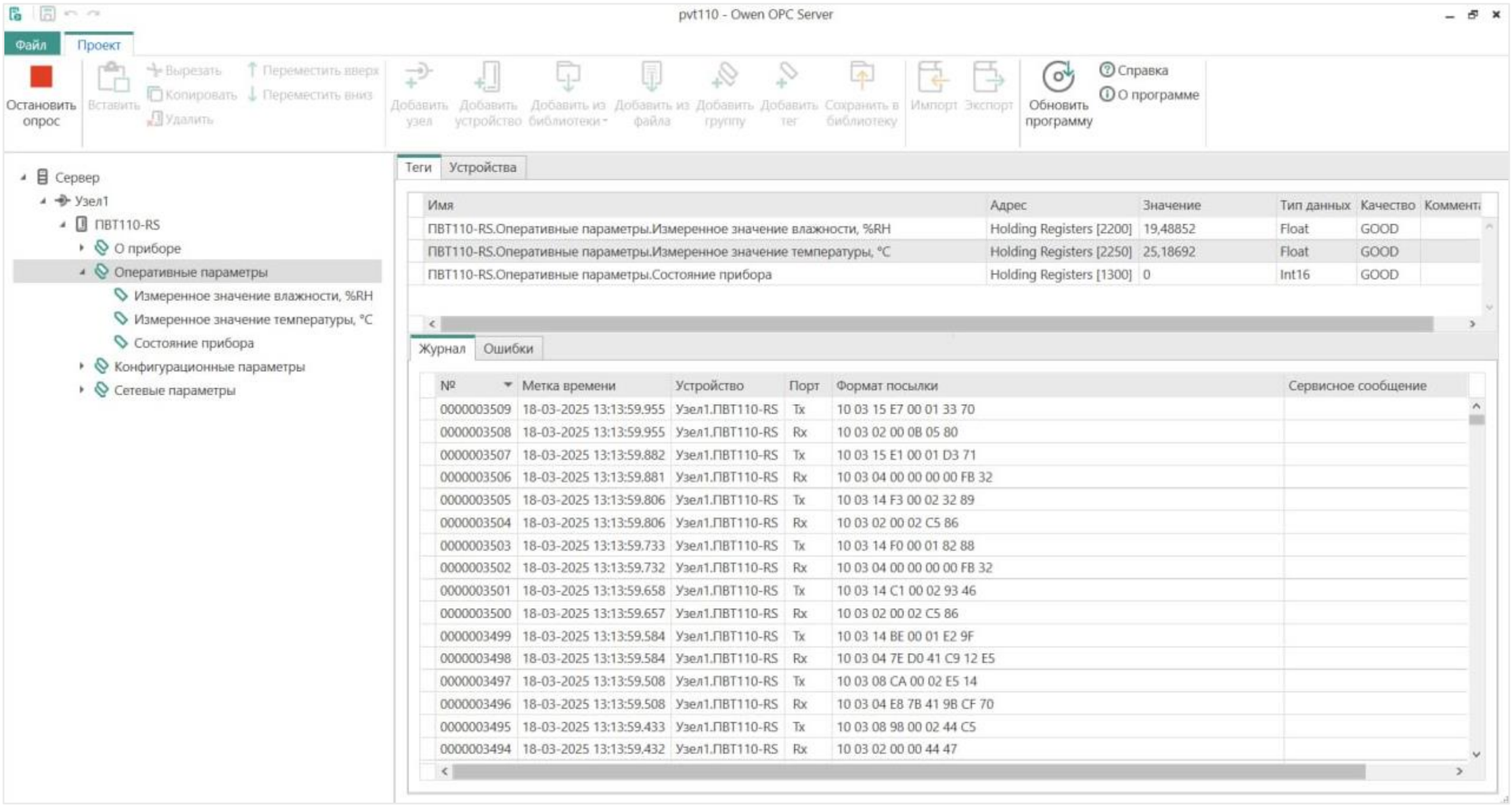
Помните первую главу «Бойцовского клуба» или первую сцену «Во все тяжкие»? (даже если не помните – то, вероятно, понимаете, о чём идёт речь).

Это и есть флешфорвард. Давайте и мы начнём наш курс с него:

На данном фото можно увидеть оборудование компании ОВЕН:

- датчик влажности и температуры ПВТ110–RS;
- блок питания БП30Б–Д3, от которого запитан этот датчик;
- преобразователь интерфейсов RS–485/USB АС4–М, с помощью которого датчик подключен к ноутбуку.

На ноутбуке запущена программа Owen OPC Server, которая производит опрос датчика по протоколу Modbus RTU – в частности, считывает с него значение влажности и температуры. В программе отображается лог обмена: он состоит из запросов, которые программа отправляет датчику, и ответов датчика на эти запросы.



В рамках курса мы рассмотрим, как формируются эти запросы и ответы, и чему соответствует каждый входящий в их состав байт.

Но перед этим давайте разберёмся, как собрать и запустить мини–стенд, показанный на фото.

В следующих шагах мы не будем обсуждать каждую деталь – вместо этого попробуем сначала пройти «по верхам», чтобы получить обобщённое представление о том, как происходит настройка обмена по протоколу Modbus. В конце урока мы сформулируем список вопросов, которые могут возникнуть к этому моменту, и будем последовательно разбираться с ними на протяжении всего модуля.

Оборудование, входящее в состав стенда

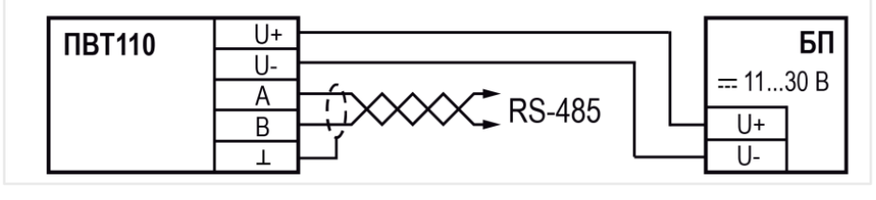
ПВТ110 – это промышленный датчик влажности и температуры, который является сертифицированным средством измерения (СИ). Модификация датчика ПВТ110–RS имеет встроенный интерфейс RS–485 с поддержкой протокола Modbus RTU, что позволяет подключать датчик к другим приборам.

> *Термин «интерфейс» будет рассмотрен в уроке 2.2.*
>
> *Информация об интерфейсе RS–485 будет приведена в уроке 2.14.*
>
> *Варианты протокола Modbus (в том числе, Modbus RTU) рассмотрены в уроке 2.3.*

Для обеспечения работы датчика требуется подключить к нему источник питания с напряжением в диапазоне 11...30 В.

В рамках стенда для этой цели используется блок питания БП30Б–Д3–24. Естественно, подойдёт и любой другой блок питания с аналогичными характеристиками.

Схема подключения питания приведена в руководстве на датчик:



На этой же схеме видны клеммы интерфейса RS–485 (A и B), а также клемма «общего провода» этого интерфейса, отмеченная пиктограммой заземления.

> *Мы поговорим про особенности маркировки клемм интерфейса RS–485 и «общий провод» в уроке 2.14.*

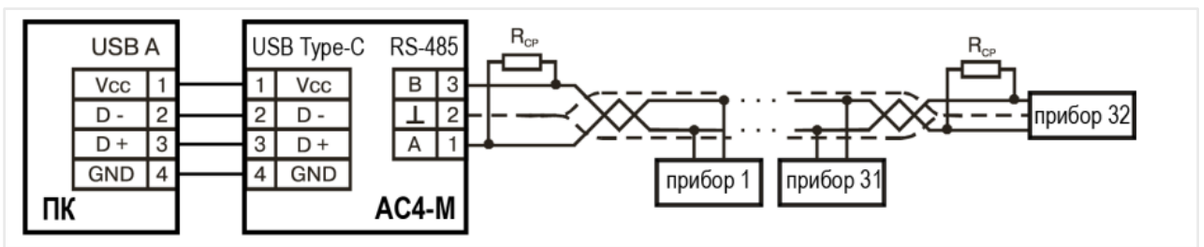
Мы можем использовать эти клеммы для подключения датчика к другому устройству, которое имеет интерфейс RS–485. В нашем случае мы хотим подключить датчик к ноутбуку – но у него такого интерфейса нет. Собственно, его нет у подавляющего большинства офисных и бытовых компьютеров; исключение составляют промышленные компьютеры и компьютеры, рассчитанные на крайне специфические области использования.

> *Пример промышленного компьютера от ПАО "ИНЭУМ им. И.С. Брука".*
>
> *Если вы работали с контроллерами Siemens – то, возможно, сталкивались с известной линейкой промышленных ноутбуков Simatic Field PG.*

Поэтому для подключения к «обычным» компьютерам устройств с интерфейсом RS–485 используется преобразователь интерфейсов (также часто называемый «конвертером» или «адаптером»). Этот прибор преобразует интерфейс RS–485 в интерфейс USB – а порты USB есть практически у всех современных компьютеров.

В рамках стенда используется конвертер АС4–М.

Вот так выглядит его схема подключения:



Для подключения датчика к конвертеру нужно соединить между собой их интерфейсные клеммы:

- клемму A датчика подключить к клемме A конвертера;
- клемму B датчика подключить к клемме B конвертера.

Конвертер подключается к компьютеру с помощью кабеля USB Type–C – USB A, который входит в комплект поставки.

Питание ему не требуется – он получает его от USB–порта компьютера.

> *До 1 января 2025 года конвертер АС4–М выпускался с интерфейсом Micro USB – и, соответственно, в комплект поставки входит кабель Micro USB – USB A.*

На приведённой выше схеме видно, что можно подключить к конвертеру несколько приборов, используя топологию «шина». Про эту топологию, а также про согласующие резисторы (Rcp на схеме) и ограничение в 32 подключаемых прибора мы поговорим в уроке 2.14.

Там же мы обсудим особенности маркировки клемм интерфейса RS–485 и другие вопросы.

Возможно, вы встречались и с другими конвертерами. Из проверенных и пользующихся нашим доверием стоит упомянуть Moxa Uport и серию ICP CON от компании ICP DAS. Достаточно популярны компактные конвертеры в формате USB flash–накопителя – например, конвертер компании Болид.

> *Существуют и конвертеры, преобразующие RS–485 в Ethernet. Мы поговорим о них в модуле 5 Шлюзы Modbus.*

Настройка датчика ПВ110–RS

Для подключения к датчику – нужно знать его сетевые настройки:

- скорость обмена;
- количество бит данных;
- режим контроля чётности;
- количество стоп–бит;
- адрес датчика.

Первые 4 настройки относятся к интерфейсу RS–485. Мы обсудим их в уроке 2.11.

Адрес является параметром, относящимся к протоколу Modbus. Мы обсудим его в уроке 2.5.

В руководстве на датчик приведены его настройки по умолчанию:

- скорость обмена: **9600** бит/с;
- количество бит данных – **8**;
- режим контроля чётности – **нет** (т. е. проверка бита чётности не выполняется);
- количество стоп–бит – **1**;
- адрес датчика – **16**.

Если настройки вашего датчика кто–то изменил – то вы можете узнать их, сняв его крышку (для этого потребуется аккуратно открутить 4 винта):

Разумно спроектированные приборы обычно имеют «ручной» способ изменения сетевых (и других) настроек, например:

- с помощью DIP– и поворотных переключателей, как это сделано у ПВТ110–RS;
- с помощью клавиш и дисплея;
- с помощью подключения по «отладочному» интерфейсу (например, USB, RS–232 или какому–то другому).

Если же прибор не имеет всего вышеперечисленного, и вы не знаете его текущих настроек – то для настройки связи вам потребуется использовать «метод перебора».

Определение номера виртуального COM–порта

Перед подключением конвертера RS–485/USB к компьютеру следует установить на компьютер его драйвер. Этот драйвер обычно выложен на сайте производителя конвертера. У используемого нами на стенде конвертера АС4–М он выложен на странице прибора.

После установки драйвера можно подключать конвертер к компьютеру. После этого перейдите в **Диспетчер устройств** Windows. В операционной Windows 10 это можно сделать следующим образом:

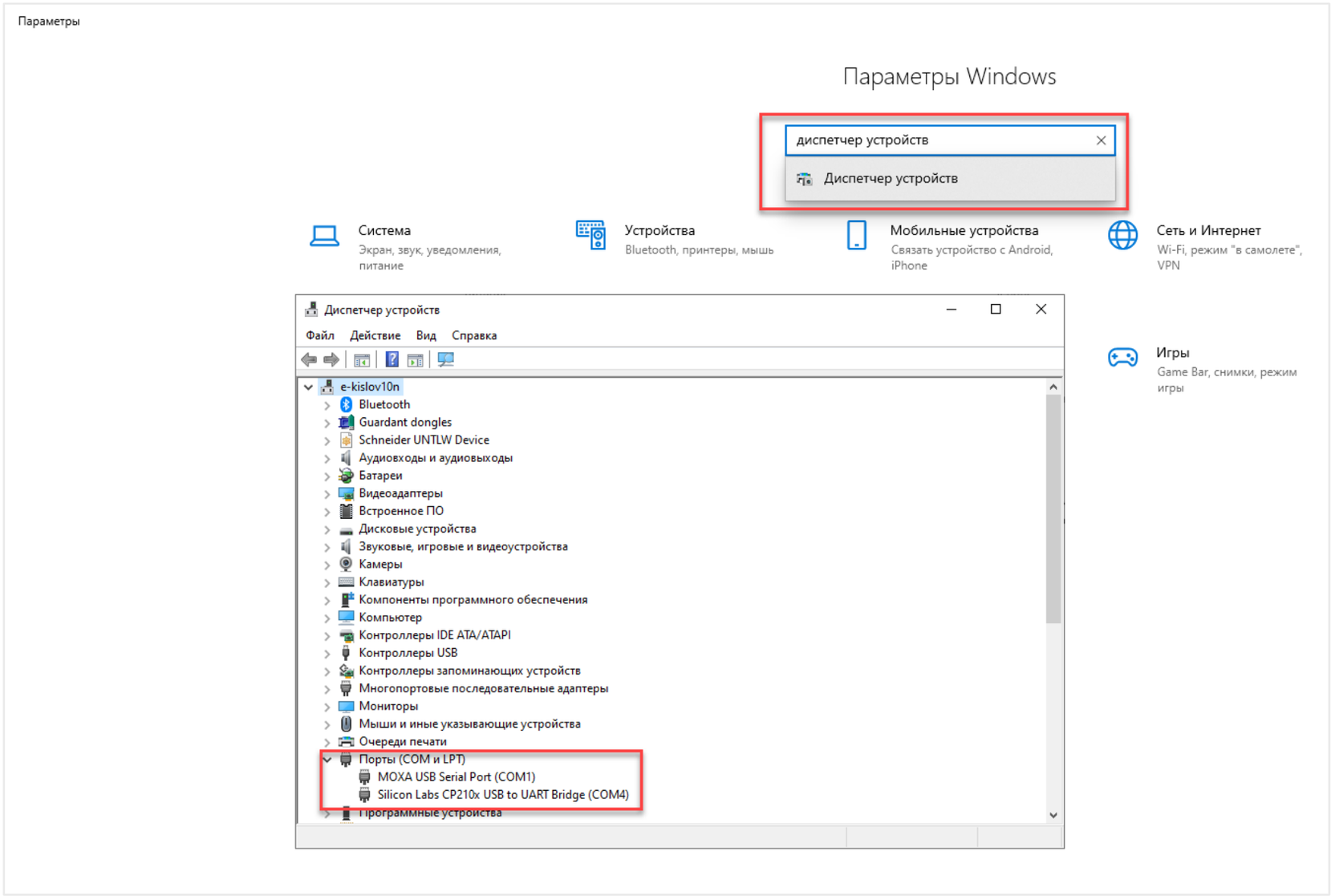
- кнопка **Пуск** – **Параметры ПК** – введите в строку поиска «диспетчер устройств».

В других версиях Windows последовательность действий может незначительно отличаться.

> *Мы не рассматриваем способ определения номера виртуального COM–порта в Linux и других операционных системах – в частности, по причине того, что большая часть используемого в курсе ПО запускается только под Windows; при необходимости вы можете найти эту информацию в Интернете.*

Во вкладке **Порты (COM и LPT)** будут отображаться номера COM–портов, присутствующих у вашего ПК. У нашего ноутбука есть два виртуальных COM–порта, созданные драйверами конвертеров интерфейсов:

- **COM1**, соответствующий конвертеру **Moxa Uport 1150**;
- **COM4**, соответствующий конвертеру **AC4–M** (запомним его номер – **4**; он потребуется нам далее).



Предположим, к вашему компьютеру подключено множество конвертеров AC4–M (вероятно, для их подключения вам потребовался USB–хаб с внешним активным питанием). Как определить, какой COM–порт какому конвертеру соответствует?

Очень просто – отключите один из конвертеров и отследите, какой COM–порт пропал из вкладки. Подключите его обратно – и убедитесь, что соответствующий COM–порт снова появился. При необходимости повторите эту процедуру для всех ваших конвертеров.

Owen OPC Server

Теперь настроим опрос нашего датчика в программе **Owen OPC Server**. Её можно загрузить с сайта ОВЕН (бесплатно и без регистрации):

https://owen.ru/product/new_opc_server

OPC–сервер – это программа, основной задачей которой является преобразование какого–либо протокола обмена в протокол OPC (*точнее, в один или несколько протоколов, описанных в спецификации технологии OPC*).

Технология OPC поддерживается большинством современных SCADA–систем (систем диспетчеризации, используемых для отображения информации о технологическом процессе, управления им, формирования журналов тревог, архивации данных и т. д.).

Как конвертер (АС4–М и т. п.) позволяет преобразовать специфичный для обычных компьютеров интерфейс RS–485 в «универсальный» интерфейс USB, так и OPC–сервер позволяет преобразовать «специфический» (с точки зрения SCADA) протокол в «универсальный» протокол технологии OPC.

> *Существуют SCADA–системы со встроенной поддержкой промышленных и других протоколов; но обычно и они поддерживают технологию OPC.*

Owen OPC Server поддерживает следующие протоколы:

- Modbus (RTU, ASCII и TCP; о вариантах Modbus мы поговорим в уроке 2.3);
- ОВЕН (устаревший протокол, поддерживаемый только приборами компании ОВЕН; в настоящее время практически не используется);
- протокол связи с облачным сервисом OwenCloud.

С точки зрения спецификации OPC – Owen OPC Server поддерживает только протокол OPC DA. В настоящее время этот протокол признан устаревшим; ему на смену пришёл OPC UA, который с каждым годом всё шире распространяется в системах автоматизации. Одним из недостатков OPC DA является его привязка к проприетарным технологиям компании Microsoft (OLE и DCOM), из–за чего его можно использовать только в рамках операционной системы Windows – поэтому Owen OPC Server может запускаться только на ПК с этой ОС.

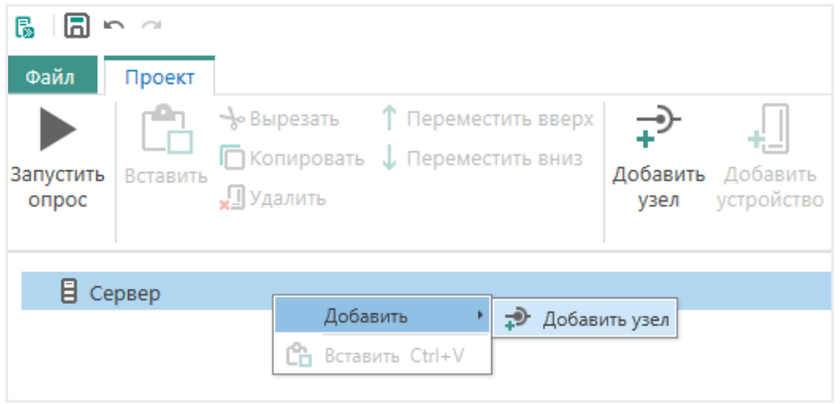
Но есть у Owen OPC Server и плюсы:

- уже упомянутая бесплатность; программа не требуется регистрации и не имеет ограничения на период работы.
- набор готовых шаблонов практически для всех приборов ОВЕН; мы воспользуемся таким шаблоном в следующем шаге;
- возможность импорта списков параметров из ПО, используемого для программирования устройств ОВЕН – **Owen Logic** и **CoDeSys V2.3**.

Настройка опроса в Owen OPC Server

Установите Owen OPC Server и запустите его.

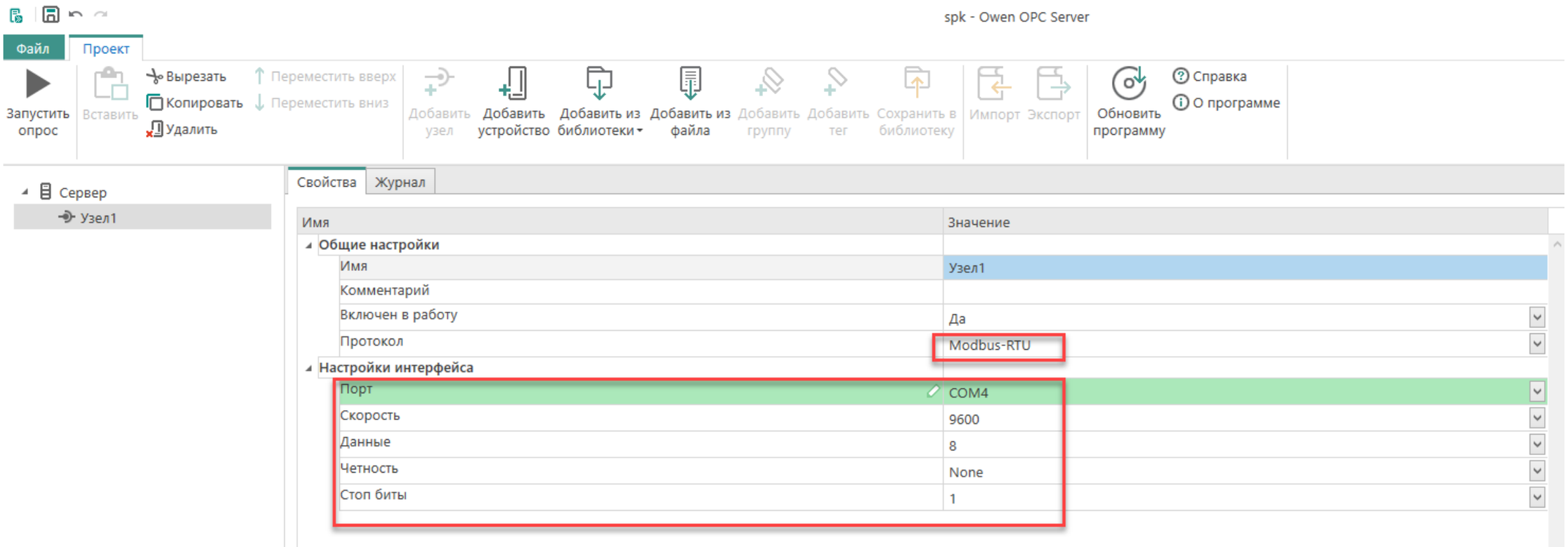
Нажмите правой кнопкой мыши на узел **Сервер** и добавьте новый узел.



В настройках этого узла укажите:

- протокол обмена; напомним, датчик ПВТ110 поддерживает только один протокол – **Modbus RTU**;
- номер виртуального COM–порта компьютера. В позапрошлом шаге мы видели, что конкретно у нас он имеет номер **4**;
- сетевые настройки COM–порта. Они должны совпадать с аналогичными настройками, заданными в датчике.

Мы используем датчик, в котором сетевые настройки имеют значения по умолчанию – поэтому на скриншоте ниже используются именно такие значения.

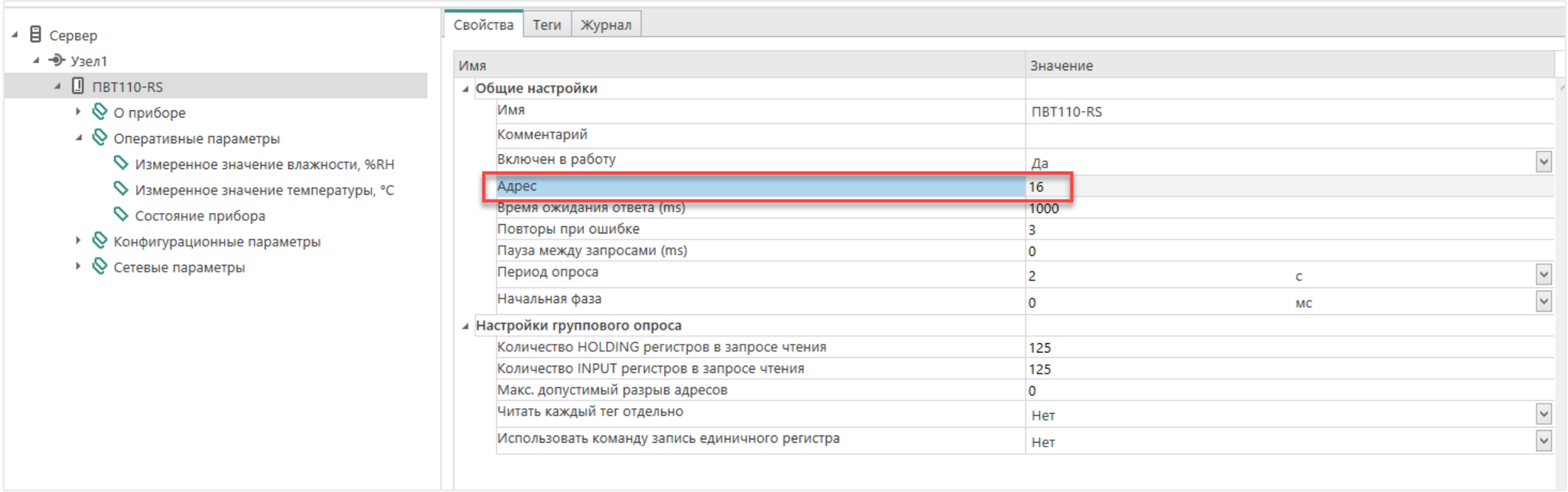


Теперь нажмите правой кнопкой мыши на наш узел и используйте команду **Добавить устройство** – **Устройство из библиотеки** – **Датчики** – **ПВТ110–RS**.

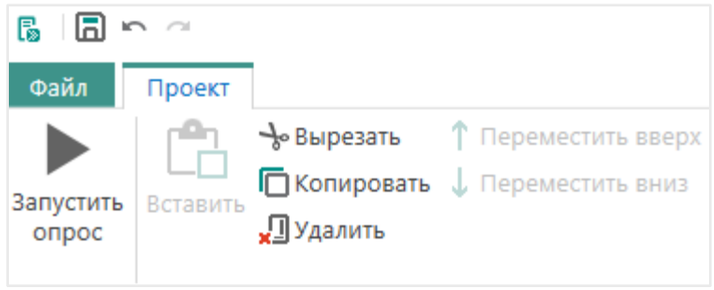
> *Чтобы добавить ещё одно устройство (например, ещё один датчик) потребуется повторно выполнить команду **Добавить устройство**.*

После этого в дерево проекта будет добавлено устройство **ПВТ110–RS**, содержащее несколько папок («О приборе», «Оперативные параметры», «Конфигурационные параметры», «Сетевые параметры»). Каждая папка содержит ряд параметров датчика.

В настройках устройства нас сейчас интересует только один параметр – адрес. Его значение должно соответствовать адресу, который задан для данного датчика. Напомним – у нашего датчика настройки по умолчанию, так что его адрес – **16**.



Теперь нажмите кнопку **Запустить опрос**...



...и созерцайте ту же картину, что и на скриншоте в самом начале первого шага нашего урока.

Круг замкнулся, и мы вернулись к тому моменту, который видели во флешфорварде.

pvt110 - Owen OPC Server

Файл | Проект

Остановить опрос | Вставить | Вырезать | Копировать | Удалить | Переместить вверх | Переместить вниз | Добавить узел | Добавить устройство | Добавить из библиотеки | Добавить из файла | Добавить группу | Добавить тег | Сохранить в библиотеку | Импорт | Экспорт | Обновить программу | Справка | О программе

- Сервер
  - Узел1
    - ПВТ110-RS
      - О приборе
      - Оперативные параметры
        - Измеренное значение влажности, %RH
        - Измеренное значение температуры, °C
        - Состояние прибора
      - Конфигурационные параметры
      - Сетевые параметры

Теги | Устройства

Имя	Адрес	Значение	Тип данных	Качество	Коммент
ПВТ110-RS.Оперативные параметры.Измеренное значение влажности, %RH	Holding Registers [2200]	19,48852	Float	GOOD	
ПВТ110-RS.Оперативные параметры.Измеренное значение температуры, °C	Holding Registers [2250]	25,18692	Float	GOOD	
ПВТ110-RS.Оперативные параметры.Состояние прибора	Holding Registers [1300]	0	Int16	GOOD	

Журнал | Ошибки

№	Метка времени	Устройство	Порт	Формат посылки	Сервисное сообщение
0000003509	18-03-2025 13:13:59.955	Узел1.ПВТ110-RS	Tx	10 03 15 E7 00 01 33 70	
0000003508	18-03-2025 13:13:59.955	Узел1.ПВТ110-RS	Rx	10 03 02 00 0B 05 80	
0000003507	18-03-2025 13:13:59.882	Узел1.ПВТ110-RS	Tx	10 03 15 E1 00 01 D3 71	
0000003506	18-03-2025 13:13:59.881	Узел1.ПВТ110-RS	Rx	10 03 04 00 00 00 00 FB 32	
0000003505	18-03-2025 13:13:59.806	Узел1.ПВТ110-RS	Tx	10 03 14 F3 00 02 32 89	
0000003504	18-03-2025 13:13:59.806	Узел1.ПВТ110-RS	Rx	10 03 02 00 02 C5 86	
0000003503	18-03-2025 13:13:59.733	Узел1.ПВТ110-RS	Tx	10 03 14 F0 00 01 82 88	
0000003502	18-03-2025 13:13:59.732	Узел1.ПВТ110-RS	Rx	10 03 04 00 00 00 00 FB 32	
0000003501	18-03-2025 13:13:59.658	Узел1.ПВТ110-RS	Tx	10 03 14 C1 00 02 93 46	
0000003500	18-03-2025 13:13:59.657	Узел1.ПВТ110-RS	Rx	10 03 02 00 02 C5 86	
0000003499	18-03-2025 13:13:59.584	Узел1.ПВТ110-RS	Tx	10 03 14 BE 00 01 E2 9F	
0000003498	18-03-2025 13:13:59.584	Узел1.ПВТ110-RS	Rx	10 03 04 7E D0 41 C9 12 E5	
0000003497	18-03-2025 13:13:59.508	Узел1.ПВТ110-RS	Tx	10 03 08 CA 00 02 E5 14	
0000003496	18-03-2025 13:13:59.508	Узел1.ПВТ110-RS	Rx	10 03 04 E8 7B 41 9B CF 70	
0000003495	18-03-2025 13:13:59.433	Узел1.ПВТ110-RS	Tx	10 03 08 98 00 02 44 C5	
0000003494	18-03-2025 13:13:59.432	Узел1.ПВТ110-RS	Rx	10 03 02 00 00 44 47	

Но что, если для вашего устройства нет шаблона?

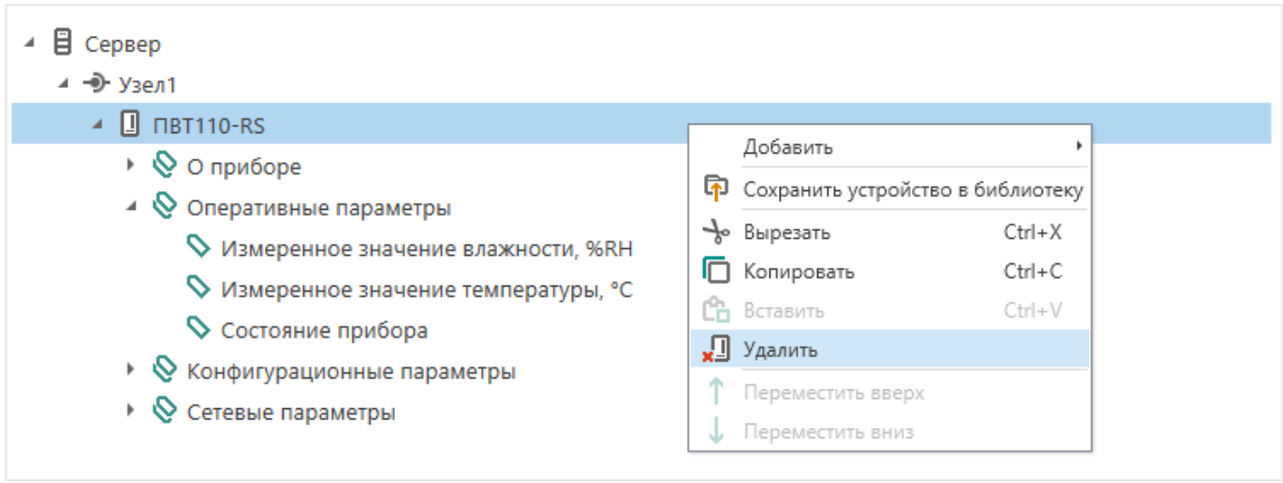
Мы добавили наш датчик в Owen OPC Server с помощью готового шаблона.

Как уже упоминалось – в Owen OPC Server есть шаблоны для большинства приборов ОВЕН.

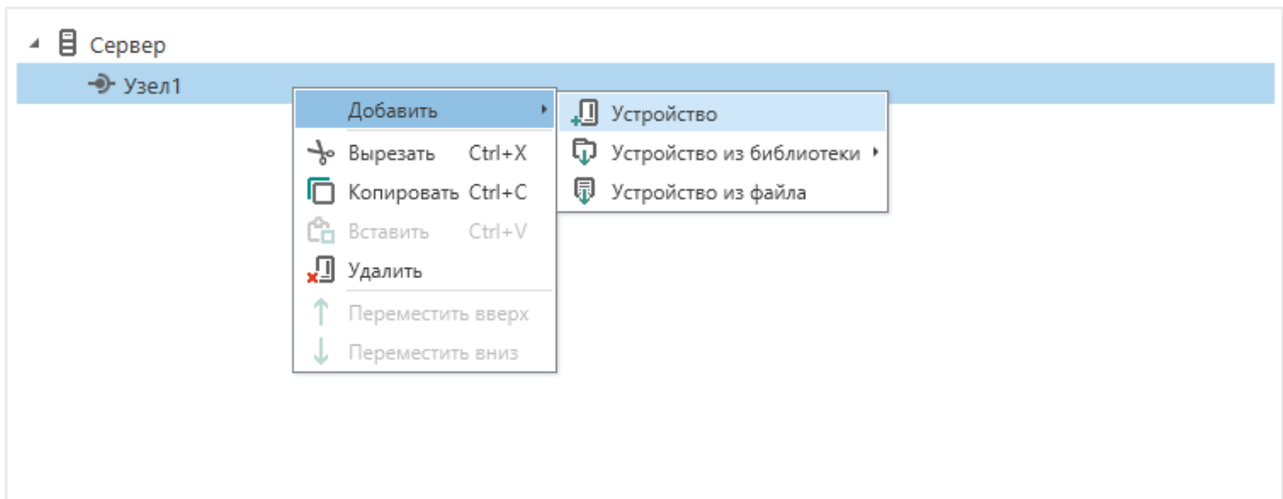
Но что, если вам потребуется настроить опрос с устройством, для которого шаблон отсутствует?

Давайте представим, что в OPC–сервере отсутствует шаблон датчика ПВТ110–RS и попробуем как–то выкрутиться из этой гипотетической ситуации.

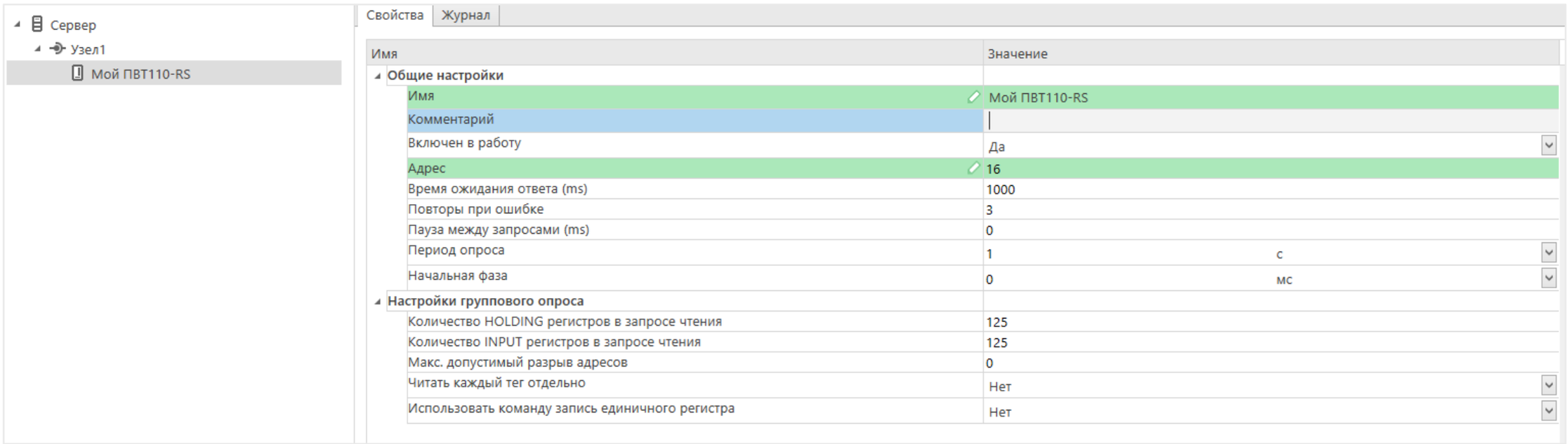
Для начала удалите добавленное ранее «шаблонное» устройство:



Теперь нажмите на узел правой кнопкой мыши и в этот раз используйте команду **Добавить** – **Устройство**.

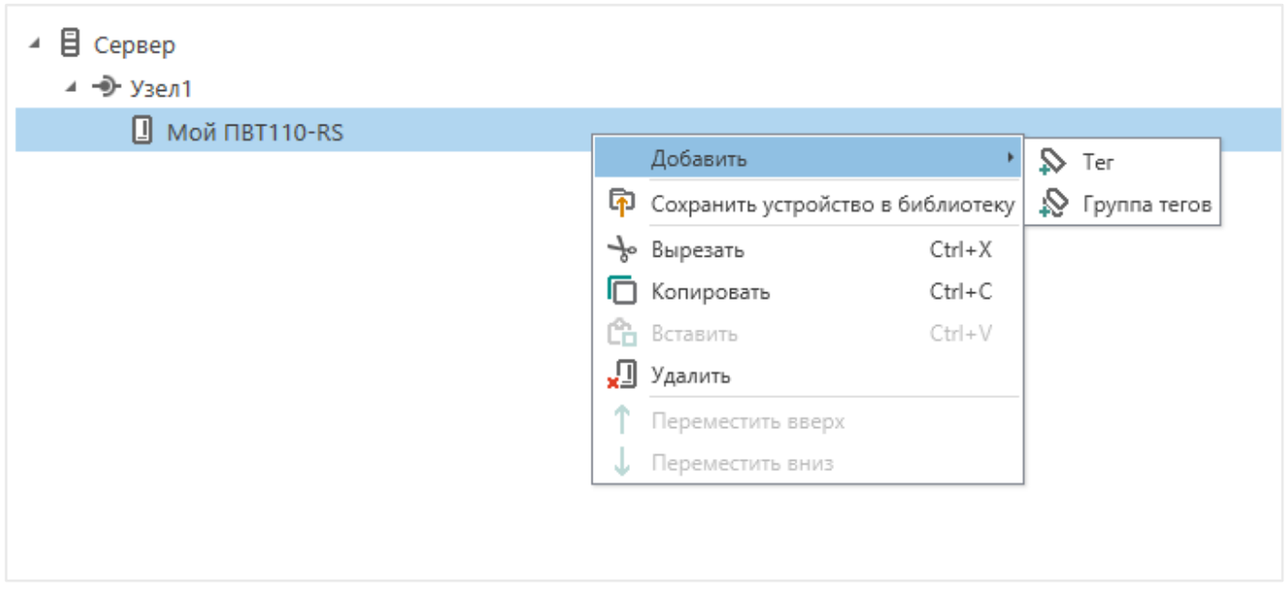


Как и в случае использования шаблона – в настройках устройства потребуется задать его адрес. Опять введите **16**. Вы также можете изменить имя устройства (имя по умолчанию – «Устройство1»), и оно сразу же изменится в дереве конфигурации.



Нажмите на устройство правой кнопкой мыши и добавьте в него тег.

Тег – это общепринятый эквивалент термина «параметр», используемый в OPC–серверах, SCADA–системах и другом ПО верхнего уровня. Группа тегов – это папка в дереве конфигурации. Мы видели такие в шаблоне, помните? («Оперативные параметры» и т. д.)



У тега довольно много настроек. Чтобы правильно задать их – нужно открыть руководство на датчик и найти информацию об интересующем нас параметре (например – об измеренной температуре):

ВНИМАНИЕ
Гайку кабельного ввода следует заворачивать до упора.
При несоблюдении данного условия производитель не может гарантировать соответствие стандарту IP65.

8.3 Назначение контактов клеммника

Схема подключения прибора приведена на *рисунке 8.2*.

ВНИМАНИЕ
Во время подключения источника питания требуется соблюдать полярность!
Неправильное подключение может привести к порче оборудования.

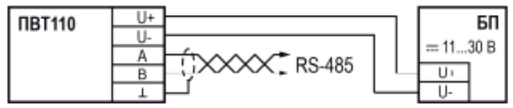


Рисунок 8.2 – Схема подключения

9 Эксплуатация

9.1 Включение и работа

Во время работы прибор проверяет исправность подключенного измерительного зонда. Состояние прибора индицируется светодиодом «Статус» и передается в регистре «Состояние прибора», см. *п. 9.2– 9.3*.

9.2 Работа по интерфейсу RS-485

Прибор работает в режиме Slave по протоколу ModBus RTU.

Список параметров, доступных по сети RS-485, приведен в таблице ниже.

Таблица 9.1 – Параметры прибора, доступные по RS-485

Наименование параметра	Номер первого регистра		Кол-во регистров	Тип	Допустимые значения*	Тип доступа
	DEC	HEX				
Общие параметры						
Название датчика	1000	3E8	6	STRING [12]	PVT110	RO
Версия ПО	1006	3EE	3	STRING[6]	01.00 … 99.99	RO
Заводской номер	1104	450	10	STRING [20]	XXXXXXXXXXXXXXXXX	RO
Состояние датчика	1300	514	1	UC8	см. *регистр 0x0514*	RO
Управление прибором						
Команда управления	1400	578	1	UC8	bit[0] = 1 – программная перезагрузка прибора; bit[1] = 1 – сброс всех настроек на заводские	WO
Оперативные параметры						
Значение влажности, %RH	2200	898	2	FLOAT32	0,00…100	RO
Значение температуры, °C	2250	8CA	2	FLOAT32	-40,00…80,00	RO

Прибор поддерживает выполнение следующих функций ModBus:
- **03** – чтение значений из нескольких регистров хранения;
- **06** – запись значения в один регистр хранения;
- **16** – запись значения в несколько регистров хранения.

Прибор поддерживает коды ошибок ModBus:
- **01** – принятый код функции не может быть обработан;
- **02** – адрес данных, указанный в запросе, не доступен;
- **03** – величина, содержащаяся в поле данных запроса, является недопустимой.

9.3 Индикация

Светодиод расположен внутри электронного блока прибора.

Таблица 9.2 – Назначение светодиода

Свето-диод	Статус	Значение	
◌	Зеленый, непрерывно светится	Нормальная работа прибора	
●	Красный, непрерывно светится	Отсутствует связь с зондом	
✳	Зеленый, непрерывно мигает	Выход за верхний предел измерения температуры	
✳	Красный, непрерывно мигает	Ошибочная конфигурация переключателя четности	
✳	Зеленый, быстро мигает	**Мигает на протяжении 0,5 с**	**Мигает на протяжении 1 с**
		Успешный прием пакета по RS-485	Подтверждение смены ручных настроек
✳	Красный, быстро мигает на протяжении 0,5 с	Ошибка при приеме пакета по RS-485	

10 Техническое обслуживание

Во время выполнения работ по техническому обслуживанию прибора следует соблюдать требования безопасности из раздела 5.

Техническое обслуживание прибора следует проводить не реже одного раза в 6 месяцев. Техническое обслуживание включает в себя следующие процедуры:
- проверка качества крепления прибора;
- проверка качества подключения внешних связей;
- удаление пыли и грязи с корпуса и клеммника прибора.

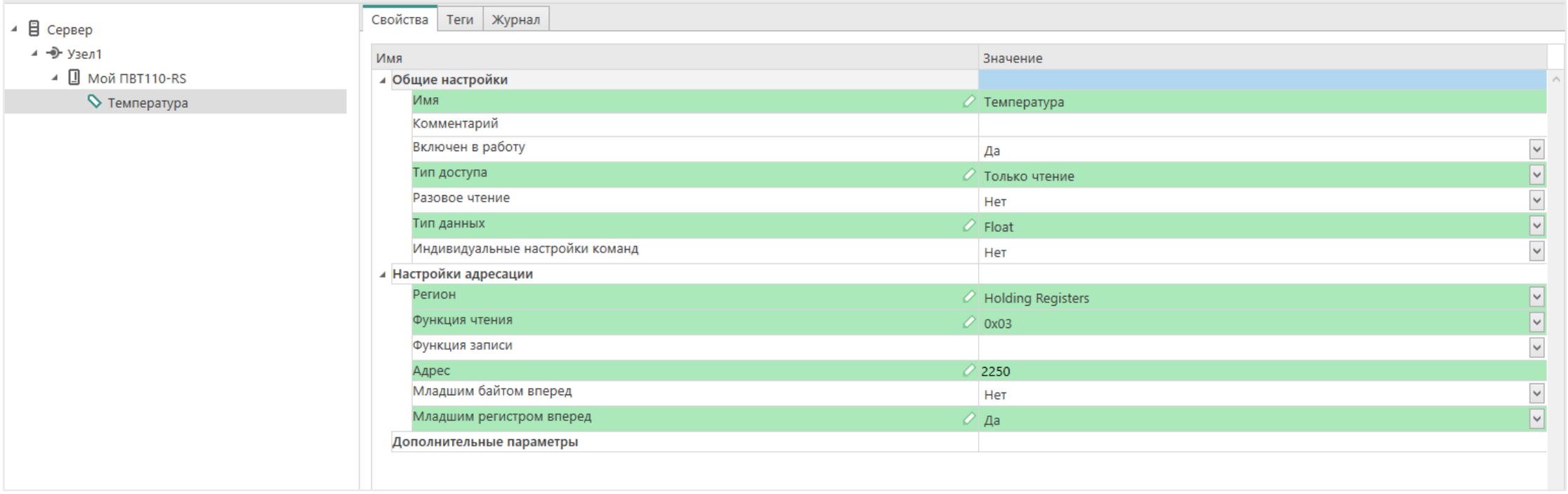
Обнаруженные при осмотре недостатки следует немедленно устранить.

Межповерочный интервал прибора – 1 год.

11 Маркировка

На корпус прибора нанесены:
- товарный знак;
- наименование и исполнение прибора;
- степень защиты корпуса по ГОСТ 14254-2015;
- напряжение питания;
- потребляемая мощность;

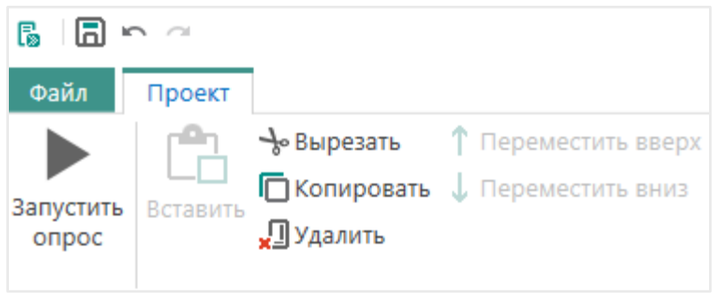
Используя выделенные красным фрагменты – вы можете ввести правильные значения почти всех нужных настроек:



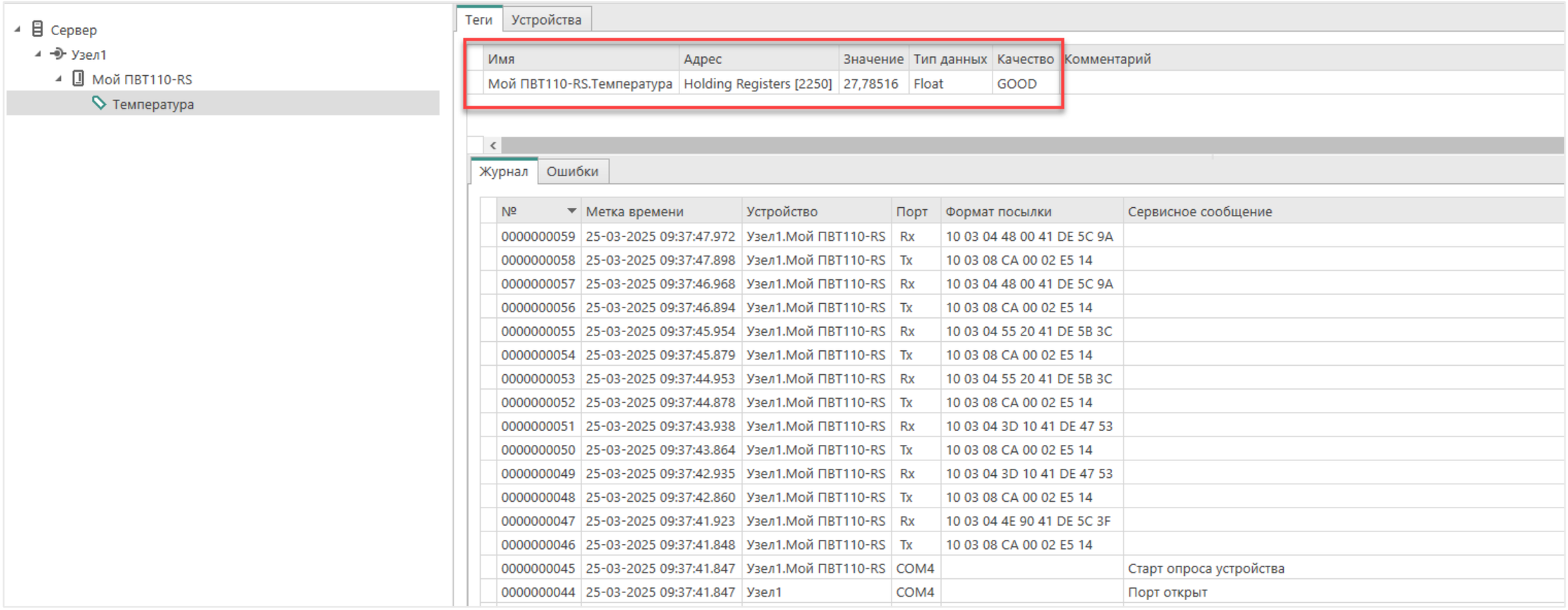
Исключение составляет параметр **Младшим регистром вперёд** – его нужно установить в значение **Да**. Вы не сможете найти эту информацию в документации на датчик. Мы ещё не раз будем обсуждать этот момент, а пока достаточно попробовать запомнить, что спецификация Modbus не определяет требования к порядку регистров в передаваемых значениях.

> *Тип доступа **RO** означает **read only** (только чтение). Это логично – температура измеряется датчиком, и её можно лишь прочитать, но нельзя записать.*
>
> *Нам не потребовалось указывать количество считываемых регистров, потому что OPC–сервер определяет их автоматически на основании типа параметра.*
>
> *Ввод адреса параметра в OPC–сервере осуществляется в десятичном формате (DEC).*
>
> *Понятие региона, связанное с областями памяти Modbus, мы обсудим в уроке 2.5.*

Нажмите кнопку **Запустить опрос**...



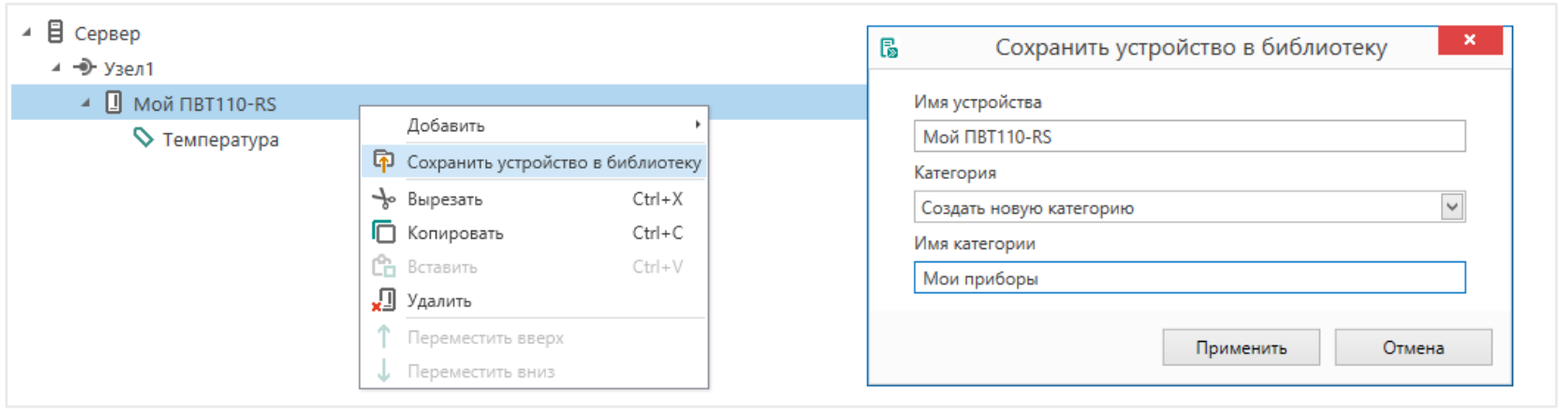
...и вы увидите корректное значение считанной из датчика температуры.



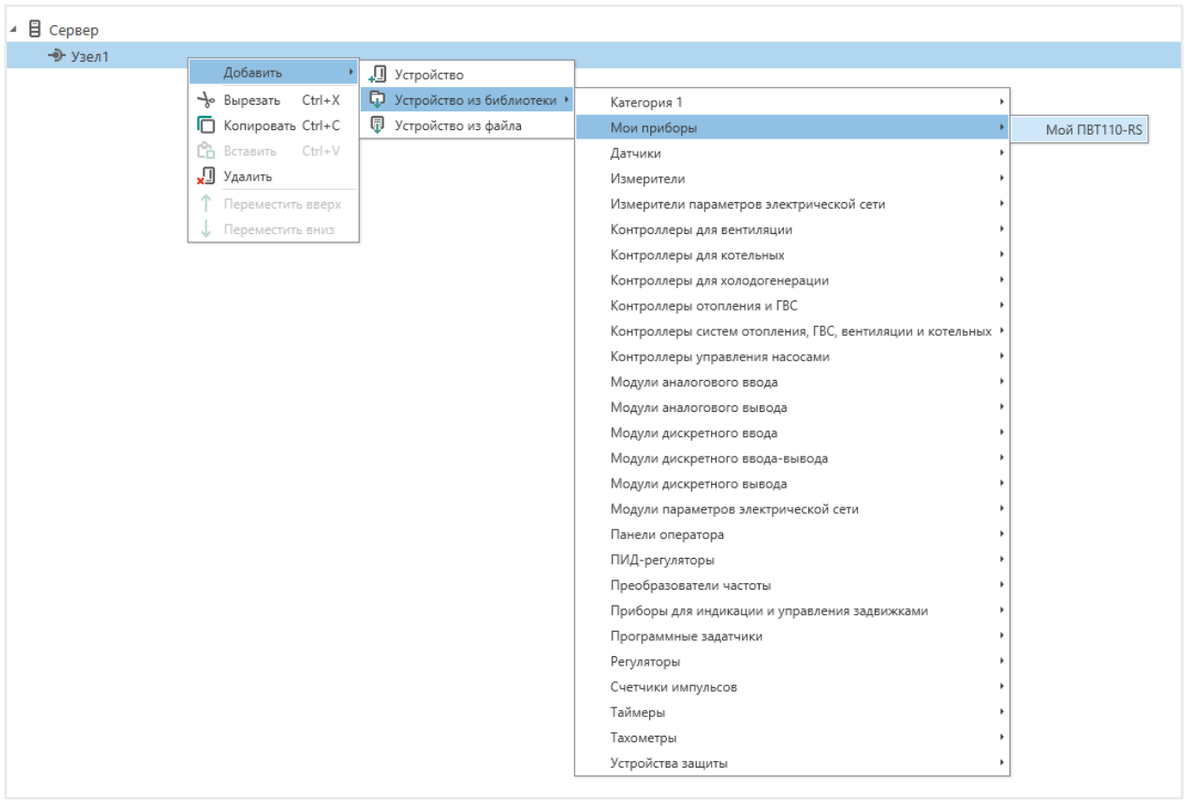
Экспорт и импорт шаблонов

Вы можете сохранить настроенное вами устройство (со всеми его тегами) в библиотеку OPC–сервера.

Для этого нажмите на устройство правой кнопкой мыши и используйте соответствующую команду. Вы можете поместить создаваемый шаблон в одну из уже существующих в OPC–сервере категорий (мы видели их в шаге 6 при добавлении шаблона нашего датчика) или создать новую категорию.



После сохранения в библиотеку – вы сможете добавить своё устройство как шаблон:



Но что, если вам потребуется переслать этот шаблон на другой компьютер?

Для этого потребуется найти его файл. Файлы пользовательских шаблонов сохраняются в директории

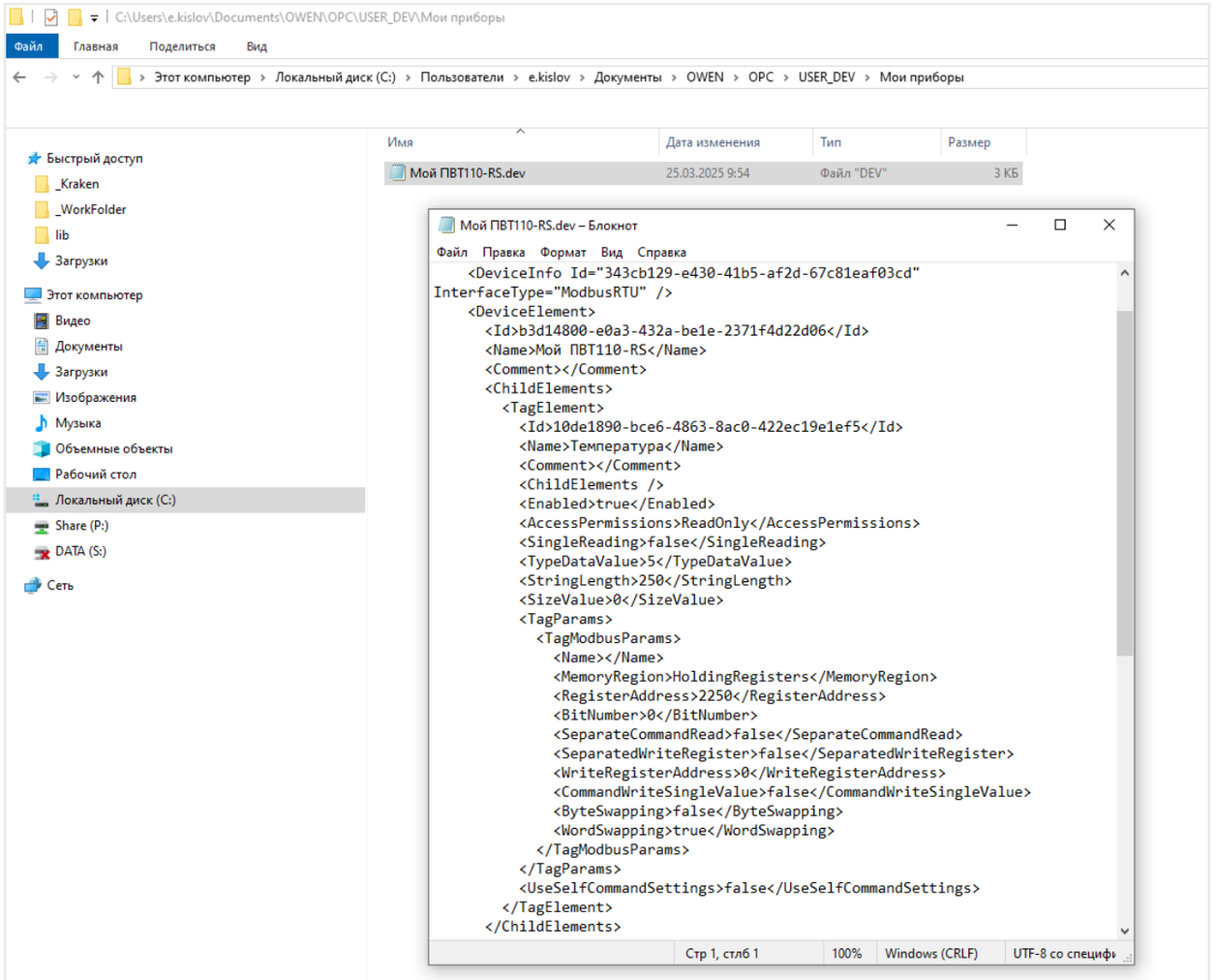
> ***C:\Users\<ваш_пользователь>\Documents\OWEN\OPC\USER_DEV***

и имеют расширение .dev.

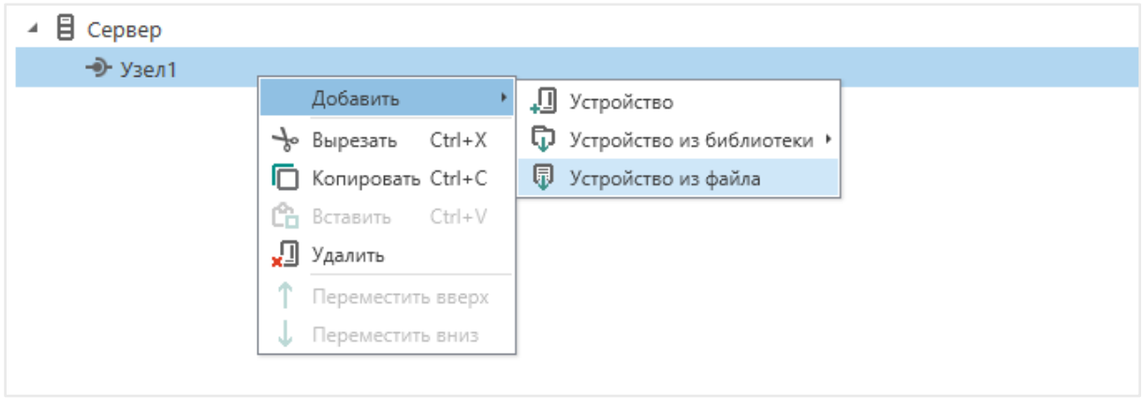
> *Фактически шаблоны представляют собой файлы формата XML.*

Созданный в прошлом шаге шаблон доступен по ссылке:

https://ftp.owen.ru/ModbusCourse/01_OpcTemplates/%CC%EE%E9%20%CF%C2%D2110–RS.dev



Вы можете переслать этот файл на другой ПК, открыть на нём Owen OPC Server, добавить узел с протоколом Modbus RTU, нажать на него правой кнопкой мыши и использовать команду **Добавить** – **Устройство из файла**, указав путь к вашему файлу с расширением .dev.



Вопросы, которые могут возникнуть после флешфорварда

Предположим, что до прохождения данного урока вы не имели существенного опыта использования протокола Modbus.

Тогда к этому моменту у вас уже может накопиться список вопросов (и это совершенно закономерно). Например:

- почему на схеме подключения устройств к конвертеру АС4–М, показанных в шаге 2, указано ограничение в 32 устройства? Как именно подключить несколько устройств к конвертеру и что за резистор Rcp изображён на этой схеме?
- можно ли обойтись без АС4–М и датчика, произведя их эмуляцию каким–то образом на ПК?
- в чём смысл каждой из настроек COM–порта, рассмотренных в шаге 3? (и что из себя в принципе представляет COM–порт?);
- чем отличаются различные виды протокола Modbus, упомянутые в шаге 5?
- за что отвечает каждая из настроек устройства, которые показаны на скриншоте в шаге 6?
- как читать логи обмена, отображаемые на скриншотах в шагах 1 и 8?
- за что отвечает каждая из настроек тега, которые показаны на скриншоте в шаге 7?
- что такое «функция Modbus»? Какие бывают функции и чем они отличаются"?
- за что отвечает настройка тега **Младшим регистром вперёд** и почему она не была описана в документации на датчик?

Все эти вопросы будут рассмотрены в дальнейших уроках данного модуля.

Если вы уже имели опыт использования протокола Modbus и внимательно изучили руководство на датчик ПВТ110–RS, то можете спросить:

- как используется его параметр «**Последовательность байт в двухбайтовых данных**»?
- как именно обрабатывается параметр «**Таймаут ответа**»?
- каково назначение параметров «**Аварийное значение влажности**» и «**Аварийное значение температуры**»?

Аварийное значение влажности, %RH	5313	14C1	2	FLOAT32	-100…**0**…100	RW
Верхний предел измерения температуры, °C	5352	14E8	2	FLOAT32	80,00	RO
Нижний предел измерения температуры, °C	5354	14EA	2	FLOAT32	-40,00	RO
Постоянная времени фильтра температуры, с	5360	14F0	1	UC8	**0**...100	RW
Аварийное значение температуры, °C	5363	14F3	2	FLOAT32	-100…**0**…100	RW
Параметры интерфейса						
Последователь-ность байт в двухбайтовых данных	5601	15E1	1	UC8	**11** – старший байт первый 12 – младший байт первый	RW
Сетевой адрес	5602	15E2	1	UC8	1…**16**…99	RO
Скорость обмена (в бодах)	5603	15E3	1	UC8	2 – 2400 3 – 4800 **4 – 9600** 5 – 14400 6 – 19200 7 – 28800 8 – 38400 9 – 56000 10 – 57600 11 – 115200	RO
Количество бит данных	5604	15E4	1	UC8	7/**8**	RO
Контроль чётности	5605	15E5	1	UC8	**0** – нет 1 – чётный 2 – нечётный	RO
Количество стоп-битов	5606	15E6	1	UC8	**0** – 1 1 – 1,5 2 – 2	RW
Таймаут ответа, мс	5607	15E7	2	UC16	1...**100**...1000	RW

Это хорошие вопросы. Мы ответим на них в последнем шаге урока 2.9.

Но давайте начнём с вопроса, который касается базовых понятий:

- в прошлых шагах регулярно использовались термины «**протокол**» и «**интерфейс**». Что именно под ними подразумевается?

2.2 Протоколы и интерфейсы

Формальные определения

Что же это такое – Modbus?

> *Modbus – это семейство протоколов обмена, используемых в системах промышленной автоматизации и некоторых других областях.*

Ключевым термином этого определения является **протокол обмена** (который также можно назвать коммуникационным протоколом, протоколом передачи данных и т. д.).

Если заглянуть в статью Протокол передачи данных на Википедии, то можно найти такую формулировку:

> *Протокол передачи данных – это набор соглашений интерфейса логического уровня, которые определяют обмен данными между различными программами. Эти соглашения задают единообразный способ передачи сообщений и обработки ошибок при взаимодействии программного обеспечения, разнесённого в пространстве аппаратуры, соединённой тем или иным интерфейсом.*

Она не выглядит интуитивно понятной; давайте сразу выделим ключевой термин и этого определения – интерфейс – и посмотрим, что Википедия расскажет нам о нём:

> *Интерфейс – граница между двумя функциональными объектами, требования к которой определяются стандартом; совокупность средств, методов и правил взаимодействия (управления, контроля и т. д.) между элементами системы.*

Что ж, такое определение тоже выглядит достаточно туманным. Вообще, эта цитата состоит из двух разных определений, позаимствованных из двух различных документов (если интересно, каких именно – то перейдите по ссылке).

Сложность для восприятия – это традиционная черта формальных определений.

Но без паники! Давайте возьмём эти термины («протокол», «интерфейс» и – он нам тоже пригодится – «стандарт») и попробуем раскрыть их более простыми словами.

Интерфейсы

Начнём с интерфейса.

Применительно к рассматриваемой нами области (напомним, мы говорим про обмен данными между различным устройствами) вполне можно сказать так:

> *Интерфейс – это описание способа передачи данных с точки зрения законов физики.*

Это неидеальное определение, но зато от него можно легко перейти к понятным ассоциациям.

Интерфейс определяет среду передачи – например, ей может быть кабель или радиоэфир.

Интерфейс определяет тип сигнала – например, он может представлять собой электрические импульсы, свет или радиоволны.

И, как следствие, интерфейс определяет все «физические» аспекты передачи данных:

- тип, длину и другие характеристики кабеля связи (для «проводных» интерфейсов);
- характеристики сигнала (амплитуда, частота, фаза, модуляция и т. д.).

Всё перечисленное формирует ограничения интерфейса:

- максимальное расстояние, на которое можно передать данные;
- максимальную скорость передачи данных;
- максимальное количество подключаемых устройств;
- степень устойчивости к различного рода помехам;
- и так далее.

При этом интерфейс никак не определяет **содержимое** передаваемых данных и принцип их кодирования.

В интерфейсах обычно хорошо разбираются инженеры–схемотехники.

Самые «общеизвестные» интерфейсы, о которых вы точно слышали – это Ethernet, USB, Wi–Fi, Bluetooth и т. д.

Строго говоря, это не просто интерфейсы, а коммуникационные стандарты, которые используют те или иные интерфейсы (например, интерфейс передачи данных для Wi–Fi описан в стандарте IEEE 802.11).

В области промышленной автоматизации широко используются такие интерфейсы, как RS–485 (который мы рассмотрим в уроке 2.10) и CAN; но кроме них используются и упомянутые выше интерфейсы – особенно Ethernet.

Если вы занимаетесь программированием микроконтроллеров – то наверняка вам известны интерфейсы SPI и I^2C.

С интерфейсами разобрались; давайте переходить к протоколам.

Протоколы

В прошлом шаге мы сказали, что

> *Интерфейс никак не определяет **содержимое** передаваемых данных и принцип их кодирования.*

И теперь можно добавить, что это является задачей протокола:

> *Протокол обмена – это набор правил и соглашений, которые определяют **способ** обмена данными между устройствами или между различными программами одного устройства.*

Если интерфейс отвечает за «физический» аспект передачи данных, то протокол обеспечивает «программный».

Поэтому в протоколах обычно хорошо разбираются программисты.

Протокол определяет:

- принцип передачи данных (циклический, событийный или какой–то иной);
- систему адресации устройств;
- структуру и формат передаваемых сообщений;
- а также структуру и формат пакетов, в которые объединяются сообщения, принцип контроля целостности и достоверности данных, их шифрование и многие другие моменты (перечисленные в этой строке аспекты присутствуют не у всех протоколов).

Примеры протоколов

Самые «общеизвестные» протоколы, о которых вы наверняка слышали – это TCP, UDP, HTTP, HTTPS.

К «общепромышленным» протоколам относятся Modbus, CANopen, Profibus, PROFINET, EtherCAT, EtherNet/IP, семейство протоколов OPC UA и другие.

Существуют протоколы, специфичные для конкретных отраслей и областей:

- телемеханика и диспетчеризация в энергетике: МЭК 60870–5–104 (часто называемый «МЭК 104» или «104–й протокол»), МЭК 60870–5–103, МЭК 60870–5–101, DNP3, протоколы стандарта МЭК 61850;
- автоматизация зданий: BACnet, KNX, LonWorks;
- «умный дом» и домашняя автоматизация: MQTT, Thread, Matter, Z–Wave и другие;
- протоколы датчиков: HART, IO–Link, AS–i;
- протоколы тепло– и электросчётчиков: DLMS/COSEM (и его российский профиль СПОДЭС), МЭК 61107, специализированные протоколы производителей (например, протокол «Меркурий», поддерживаемый электросчётчиками Меркурий).

В рамках нашего курса мы будем рассматривать исключительно Modbus.

Взаимосвязь интерфейсов и протоколов

Давайте проговорим ещё раз – интерфейс описывает «физику» передачи данных (характеристики среды передачи и проходящих по ней сигналов), а протокол – «логику» (как закодировать в этих сигналах какую–либо информацию).

Соответственно, для передачи данных нужны и интерфейс, и протокол.

С одной стороны, протоколы обычно проектируются без привязки к определённому интерфейсу – это позволяет использовать один и тот же протокол для передачи данных по медному кабелю, оптоволокну, радиоэфиру и т. д.

Но при этом – протоколы и интерфейсы обычно создают «устойчивые связки».

Например, протокол PROFINET в подавляющем большинстве случаев будет использоваться совместно с интерфейсом Ethernet; сложно представить ситуации, когда он будет использоваться с RS–485 или Wi–Fi.

Стеки протоколов

Протоколы могут использоваться совместно, дополняя друг друга. В этом случае пакеты одного протокола помещаются (инкапсулируются) в пакеты другого по принципу матрёшки. Набор таких инкапсулированых протоколов называют **стеком**.

Одними из известнейших стеков протоколов является TCP/IP и UDP/IP; упомянутый выше протокол PROFINET (точнее, один из его профилей [разновидностей] – PROFINET CBA) реализован как раз поверх стека UDP/IP.

Модель OSI

Протоколы, работающие в рамках стека, выполняют разные задачи. Для их классификации и иерархии была разработана сетевая модель OSI (**O**pen **S**ystems **I**nterconnection). С течением времени в адрес этой модели можно услышать всё больше критики; с её помощью сложно выразить всё многообразие существующих протоколов и взаимосвязей между ними, но она достаточно неплохо описывает их базовое устройство.

Модель OSI определяет 7 уровней протоколов. Приведём скриншот из соответствующей статьи Википедии.

Обратите внимание, что самый нижний уровень – физический – соответствует тому, что мы называем «коммуникационным интерфейсом».

Модель					
Уровень (layer)		Тип данных (PDU)		Примеры	Оборудование
Host layers	7. Прикладной (application)	Данные	Доступ к сетевым службам	HTTP, FTP, POP3, SMTP, WebSocket	Хосты (клиенты сети), Межсетевой экран
	6. Представления (presentation)		Представление и шифрование данных	ASCII, EBCDIC, SSL, gzip	
	5. Сеансовый (session)		Управление сеансом связи	RPC, PAP, L2TP, gRPC	
	4. Транспортный (transport)	Сегменты (segment) / Датаграммы (datagram)	Прямая связь между конечными пунктами и надёжность	TCP, UDP, SCTP, Порты	
Media layers	3. Сетевой (network)	Пакеты (packet)	Определение маршрута и логическая адресация	IPv4, IPv6, IPsec, AppleTalk, ICMP	Маршрутизатор, Сетевой шлюз, Межсетевой экран
	2. Канальный (data link)	Биты (bit)/ Кадры (frame)	Физическая адресация	PPP, IEEE 802.22, Ethernet, DSL, ARP, сетевая карта.	Сетевой мост, Коммутатор, точка доступа
	1. Физический (physical)	Биты (bit)	Работа со средой передачи, сигналами и двоичными данными	USB, RJ («витая пара», коаксиальный, оптоволоконный), радиоканал	Концентратор, Повторитель (сетевое оборудование)

Спецификации и стандарты

Вы ведь ещё помните, что интерфейс – это «описание...», а протокол – «набор правил...» (в сущности – тоже описание)?

То есть по своей природе – и интерфейсы, и протоколы являются документами. Эти документы называются **спецификациями**.

Если спецификация прошла ряд проверок и других регламентированных процедур в соответствующей организации – то она становится **стандартом**.

Производители оборудования и ПО обеспечивают **реализацию** стандарта:

- для интерфейса реализацией являются коммуникационные микросхемы, кабели связи и т. д.;
- для протокола реализацией является программа (часто называемая драйвером).

Стандарты обеспечивают совместимость между оборудованием и ПО различных производителей, в которых реализованы одни и те же интерфейсы и протоколы; ответственность за соответствие стандарту лежит на производителях. В конце курса, в модуле 7 мы рассмотрим примеры реализации Modbus, имеющие отступления от спецификации, и обсудим, к каким последствиям это приводит.

А теперь пришло время вернуться к теме нашего курса – к Modbus.

2.3 Modbus в прошлом и настоящем

Давайте повторим определение, с которого мы начали предыдущий урок:

> *Modbus – это семейство протоколов обмена, используемых в системах промышленной автоматизации и некоторых других областях.*

Мы уже знаем, что такое «протокол» и «интерфейс».

Modbus – это «полевой» протокол (fieldbus), используемый для обмена данными между датчиками, контроллерами, модулями ввода–вывода и другими промышленными устройствами. Эти устройства обычно устанавливаются непосредственно рядом с оборудованием, обеспечивающим протекание технологического процесса («в поле») и используются для контроля и управления этим процессом.

Разновидности Modbus и используемые ими интерфейсы

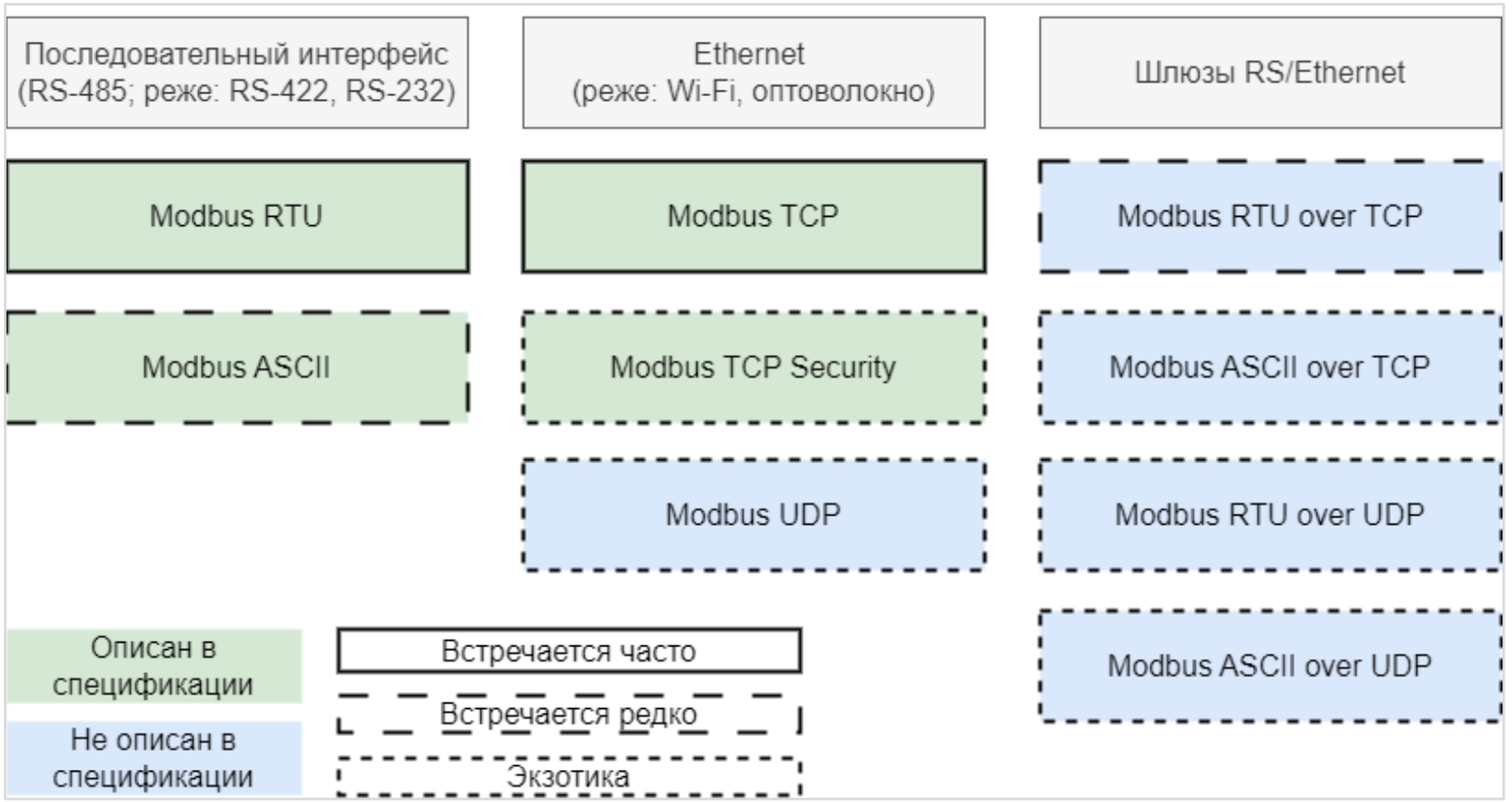
В рамках спецификации (точнее, набора спецификаций) Modbus описаны следующие протоколы:

- **Modbus RTU** и **Modbus ASCII** (совместно обозначаемые как **Modbus Serial**), реализуемые поверх последовательного интерфейса (в большинстве случаев – поверх интерфейса **RS–485**; в более редких случаях используются интерфейсы **RS–422** и **RS–232**);
- **Modbus TCP**, реализуемый поверх стека протоколов **TCP/IP**. В качестве интерфейса обычно используется **Ethernet**; реже – **Wi–Fi**, оптоволокно или какая–то другая технология;
- **Modbus TCP Security** – вариация протокола Modbus TCP с поддержкой шифрования канала связи.

На практике – изредка можно встретиться с вариациями Modbus, не описанными в спецификации:

- **Modbus UDP** – аналог Modbus TCP, реализованный поверх стека **UDP/IP**;
- **Modbus RTU over TCP (UDP)**, **Modbus ASCII over TCP (UDP)** – варианты использования протоколов Modbus RTU и Modbus ASCII поверх стеков протоколов TCP/IP (или UDP/IP). Связаны с интеграцией в рамках одной промышленной сети устройств с различными интерфейсами и использованием шлюзов COM/Ethernet, о которых мы поговорим в модуле 5.

В реальной жизни в 95+% случаев вы будете сталкиваться с протоколами Modbus RTU и Modbus TCP.



В рамках данного курса мы не будем рассматривать протоколы, имеющие те или иные связи с Modbus, но не упоминаемые в его официальных спецификациях:

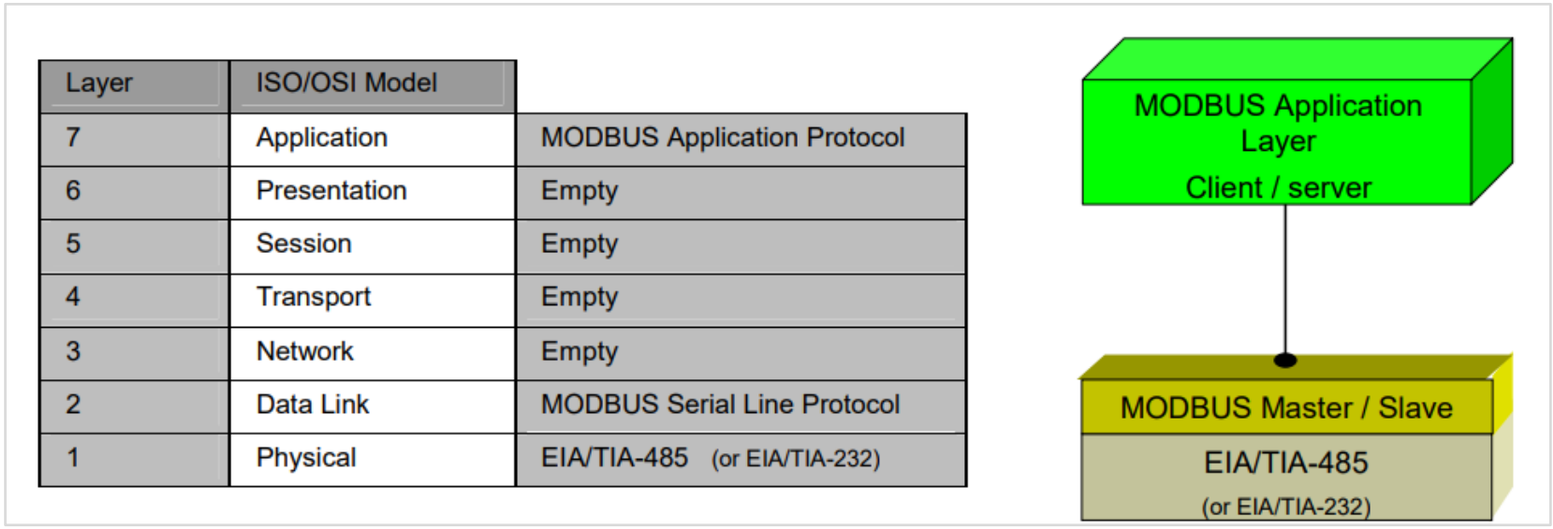
- Modbus+ (Modbus Plus) – протокол, разработанный компанией Schneider Electric и связанный с «оригинальным» Modbus, в основном, лишь названием;
- Enron Modbus, JBUS и др. – протоколы, являющиеся расширениями Modbus, разработанные различными компаниями–производителями;
- Modbus RTPS – вариация Modbus для систем реального времени, разрабатываемая в середине 2000–х. Не была завершена; наработки легли в основу протокола DDSI–RTPS.

Протокол Modbus и модель OSI

С точки зрения модели OSI, рассмотренной в конце прошлого урока:

- протоколы Modbus RTU и Modbus ASCII охватывают канальный и прикладной уровень;
- протокол Modbus TCP охватывает прикладной уровень.

«Modbus Application Protocol» и «Modbus Serial Line Protocol», упомянутые на рисунке ниже – это просто разные фрагменты, в сумме составляющие спецификацию протоколов Modbus RTU и Modbus ASCII.



Layer	ISO/OSI Model	
7	Application	MODBUS Application Protocol
6	Presentation	Empty
5	Session	Empty
4	Transport	Empty
3	Network	Empty
2	Data Link	MODBUS Serial Line Protocol
1	Physical	EIA/TIA-485 (or EIA/TIA-232)

Историческая справка

Протокол Modbus был разработан в 1979 году компанией Gould Electronics для обмена данными между контроллерами Modicon и их периферийными устройствами. В качестве интерфейса связи использовался RS–232 или другие схожие интерфейсы, существовавшие в те времена; до первых пробных внедрений RS–485 и Ethernet остаётся ещё несколько лет.

Modicon – это первый бренд программируемых логических контроллеров (ПЛК), созданный в конце 1960–х, а Modbus – это первый промышленный протокол обмена. Modbus означает «Modicon Bus» (шина данных контроллеров Modicon).

> *Подробнее об истории Modicon можно узнать из статьи Ванесса Ромеро Сеговия, Альфред Теорин. История управления. История ПЛК и РСУ.*

В 1997 права на бренд Modicon и протокол Modbus были проданы компании **Schneider Electric**.

В 2004 году Schneider Electric передала права на протокол Modbus некоммерческой организации **Modbus Organization**, которая сделала спецификации протокола общедоступными.

Вот как выглядели одни из первых устройств с поддержкой протокола Modbus:

Источник: https://all–andorra.com/modicon–584–hmi/

Modvue, a man-machine interface system from Gould, officially designated the CR-900 Programmable Touch Control Center, features a high-resolution, touch-sensitive screen that allows users to interact with the system by touching specific areas of the display.

Control center is designed as aid to plant management

Modvue from Gould, Inc., is a man-machine interface system with applications in the process and discrete-parts manufacturing industries. The disk-based, graphic, color-CRT system facilitates plant monitoring, reporting, alarming, computing, and supervisory control for applications using Gould Modicon programmable controllers.

Principal features of the Modvue CR-900 include a high-resolution, touch-sensitive screen; menu-driven, operator-prompting graphic commands; pixel-oriented graphics with a resolution of 480×311 addressable dots; a 19-inch color monitor with graphic processors to display up to 512 colors; and split-screen, multi-tasked support of up to four zones on a video screen in order to monitor and control independent aspects of operations asynchronous to the process.

The touch-screen feature is designed to reduce the complexity usually associated with keyboards. An operator can touch areas of the video screen and call up displays, acknowledge alarms, start and stop devices, or modify setpoints and alarm limits.

Modvue is priced in the $26,000 range, depending on peripheral equipment.

Reader Service Number 42

Источник: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1654057

Спецификации и стандарты Modbus

Спецификации Modbus доступны для загрузки на сайте Modbus Organization:

1. MODBUS over Serial Line. Specification and Implementation Guide V1.02 **(12.2006 / 44 стр.)**

Рассматривает вопросы реализации Modbus RTU и Modbus ASCII (требования к физическому уровню и др.).

2. MODBUS Messaging on TCP/IP Implementation Guide V1.0b **(10.2006 / 46 стр.)**

Рассматривает вопросы реализации Modbus TCP.

3. MODBUS APPLICATION PROTOCOL SPECIFICATION V1.1b3 **(06.2012 / 50 стр.).**

Рассматривает общие аспекты протокола Modbus, касающиеся прикладного уровня модели OSI и универсальные для Modbus RTU, Modbus ASCII и Modbus TCP – формат пакетов, модели данных, функции (операции над данными) и т. д.

*****

Далее в курсе мы часто будем ссылаться на эти документы – поэтому рекомендуем сохранить ссылки на них или скачать.

В ссылках мы будем использовать обозначение типа [3, п. 4.1] – соответственно, это будет означать «п. 4.1 в документе MODBUS APPLICATION PROTOCOL SPECIFICATION V1.1b3».

Также в разделе спецификаций доступна спецификация протокола **Modbus TCP Security** и некоторые другие документы (например, прошлая версия спецификации Modbus Serial, принятая в 1996 году и ориентированная на применение в ПЛК Modicon).

***

Спецификация протокола Modbus TCP принята как стандарт и входит в состав стандартов МЭК 61158 (Промышленные сети. Спецификации полевых шин), МЭК 61784–2 (Промышленные сети. Профили. Часть 2. Дополнительные профили полевых шин для сетей, работающих в реальном времени на базе ISO/IEC/IEEE 8802–3) и МЭК 62030 (Передача цифровых данных измерений и контроля. Полевая шина для промышленных систем управления).

Спецификация протоколов Modbus RTU и Modbus ASCII, насколько нам известно, не принята в качестве официального стандарта, но из–за широты своего распространения является де–факто стандартом промышленной отрасли.

Modbus в современном мире

Два слайда назад мы упомянули, что протокол Modbus был разработан в 1979 году – и это должно было вызвать у вас закономерный вопрос:

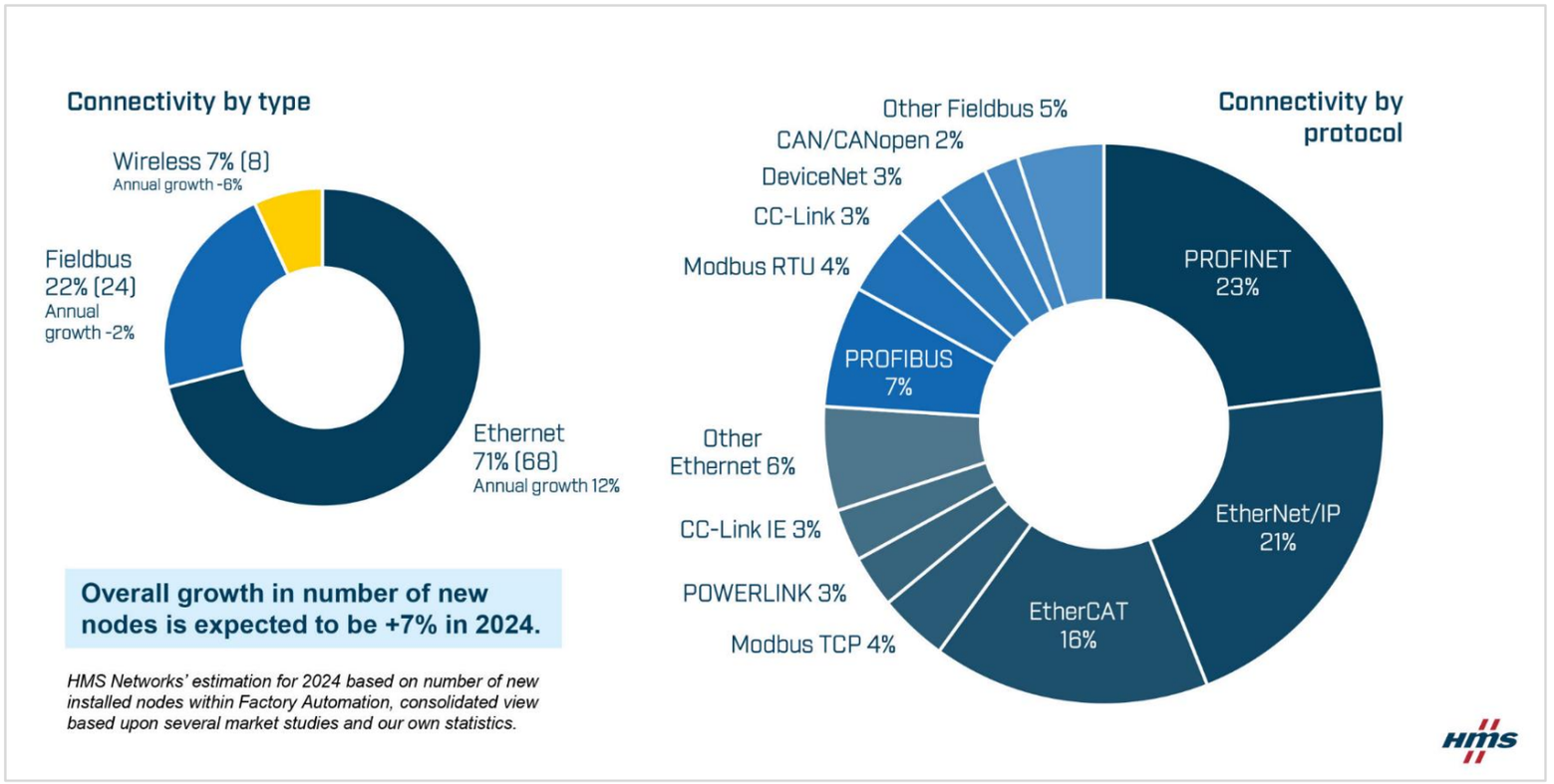
- какой же смысл говорить о нём сейчас, в 2025 (или более позднем) году?

Дело в том, что Modbus продолжает использоваться. Согласно исследованию, проведённому компанией HMS Networks в 2024 году, он занимает 8% рынка промышленных протоколов (4% приходится на Modbus RTU и 4% на Modbus TCP).

Это, безусловно, выглядит не так эпично, как «большая тройка» (PROFINET, EtherNet/IP и EtherCAT), суммарно занимающая 60% проценков рынка и наглядно демонстрирующая постепенный переход систем автоматизации к протоколам реального времени. Но для протокола, разработанного больше 45 лет назад – 8% совсем не мало.

Кроме того:

- распространённость Modbus достаточно медленно падает с течением времени. В отчёте HMS Networks за 2017 год на долю Modbus TCP приходилось всё те же 4% рынка, что и в 2024. С другой стороны – доля Modbus RTU в тот год составляла 6%;
- отчёты HSM Networks не привязаны к каким–то регионам. Как показывает наша практика – в России распространённость Modbus выше, чем в Европе или США.



Источник: https://www.hms–networks.com/news/news–details/17–06–2024–annual–analysis–reveals–steady–growth–in–industrial–network–market

Modbus в приборах ОВЕН

В настоящее время Modbus является основным протоколом приборов компании ОВЕН. Он поддерживается в:

- датчиках с интерфейсом RS–485;
- терморегуляторах ТРМ и других приборах ассортиментного направления КИП (счётчиках импульсов, тахометрах, архиваторах и т. д.);
- модулях ввода–вывода Mx110 и Mx210;
- преобразователях частоты ПЧВ;
- блоках питания и других устройствах направления Силовые устройства (УЗД1, ПБР10, драйверах шаговых двигателей);
- программируемых реле ПР;
- конфигурируемых контроллерах (КТР, СУНА, МГ1, Вьюга и т. д.);
- программируемых контроллерах ПЛК и СПК;
- панелях оператора.

Протокол Modbus поддерживается в использованном нами ПО Owen OPC Server (вспомните урок 2.1) и облачном сервисе OwenCloud.

На этом изображении приведены лишь **некоторые** из упомянутых выше приборов с поддержкой Modbus:

Причины популярности

Распространённости протокола Modbus способствует:

1. Открытость и отсутствие зависимости от какого–либо производителя

- спецификации Modbus находятся в открытом доступе – для их получения не требуется вступать в какую–либо организацию и оплачивать их приобретение;
- любой производитель оборудования может разработать прибор с поддержкой Modbus и продавать его без каких–либо лицензионных отчислений;
- спецификация является стабильной (последние значимые изменения были внесены в середине 2000–х годов);
- за протоколом Modbus не стоит какой–то «разработчик–владелец» (например, для EtherCAT таковым является компания Beckhoff, для Profibus и Profinet – компания Siemens и т. д.), что исключает возможность недобросовестной конкуренции. В этом аспекте независимость Modbus является исключением (среди других подобных исключений можно отметить разве что CANopen и относительно новую по меркам отрасли технологию OPC UA);
- для аппаратной поддержки Modbus достаточно, чтобы устройство имело порт RS–485 (или другой последовательный интерфейс) или Ethernet (для Modbus TCP). Не требуется использования специфических коммуникационных чипов – как, например, в случае протоколов Profibus, Profinet, EtherCAT и т. д.

2. Компактная спецификация, обеспечивающая простоту реализации

- три основных документа суммарно занимают около 150 страниц и написаны простым и понятным языком;
- квалифицированному инженеру–программисту обычно хватает нескольких дней, чтобы «поднять» Modbus на новом разрабатываемом устройстве – даже в случае отсутствия готовой библиотеки;
- полнофункциональная реализация обычно занимает не больше месяца. Для сравнения – реализация других современных промышленных протоколов (OPC UA, BACnet, DNP3 и др.) может занять вплоть до нескольких лет. Зачастую производители оборудования предпочитают приобретать готовую реализацию подобных протоколов, чем разрабатывать её самостоятельно;
- с другой стороны – компактность спецификации перекладывает на разработчика принятие ряда решений, которые формируют в приборах разных компаний (иногда – даже в пределах одной компании) отличия, затрудняющие их совместное использование.

3. Возраст и «проверенность временем»

- за 45 лет существования протокола выпущено множеств устройств с его поддержкой, которые внедрены во множество систем автоматизации;
- инертность и ориентированность на поддержку legacy – это дух АСУТП.

Modbus получил распространение как протокол интеграции, использующийся для настройки обмена между устройствами и ПО различных производителей. Обычно у крупных производителей есть «родной» протокол:

- Profibus и Profinet для Siemens;
- EtherCAT для Beckhoff;
- CC–Link для Mitsubishi Electric;
- FINS для Omron;
- и т. д.

Но при этом совершенно необязательно, что ПЛК Mitsubishi будет поддерживать протокол FINS, а ПЛК Omron – протокол CC–Link. Зато они оба поддерживают Modbus – который за счёт простоты реализации и десятилетий внедрения стал «универсальным» языком автоматизации.

> *В начале 2000–х годов компания ОВЕН разработала собственный протокол обмена, который так и назывался – «ОВЕН». Он не получил широкого распространения и в настоящее время не поддерживается в новых приборах; основным протоколом приборов компании стал Modbus – как раз за счёт того, что он поддерживается практически всеми производителями, что облегчает совместное использование их устройств.*

Но в тоже время...

Тот факт, что Modbus был разработан давно и ориентировался на решение конкретных задач своей эпохи, накладывает определённые ограчения на сферу его применения:

1. Modbus – это не протокол реального времени

Когда нужно опросить «100 приборов за одну секунду» – Modbus просто не подходит. Вам нужны устройства, поддерживающие протокол реального времени – EtherCAT, Profinet и т. д.

Но если вы сами разрабатываете прибор с поддержкой Modbus – то постарайтесь обеспечить, чтобы в вашей реализации обмен работал как можно быстрее. Для этого нужно обеспечить поддержку **групповых запросов**, о которых мы ещё часто будем упоминать в дальнейших уроках.

2. Modbus – это не про «информационную безопасность»

Спецификация Modbus Serial не описывает никаких средств для обеспечения информационной безопасности – во времена разработки протокола такой проблемы просто не существовало.

Не описывает их и спецификация Modbus TCP.

В 2018 году была опубликована спецификация протокола Modbus TCP Security, представляющего собой вариацию Modbus TCP с использованием известного криптографического протокола TLS. В данный момент этот протокол не получил широкого распространения.

3. Компактность спецификации приводит к разным нюансам конкретных реализаций

В частности – в спецификации практически не описаны типы данных, поэтому способ представления близких по смыслу параметров в разных приборах может существенно отличаться. Одно из характерных проявлений этого факта – различный порядок регистров, используемых разными приборами при передаче данных. Помните настройку **Младшим регистром вперёд**, которую нам потребовалось установить в Owen OPC Server при опросе датчика ПВТ110–RS в уроке 2.1? Она как раз связана с этим.

4. Отсутствие универсального формата описания параметров устройства

Наиболее распространённые промышленные протоколы к данному моменту поддерживают универсальный формат описания параметров. В этом случае вместе с прибором распространяется файл (обычно XML–подобного формата), который может быть импортирован в любое ПО с поддержкой данного протокола с целью предоставить пользователю весь список его параметров.

Примеры таких форматов:

- протокол EtherCAT – формат ESI (EtherCAT SubDevice Information);
- протокол Profibus – формат GSD (General Station Description);
- протокол Profinet – формат GSDML (General Station Description Markup Language);

- протоколы CANopen и EtherNet/IP – формат EDS (Electronic Description).

Эти файлы в определённом смысле похожи на шаблон, который мы добавляли в Owen OPC Server в уроке 2.1 – только вот в случае OPC–сервера формат шаблона уникален и поддерживается лишь им самим.

Поэтому при использовании Modbus список опрашиваемых параметров обычно приходится вводить вручную (*как это потом сделали и мы*), ориентируясь на предоставленную производителем документацию (*которая далеко не всегда представляет из себя образец наглядности*). Это является одним из существенных недостатков спецификации Modbus.

2.4 Тестовые задания

Ответы приведены в п. 9.

1. Как расшифровывается название протокола Modbus
    - Modicon bus
    - Module bus
    - Modern bus
    - Modified bus

2. Укажите исторический период, в котором был создан протокол Modbus.
    - 1990-е - 2000-е
    - 2010-е - 2020-е
    - 1970-е - 1980-е

3. Какие протоколы определены в спецификации Modbus?
    - Modbus UDP
    - Modbus TCP
    - Modbus RTU over TCP
    - Modbus RTPS
    - Modbus ASCII
    - Modbus RTU

4. Какой из документов описывает особенности физического уровня протоколов Modbus RTU и Modbus ASCII?
    - MODBUS Messaging on TCP/IP Implementation Guide
    - MODBUS APPLICATION PROTOCOL SPECIFICATION
    - MODBUS over Serial Line. Specification and Implementation Guide

2.5 Основы Modbus, часть 1

В следующих шагах мы начнём обсуждать спецификацию протокола Modbus.

Сначала мы сфокусируемся на Modbus Serial (на вариантах Modbus, реализованных поверх последовательных интерфейсов типа RS–485 – протоколах Modbus RTU и Modbus ASCII), но существенная часть приведённой информации является универсальной и будет справедлива и для протокола Modbus TCP.

> *RTU означает Remote Terminal Unit (современным эквивалентом этого понятия является «модуль ввода–вывода»).*

Архитектура протокола Modbus

Протокол Modbus основан на архитектуре **master/slave**.

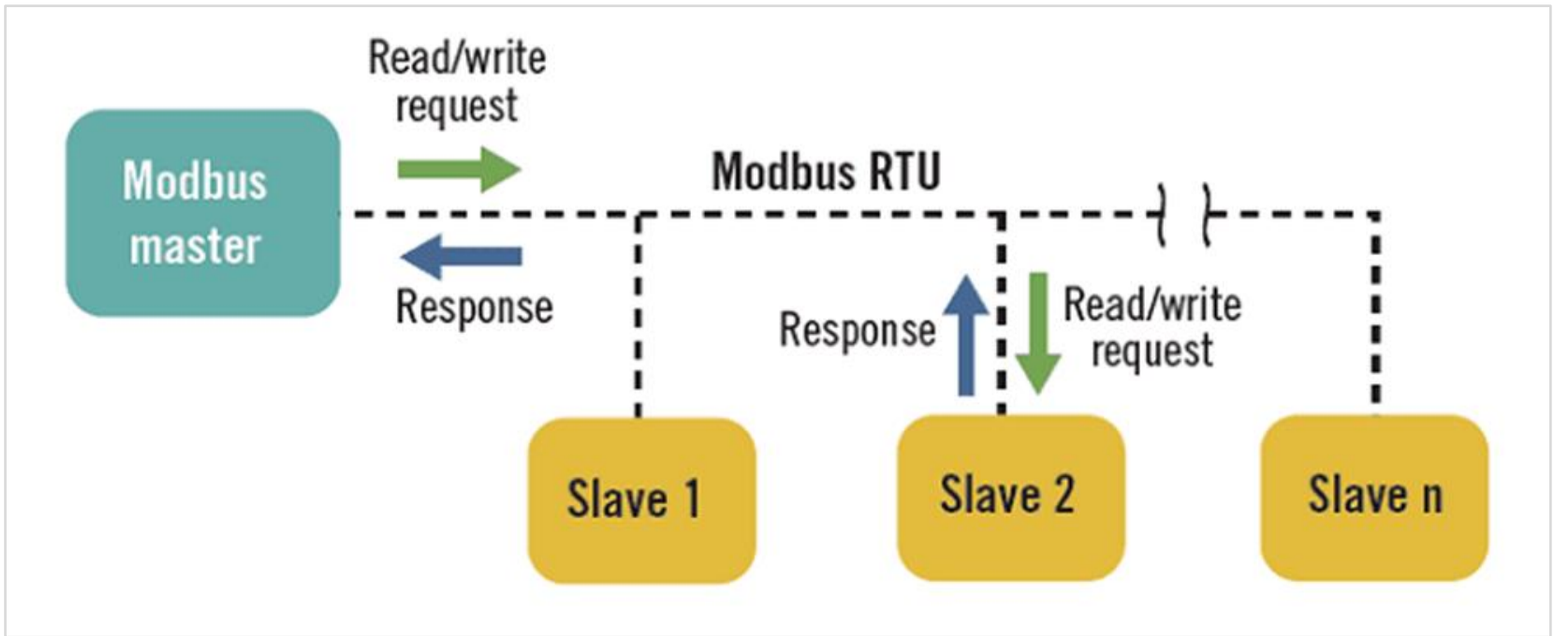
> *Применительно к Modbus TCP эту архитектуру иногда называют **client/server** (где **client** – синоним термина **master**, а **server** – термина **slave**).*
>
> *Летом 2020 года Modbus Organization выпустила пресс–релиз, в котором объявила, что отказывается от использования терминов master и slave в своих спецификациях (после чего документы были отредактированы, хотя не очень тщательно – кое–где эти термины остались) в связи с тем, что они относятся к «неподходящей лексике», и могут задеть чувства определённых категорий людей. В данном курсе используются термины master и slave – применительно и к Modbus TCP, и к Modbus Serial. Это связано с тем, что они и в настоящий момент активно используются на практике (в графическом интерфейсе различного ПО, документации и т. д.).*

В рамках этой архитектуры существуют две роли (режима работы):

- master – инициатор опроса. Он отправляет запрос на чтение или запись данных в slave–устройство;
- slave – обработчик запросов. Он ожидает запроса от мастера и, при его получении, выполняет заданную операцию и отправляет ответ.

В уроке 2.1 утилита Owen OPC Server использовалась в режиме master, а датчик ПВТ110–RS – в режиме slave.

Принцип работы устройств с протоколом Modbus Serial можно продемонстрировать следующей схемой:



Источник: https://www.controlglobal.com/network/industrial–networks/article/11304326/introduction–to–modbus

- master–устройство отправляет запрос (request) одному из slave–устройств;
- slave–устройство обрабатывает этот запрос и отправляет ответ (response);
- master–устройство получает этот ответ и отправляет следующий запрос (либо этому же slave–устройству, либо другому);
- slave–устройство обрабатывает этот запрос и отправляет ответ;
- ...
- и так «по кругу».

Эта схема даёт хорошее базовое понимание, как происходит обмен между устройствами в рамках рассматриваемой архитектуры. Можно уже сейчас сформулировать вопросы типа «что будет в случае разрыва линии связи?», «что будет, если slave–устройство не ответит на запрос по причине выхода из строя или отсутствия питания?» и т. п. – но давайте отложим их до урока 2.12.

Для Modbus Serial существует **важное ограничение** [1, п. 1.2]:

- в пределах одной конкретной последовательной линии связи (её часто называют «шиной») может функционировать только одно устройство, выполняющее роль Master.

> *Это ограничение связано с принципом работы интерфейса RS–485, который мы рассмотрим в уроке 2.14. В некоторых других протоколах, использующих последовательные интерфейсы (например, Profibus и CAN–протоколах), это ограничение отсутствует за счёт специальных механизмов; спецификация Modbus Serial не описывает подобные механизмы.*

Для Modbus TCP такого ограничения нет; в сети может одновременно присутствовать несколько устройств, функционирующих в режиме Modbus TCP Master. Более того, в рамках Modbus TCP одно устройство может одновременно функционировать и в режиме Master, и в режиме Slave (если такая работа поддержана самим устройством). Мы подробнее поговорим об этом в уроке 2.20.

Типичные примеры устройств, которые могут работать только в режиме slave:

- датчики;
- терморегуляторы;
- модули ввода–вывода;
- преобразователи частоты;
- и т. д.

Но есть и исключения – существуют специфические представители всех упомянутых категорий, которые поддерживают работу в режиме Master.

Программируемые устройства – контроллеры, панели оператора, ПО верхнего уровня и т. д. – обычно могут работать и в режиме Master, и в режиме Slave.

Исключения есть и среди них – например, использованный нами в уроке 2.1 Owen OPC Server может работать только в режиме Modbus Master.

> *Тут у вас может возникнуть вопрос: если master – это то же самое, что и client, то почему программа называется Owen OPC Server, а не Owen OPC Client? Дело в том, OPC–сервер является «сервером» именно с точки зрения технологии OPC – он обрабатывает запросы OPC–клиента, которым обычно является SCADA–система. С точки зрения своих «полевых» протоколов (Modbus и т. д.) OPC Server обычно работает в режиме мастера; некоторые OPC–серверы могут работать и в режиме Slave.*

Основные понятия

Под **запросом** мы будем подразумевать команду, которое master–устройство отправляет slave–устройству с заданным **адресом**.

Каждый тип команды характеризуется уникальным идентификатором, называемым в спецификации Modbus **кодом функции**.

Функция определяет операцию (чтение или запись) и **область памяти** slave–устройства, над которой она выполняется.

Область памяти определяет тип данных (**биты** или **регистры**) и **тип доступа** к ним (только чтение или чтение/запись).

В следующих шагах мы подробнее поговорим о всех терминах, выделенных жирным шрифтом.

Начнём с адресов slave–устройств.

Адресация slave–устройств в Modbus Serial

К линии связи может быть подключено несколько slave–устройств.

Чтобы мастер мог «отличить» их друг от друга – каждому slave–устройству должен быть назначен уникальный адрес. Master–устройство не имеет адреса (потому что ему никто не отправляет запросов).

Вспомните – в уроке 2.1 датчику ПВТ110–RS был назначен адрес **16**, и именно его мы указывали в Owen OPC Server.

В спецификации Modbus Serial указано [1, п. 2.1], что адрес устройства (Slave ID) является целым числом в диапазоне 1...247 .

Адрес **0** зарезервирован под широковещательную рассылку, которую мы обсудим в уроке 2.12. Согласно спецификации этот адрес нельзя назначить slave–устройству.

Адреса **248...255** зарезервированы на потенциальные будущие расширения протокола. Некоторые устройства позволяют использовать эти адреса.

Адрес устройства не может иметь значение, превышающее **255**, по техническим причинам – в составе пакета Modbus под его хранение выделен один байт.

***

Адресацию устройств в Modbus TCP мы обсудим в уроке 2.20.

Типы данных

Спецификация Modbus определяет только два типа данных: биты и регистры [3, п. 4.2, 4.3].

Что такое бит – вы, вероятно, знаете; если нет – то, пожалуйста, воспользуйтесь статьёй на Википедии или аналогичным учебным материалом.

Регистр – это блок данных, размер которого составляет 16 бит (2 байта).

Больше спецификация Modbus не определяет никаких типов данных.

С исторической точки зрения (*вспомним – Modbus появился в 1979 году*) это вполне можно понять – двух приведённых типов хватает, чтобы с их помощью описать сигналы входов и выходов типичного контроллера:

- биты соответствуют дискретным входам и выходам;
- регистры соответствуют аналоговым входам и выходам (с помощью регистра достаточно просто можно представить целочисленное знаковое или беззнаковое значение – как, например, с помощью типа Int16 или Uint16 в языках программирования; регистров достаточно для работы с сигналами 16–битных АЦП и ЦАП).

То есть с точки зрения спецификации Modbus не существует типов Float, String, Int32/Int64/Uint32/Uint64 и т. д., к которым привыкли современные разработчики ПО; существуют только наборы регистров. Прикладное ПО, исполняемое на master–устройстве и на slave–устройстве, должно «знать», что именно «вот эти два регистра» представляют собой значение с плавающей точкой в формате Float/IEEE–754.

> *– Почему именно два регистра?*
>
> *– Потому что значение типа Float занимает 32 бита (см. ссылку выше), а регистр, как упоминалось выше, представляет собой блок данных размером в 16 бит. Соответственно, чтобы выделить 32 бита для хранения значения – нужно использовать 2 регистра.*

Отсутствие полноценной типизации – одно из самых трагичных упущений спецификации Modbus.

Предположим, вы каким–то образом узнали, что master–устройство отправило в slave–устройство такой набор данных (для их записи использована шестнадцатеричная система счисления/HEX):

> *41 42 43 44*

(это не весь запрос со всей его «обвязкой», а лишь сами передаваемые в его составе данные)

Мы знаем про них лишь то, что это 4 байта (т.е. 2 регистра) – и больше ничего.

Какому «настоящему» значению они соответствуют?

Это может быть:

- значение с плавающей точкой (Float) 12.141422;
- ASCII–строка (String) ABCD;
- целочисленное беззнаковое значение (Uint32) 1094861636
- два отдельных целочисленных беззнаковых значения (2 значения типа Uint16) 16706 и 17220;
- или что–то ещё – вариантов масса.

Таким образом, при настройке обмена по Modbus нужно изучить документацию на slave–устройство, чтобы выяснить, в каком виде оно предоставляет свои данные.

Quick links

- IEEE754
- FileHasher
- World sat image

Latest article

- Zone-based firewalling on Mikrotik routers

Contact

IEEE-754 Floating Point Converter

Translations: de

This page allows you to convert between the decimal representation of a number (like "1.02") and the binary format used by all modern CPUs (a.k.a. "IEEE 754 floating point").

IEEE 754 Converter, 2024-02

	Sign	Exponent	Mantissa
Value:	+1	2^3	1 + 0.5176777839660645
Encoded as:	0	130	4342596
Binary:	☐	☑☐☐☐☐☐☑☐	☑☐☐☐☐☑☐☐☑☐☐☐☐☑☑☐☑☐☐☐☑☐☐

Decimal Representation	12.141422
Value actually stored in float:	12.141422271728515625
Error due to conversion:	0.000000271728515625
Binary Representation	01000001010000100100001101000100
Hexadecimal Representation	41424344

1 -1

Использованный конвертер: https://www.h-schmidt.net/FloatConverter/IEEE754.html

ASCII to Hex

...and other free text conversion tools

Text (ASCII / ANSI)

ABCD

Convert Highlight Text

Binary

01000001 01000010 01000011 01000100

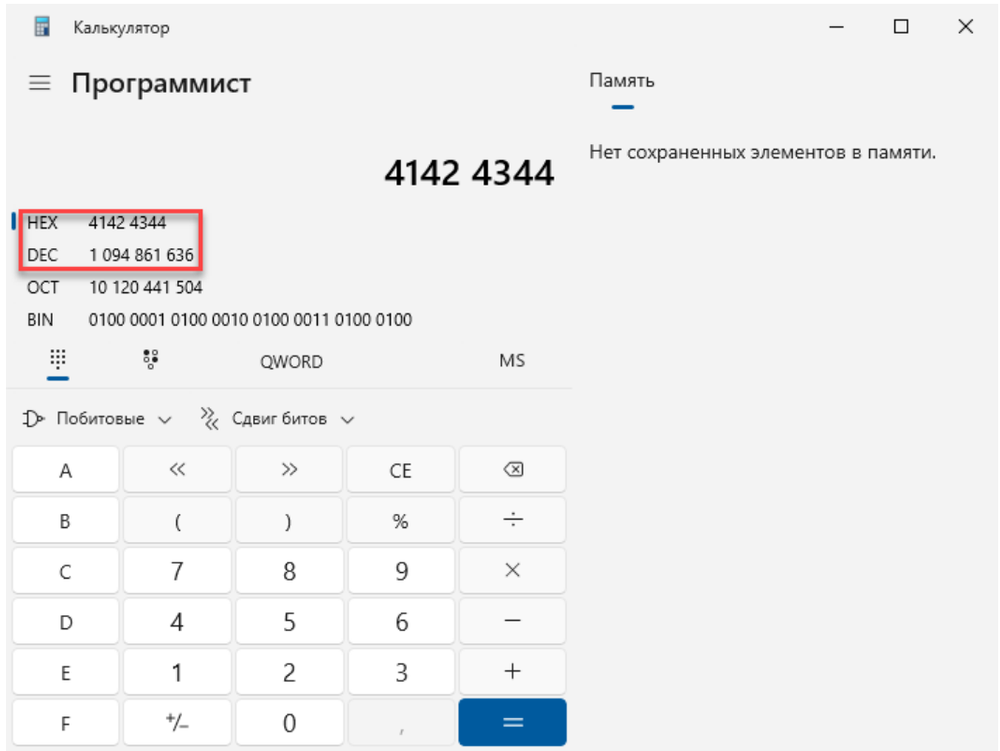
Convert Highlight Text

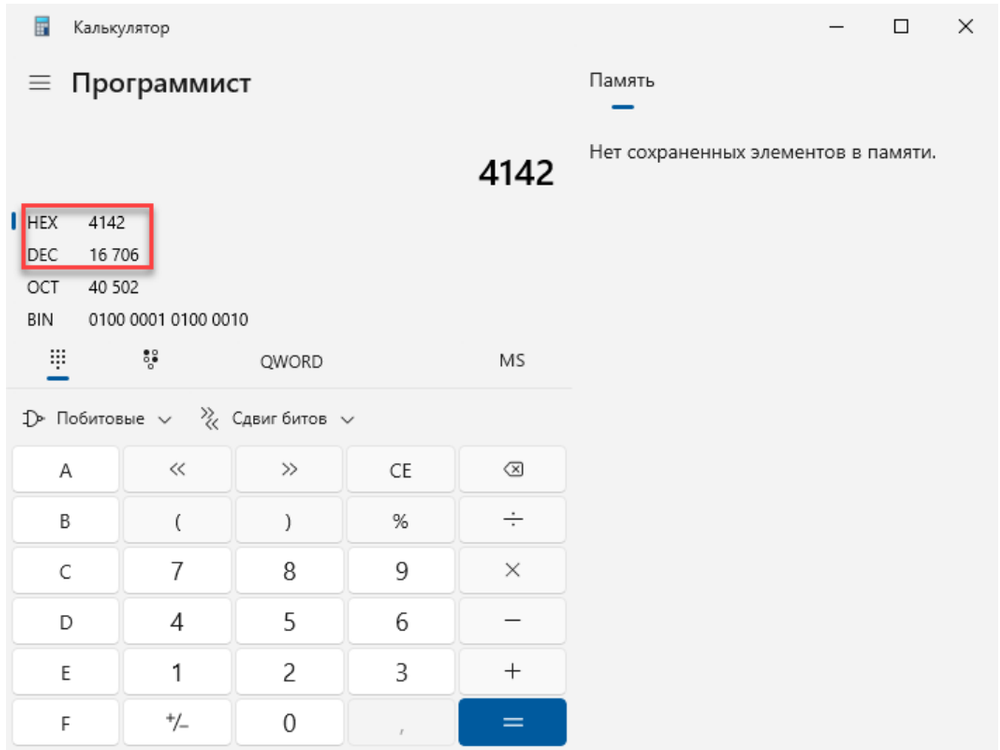
Hexadecimal

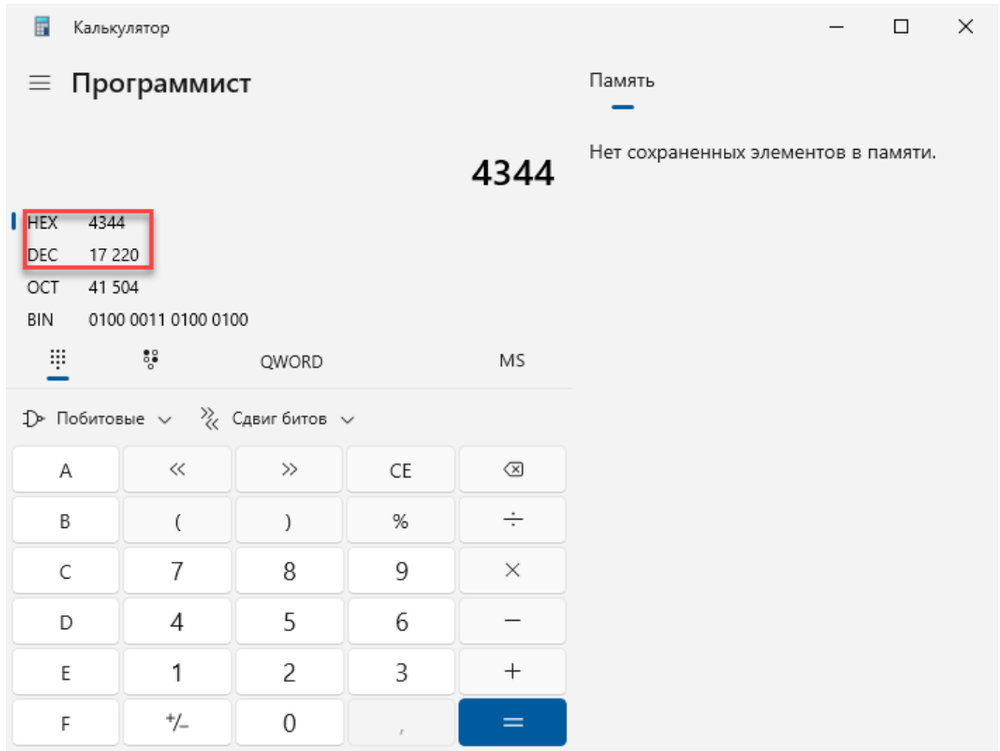
41 42 43 44

Convert Highlight Text

Использованный конвертер: https://www.asciitohex.com/

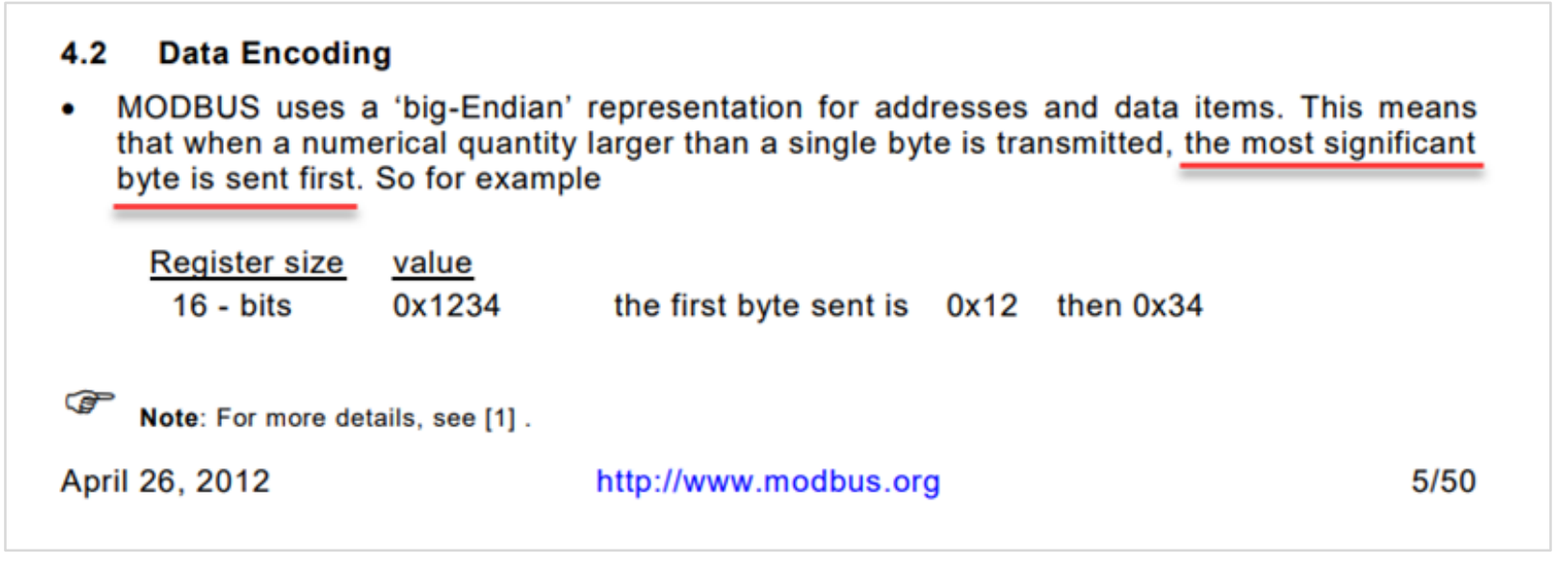

Калькулятор
Программист
4142 4344
HEX 4142 4344
DEC 1 094 861 636
OCT 10 120 441 504
BIN 0100 0001 0100 0010 0100 0011 0100 0100
QWORD
MS
Побитовые
Сдвиг битов
A
B
C
D
E
F
CE
%
7
8
9
4
5
6
1
2
3
0
Память
Нет сохраненных элементов в памяти.


Калькулятор
Программист
4142
HEX 4142
DEC 16 706
OCT 40 502
BIN 0100 0001 0100 0010
QWORD
MS
Побитовые
Сдвиг битов
A
B
C
D
E
F
CE
%
7
8
9
4
5
6
1
2
3
0
Память
Нет сохраненных элементов в памяти.


Калькулятор
Программист
4344
HEX 4344
DEC 17 220
OCT 41 504
BIN 0100 0011 0100 0100
QWORD
MS
Побитовые
Сдвиг битов
A
B
C
D
E
F
CE
%
7
8
9
4
5
6
1
2
3
0
Память
Нет сохраненных элементов в памяти.

Порядок байт и регистров (часть 1)

Спецификация Modbus указывает, что регистры должны передаваться старшим байтом вперёд [3, п. 4.2].



4.2 Data Encoding

- MODBUS uses a 'big-Endian' representation for addresses and data items. This means that when a numerical quantity larger than a single byte is transmitted, the most significant byte is sent first. So for example

Register size	value			
16 - bits	0x1234	the first byte sent is	0x12	then 0x34

Note: For more details, see [1] .

April 26, 2012 http://www.modbus.org 5/50

Но порядок регистров при передаче значений, которые занимают больше одного регистра, спецификация не определяет – поэтому он отличается у различных устройств.

В прошлом шаге мы упомянули, что набор байт

> *41 42 43 44*

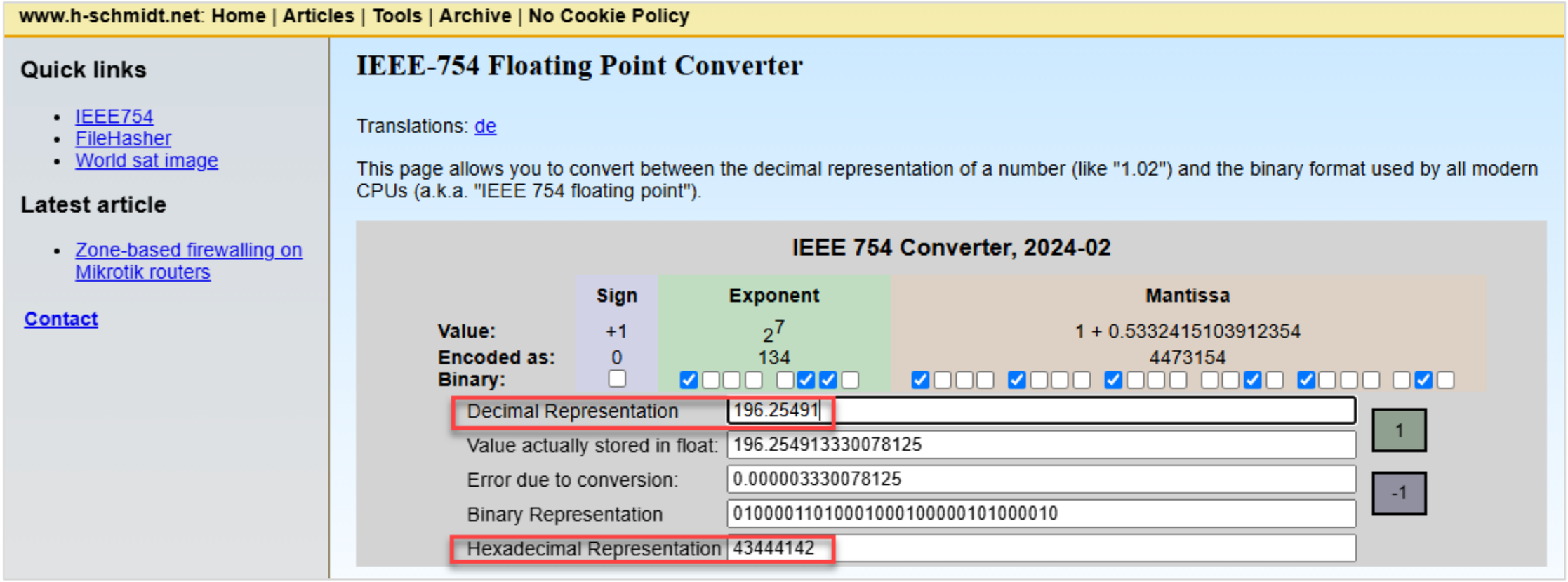
может соответствовать значению с плавающей точкой (Float) 12.141422.

Но с другой стороны – возможно и то, что в памяти устройства это значение хранится в таком виде:

> *43 44 41 42*

а при отправке по интерфейсу происходит «перестановка» регистров.

Тогда, соответственно, при его получении мы должны восстановить «правильный» порядок регистров – и окажется, что этот набор байт соответствует значению с плавающей точкой (Float) 196.25491.



> *Если вы имели опыт разработки сетевых приложений – то, вероятно, сразу вспомнили про network byte order / host byte order и связанные с этим преобразования. Действительно, это подходящая аналогия – только в Modbus речь идет о порядке регистров (16–битных блоках данных) и «network order» не специфицирован.*

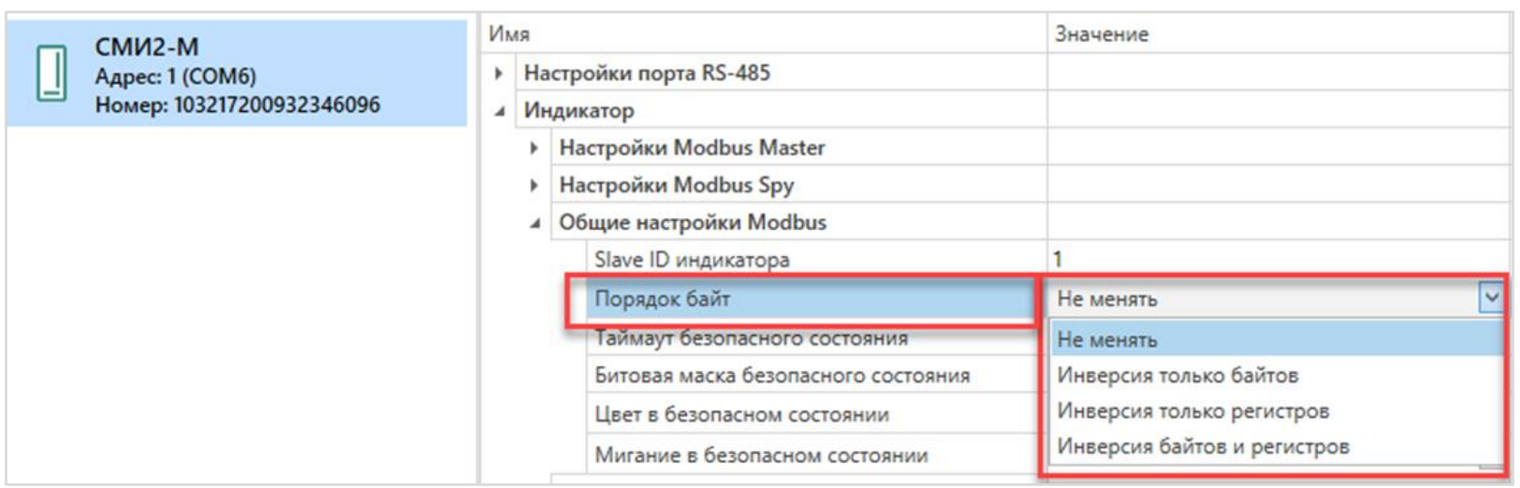
Обычно master–устройства позволяют настроить порядок регистров, который будет применяться при чтении или записи данных – вспомните параметр **Младшим регистром вперёд**, который мы установили в Owen OPC Server при считывании температуры с датчика ПВТ110–RS в уроке 2.1.

Ещё тогда мы отметили, что порядок регистров ПВТ110–RS не отражён в его документации. И действительно, такие вещи документируются крайне редко, поскольку касаются специфики аппаратной и программной платформы прибора.

Порядок байт и регистров (часть 2)

Разумно спроектированные slave–устройства тоже позволяют изменить порядок регистров – это позволяет «подстроиться» под те master–устройства, которые не позволяют настроить порядок или которые нет возможности перенастроить (например, из–за отсутствия исходников ПО).

Вот скриншот конфигурации Modbus–индикатора СМИ2–М:



Настройка **Порядок байт** позволяет задать порядок байт в отображаемом на индикаторе значении:

- **Не менять** – полученные по Modbus данные отображаются «как есть», без преобразований;
- **Инверсия только байтов** – в полученных по Modbus регистрах будут попарно переставлены байты;
- **Инверсия только регистров** – в полученных по Modbus данных будут попарно переставлены регистры (настройка применятся только для тех типов данных, которые занимают больше одного регистра – например, Float);
- **Инверсия байтов и регистров** – в полученных по Modbus данных будут попарно переставлены и байты, и регистры (перестановка регистров применяется только для тех типов данных, которые занимают больше одного регистра – например, Float).

Данный прибор предоставляет возможность выбора не только порядка регистров, но и порядка байт, хотя, как упоминалось выше, он определён спецификацией Modbus. Это позволяет настроить обмен с приборами, которые не соблюдают требование спецификации.

Важно отметить, что параметр **Порядок байт** должен влиять только на передаваемые в пакете Modbus данные, но не затрагивать служебные поля (например, адрес начального регистра и контрольную сумму), порядок байт которых определён в спецификации Modbus.

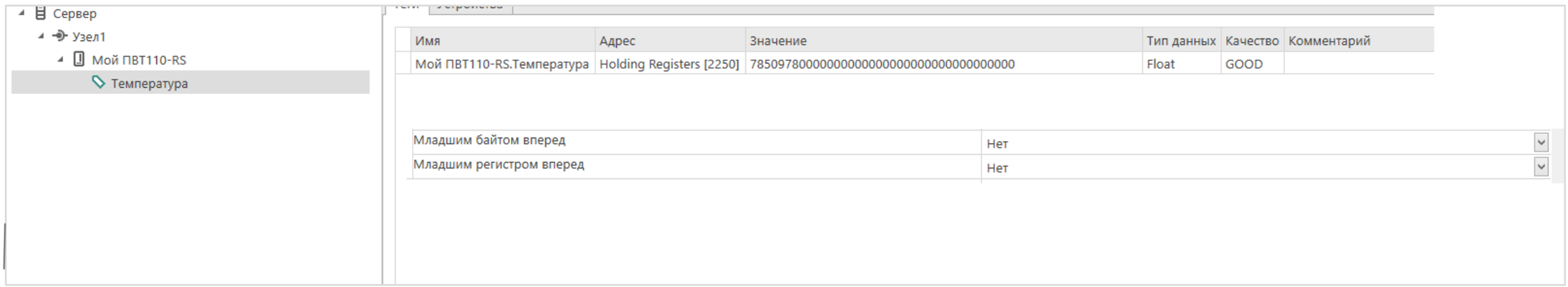
В рассмотренном нами в уроке 2.1 датчике ПВТ110–RS тоже имеется подобная настройка – **Последовательность байт в двухбайтовых данных** (возможные значения: **старший байт первый** и **младший байт первый**). Настроить порядок регистров в датчике не получится, хотя как раз это было бы более полезно.

Как определить, что вам требуется менять порядок байт/регистров? Очень просто – если опрос происходит, но вы получаете значение, которое совсем не похоже на то, которое вы ожидали, и при этом вы проверили, что все остальные настройки обмена (адрес регистра, количество регистров, тип данных и т. д.) соответствуют документации – то, вероятно, пришло время экспериментировать с порядком.

Вернёмся к ПВТ110–RS.

Предположим, вы не знали, что в параметре **Младший регистр вперёд** нужно установить значение **Да**.

Тогда при опросе вы бы увидели значение 0, которое, возможно, иногда бы менялось на какое–то огромное число:



Получение «очень маленьких» и или «очень больших» чисел – характерный признак некорректной настройки порядка байт/регистров.

В этом случае вам следовало бы изменить в OPC–сервере значения настроек **Младшим байтом вперёд** и **Младшим регистром вперёд**. Каждая из этих настроек имеет два возможных значения: **Да** или **Нет**. Соответственно, в совокупности имеется всего 4 возможных варианта:

- Младшим байтом вперед = Нет, Младшим регистром вперед = Нет;
- Младшим байтом вперед = Да, Младшим регистром вперед = Нет;
- Младшим байтом вперед = Нет, Младшим регистром вперед = Да;
- Младшим байтом вперед = Да, Младшим регистром вперед = Да.

При тестировании третьего варианта (Младшим байтом вперед = Нет, Младшим регистром вперед = Да) вы бы увидели корректное (ожидаемое вами) значение температуры – и поняли, что подобрали правильный порядок байт/регистров.

Таким образом – определение порядка байт/регистров для каждого устройства обычно происходит «методом перебора».

Функции Modbus

Функция определяет операцию (чтение или запись) и область памяти slave–устройства, над которой она выполняется.

Спецификация Modbus [3, п. 6] определяет 20 функций и позволяет разработчикам приборов реализовать собственные функции [п. 5 там же].

Важно отметить – конкретный прибор не обязан поддерживать все 20 функций; большинство приборов поддерживают всего несколько из них. Список поддерживаемых функций должен быть приведён в документации на устройство.

В таблице приведён полный список функций Modbus, разбитых по категориям:

Категория	Функции Modbus
Работа с регистрами (типовые функции)	0x03 Read Holding Registers 0x04 Read Input Registers 0x06 Write Single Register 0x10 Write Multiple Registers
Работа с битами	0x01 Read Coils 0x02 Read Discrete Inputs 0x05 Write Single Coil 0x0F Write Multiple Coils
Работа с регистрами (специфичные функции)	0x16 Mask Write Register 0x17 Read/Write Multiple Registers 0x18 Read FIFO queue
Работа с файлами	0x14 Read File Record 0x15 Write File Record
Диагностика (для RS-485/RS-232)	0x07 Read Exception Status 0x08 Diagnostic 0x0B Get Com Event Counter 0x0C Get Com Event Log 0x11 Report Server ID
Остальное	0x2B Encapsulated Interface Transport

На практике ситуация выглядит следующим образом:

- **в основном** используются типовые функции работы с регистрами (0x03, 0x04, 0x06, 0x10);
- функции работы с битами (0x01, 0x02, 0x05, 0x0F) используются существенно реже – в основном, при работе со старым или специфически спроектированным оборудованием. В современных приборах вместо них обычно используется принцип хранения бит в регистрах с помощью битовой маски – и, соответственно, для работы с этими масками можно применять «регистровые» функции. О битовых масках мы поговорим подробнее в уроке 2.6;
- все остальные функции используются в специфических ситуациях – возможно, вы никогда с ними не столкнетесь. В дальнейших уроках мы обсудим некоторые из них.

В уроке 2.8 мы подробнее обсудим типовые функции работы с регистрами и битами. Пока что только отметим, что каждая из этих категорий включает две функции чтения (название которых начинается со слова **Read**) и две функции записи (название которых начинается со слова **Write**).

Наличие двух функций чтения связано с областями памяти Modbus – мы обсудим их в следующем шаге.

Области памяти

Спецификация Modbus [3, п. 4.3] определяет для slave–устройств 4 области памяти, которые отличаются типом данных и типом доступа к ним. При этом конкретное устройство может иметь лишь **некоторые** области памяти из перечисленных:

ID	Обозначения	Применяемые функции	Описание
0x	Coils Дискретные выходы Ячейки, катушки, обмотки	0x01, 0x05, 0x0F	Биты, доступные для чтения и записи
1x	Discrete Inputs (DI) Дискретные входы	0x02	Биты, доступные только для чтения
3x	Input Registers (IR) Input-регистры Регистры ввода Аналоговые входы	0x04	Регистры, доступные только для чтения
4x	Holding Registers (HR) Holding-регистры Регистры хранения Аналоговые выходы	0x03, 0x06, 0x10, 0x16, 0x17	Регистры, доступные для чтения и записи

> *В OPC–серверах иногда в качестве синонима для «области памяти Modbus» используется термин «**регион**».*

Эти области хорошо накладываются на «физические» входы и выходы устройств:

- **Coils** – дискретные выходы. Их значения можно и прочитать, и записать;
- **Discrete Inputs** – дискретные входы. Их значения можно прочитать; записать их нельзя, потому что значение на дискретном входе определяется исключительно наличием или отсутствием на нём электрического сигнала;
- **Input Registers** – аналоговые входы. Опять же – их значения можно только прочитать;
- **Holding Registers** – аналоговые выходы. Их значения можно и прочитать, и записать.

Эта концепция была уместной во времена появления протокола, но у современных приборов по Modbus может быть доступно много параметров, которые нельзя соотнести с их «физическими» входами и выходами: версия прошивки, системное время, пароль и т. д.

Принцип распределения параметров прибора по областям памяти не описывается в спецификации и является ответственностью разработчика. Реализации могут быть разными.

Например, в рассмотренном нами в [уроке 2.1] датчике ПВТ110–RS все параметры прибора (включая измеренную температуру и влажность) размещены в области holding–регистров, что может вызвать диссонанс: ведь выше упоминается, что такие регистры доступны и для чтения, и для записи, а пытаться записать в датчик значение температуры и влажности просто не имеет смысла, ведь его задача как раз заключается в их измерении.

Но в документации на датчик указано, что конкретно эти (и ещё некоторые) параметры датчика доступны только для чтения (**ro** означает **read only**):

ВНИМАНИЕ
Гайку кабельного ввода следует заворачивать до упора.
При несоблюдении данного условия производитель не может гарантировать соответствие стандарту IP65.

8.3 Назначение контактов клеммника

Схема подключения прибора приведена на *рисунке 8.2*.

ВНИМАНИЕ
Во время подключения источника питания требуется соблюдать полярность!
Неправильное подключение может привести к порче оборудования.

Рисунок 8.2 – Схема подключения

9 Эксплуатация

9.1 Включение и работа

Во время работы прибор проверяет исправность подключенного измерительного зонда. Состояние прибора индицируется светодиодом «Статус» и передается в регистре «Состояние прибора», см. *п. 9.2– 9.3*.

9.2 Работа по интерфейсу RS-485

Прибор работает в режиме Slave по протоколу ModBus RTU.

Список параметров, доступных по сети RS-485, приведен в таблице ниже.

Таблица 9.1 – Параметры прибора, доступные по RS-485

Наименование параметра	Номер первого регистра		Кол-во регистров	Тип	Допустимые значения*	Тип доступа
	DEC	HEX				
Общие параметры						
Название датчика	1000	3E8	6	STRING [12]	PVT110	RO
Версия ПО	1006	3EE	3	STRING[6]	01.00 … 99.99	RO
Заводской номер	1104	450	10	STRING [20]	XXXXXXXXXXXXXXXXX	RO
Состояние датчика	1300	514	1	UC8	см. *регистр 0x0514*	RO
Управление прибором						
Команда управления	1400	578	1	UC8	bit[0] = 1 – программная перезагрузка прибора; bit[1] = 1 – сброс всех настроек на заводские	WO
Оперативные параметры						
Значение влажности, %RH	2200	898	2	FLOAT32	0,00…100	RO
Значение температуры, °C	2250	8CA	2	FLOAT32	-40,00…80,00	RO

Прибор поддерживает выполнение следующих функций ModBus:
- **03** – чтение значений из нескольких регистров хранения;
- **06** – запись значения в один регистр хранения;
- **16** – запись значения в несколько регистров хранения.

Прибор поддерживает коды ошибок ModBus:
- **01** – принятый код функции не может быть обработан;
- **02** – адрес данных, указанный в запросе, не доступен;
- **03** – величина, содержащаяся в поле данных запроса, является недопустимой.

9.3 Индикация

Светодиод расположен внутри электронного блока прибора.

Таблица 9.2 – Назначение светодиода

Свето-диод	Статус	Значение	
	Зеленый, непрерывно светится	Нормальная работа прибора	
	Красный, непрерывно светится	Отсутствует связь с зондом	
	Зеленый, непрерывно мигает	Выход за верхний предел измерения температуры	
	Красный, непрерывно мигает	Ошибочная конфигурация переключателя четности	
	Зеленый, быстро мигает	**Мигает на протяжении 0,5 с**	**Мигает на протяжении 1 с**
		Успешный прием пакета по RS-485	Подтверждение смены ручных настроек
	Красный, быстро мигает на протяжении 0,5 с	Ошибка при приеме пакета по RS-485	

10 Техническое обслуживание

Во время выполнения работ по техническому обслуживанию прибора следует соблюдать требования безопасности из раздела 5.

Техническое обслуживание прибора следует проводить не реже одного раза в 6 месяцев. Техническое обслуживание включает в себя следующие процедуры:
- проверка качества крепления прибора;
- проверка качества подключения внешних связей;
- удаление пыли и грязи с корпуса и клеммника прибора.

Обнаруженные при осмотре недостатки следует немедленно устранить.

Межповерочный интервал прибора – 1 год.

11 Маркировка

На корпус прибора нанесены:
- товарный знак;
- наименование и исполнение прибора;
- степень защиты корпуса по ГОСТ 14254-2015;
- напряжение питания;
- потребляемая мощность;

Как мы определили, что все параметры датчика размещены в области Holding–регистров?

На это указывает список поддерживаемых функций (выделен красным справа вверху на скриншоте выше) – из функций чтения поддерживается лишь **0x03 (Read Holding Registers)**.

Но вы также можете встретить датчики, в которых температура, влажность, давление и другие измеряемые ими параметры будут размещены в области Input–регистров – и, соответственно, для их чтения потребуется использовать функцию **0x04 (Read Inputs Registers)**.

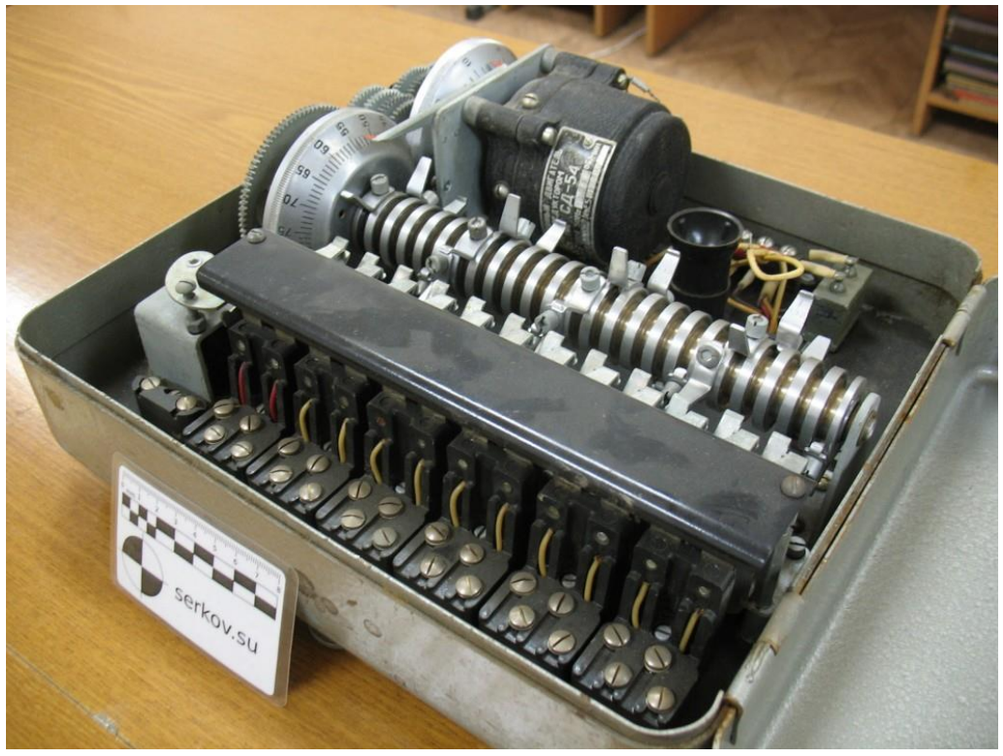
Постарайтесь запомнить, что идентификатор (столбец ID в таблице) области памяти не совпадает с кодом функции чтения данных этой области.

Мы вернёмся к этим идентификаторам в уроке 7.2, когда будем говорить о моделях адресации регистров.

Область памяти с идентификатором 2x

Если вы внимательны – то, возможно, заметили, что в таблице из предыдущего шага отсутствует область памяти с ID = **2x**.

В годы создания протокола эта область отвечала за доступ к битам барабанного командоаппарата (drum sequencer). Уже давно такие командоаппараты не используются – и поэтому необходимость в этой области исчезла.



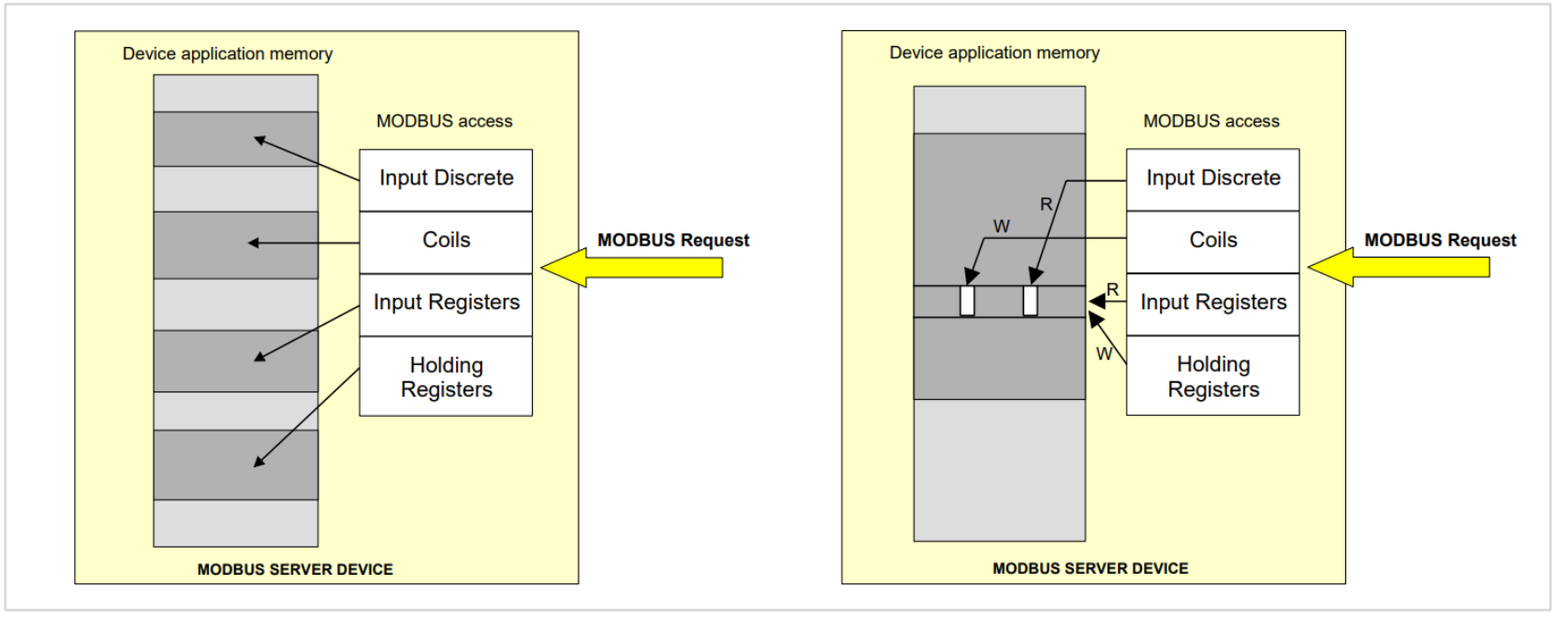
Источник фото: https://serkov.su/blog/?p=434

Модели памяти Modbus (часть 1)

Спецификация [3, п. 4.2] приводит два примера того, как данные, доступные по Modbus, могут быть размещены в памяти slave–устройства:

- независимые области памяти;
- общая область памяти.

Эти модели связаны с типовыми функциями работы с регистрами и функциями работы с битами; «специфические» функции (например, функции работы с файлами, функции диагностики и т. д.) обрабатываются обособленно – их данные не размещаются в этих областях.



Проще пояснить отличия этих областей на примере: пусть master–устройство отправляет 4 запроса с различными кодами функций:

- 0x01 (Read Coils);
- 0x02 (Read Discrete Inputs);
- 0x03 (Read Input Registers);
- 0x04 (Read Holding Registers).

и в каждом запросе считывает соответствующий объект (бит или регистр) с адресом 0.

Если slave–устройство имеет независимые области памяти – то в ответ на каждый из запросов оно отправит уникальное значение.

Если slave–устройство имеет общую модель памяти – то в ответ на запросы с кодами функций **0x03** и **0x04** будет отправлено одно и то же значение, а в ответ на запросы с кодами функций **0x01** и **0x02** будет отправлен нулевой (младший) бит этого же значения. То есть в случае общей модели – все поддерживаемые slave–устройством типовые функции работы с регистрами и битами применяются к **одной и той же области памяти.**

В редких случаях используется комбинированная модель памяти, в рамках которой у slave–устройства есть две независимые области:

- наложенные друг на друга Coils и Holding–регистры (т. е. Coil 0 – это бит 0 Holding–регистра 0);
- наложенные друг на друга Discrete Inputs и Input–регистры (т. е. Discrete Input 0 – это бит 0 Input–регистра 0).

В частности, такая модель долгое время была единственным доступным вариантом при настройке ПЛК, программируемого в среде CODESYS V3.5, в режиме slave–устройства.

Модели памяти Modbus (часть 2)

Максимальный размер области памяти составляет **65536** объектов [3, п. 4.3]:

- максимальный размер областей Coils и Discrete Inputs = 65536 бит = 8192 байта;
- максимальный размер областей Input–регистров и Holding–регистров = 65536 регистров = 131072 байта.

Обратите внимание, что речь идёт о максимальном размере – в конкретном устройстве размер области памяти может быть гораздо меньше.

Например, в датчике ПВТ110–RS присутствует всего 47 регистров.

Адреса параметров

Чтобы master–устройство могло прочитать конкретный параметр из slave–устройства – вам должен быть известен начальный адрес этого параметра и число объектов (битов или регистров), которые он занимает. Адрес является числом из диапазона **0...65535**.

Даже если гипотетически в приборе присутствует 65536 регистров (или бит), считать или записать их одним запросом не получится – функции чтения и записи имеют ограничения, которые мы обсудим в уроке 2.8.

Адреса параметров распределяет производитель устройства, и это может быть сделано совершенно произвольным образом. Например, параметры датчика ПВТ110–RS занимают следующие адреса регистров:

- 1000–1008
- 1104–1113
- 1300
- 1400
- 2200–2201
- 2250–2251
- 5302–5305
- 5310
- 5313–5314
- 5352–5355
- 5360
- 5363–5364
- 5601–5607

На вопрос «почему именно так?» мы, к сожалению, не можем дать какого–то осмысленного ответа; это решение разработчиков данного датчика.

***

Обратите внимание на то, что адреса параметров ПВТ110–RS распределены не последовательно, а с «разрывами».

Например, значение влажности занимает регистры 2200–2201, а значение температуры – 2250–2251.

Регистры 2202–2249 в приборе отсутствуют, и это делает невозможным считывание значений влажности и температуры одним **групповым запросом**. Master–устройству придётся отправить два отдельных запроса для получения этих значений, что, естественно, займёт больше времени по сравнению с тем случаем, если бы запрос был один.

Мы ещё отдельно обсудим групповые запросы, но поскольку сама концепция групповых запросов в Modbus является крайне важной – то было полезно заранее обратить ваше внимание на упомянутый факт.

Карта регистров

После предыдущих шагов вы знаете, что для опроса параметров прибора по Modbus вам нужна следующая информация:

- список этих параметров;
- их адреса;
- указание, в каких областях памяти они размещены, чтобы определить функции Modbus, которые нужно применить к этим адресам (обычно достаточно списка функций, поддерживаемых прибором);
- типы данных для их корректной интерпретации.

Таблица, которая содержит всю перечисленную информацию, называется «картой регистров Modbus» (в англоязычной документации часто используется термин **Modbus Map**). Обычно эта таблица входит в состав руководства на прибор или распространяется в виде отдельного документа.

Без её наличия – настроить обмен с прибором в подавляющем большинстве случаев не получится; вы просто не будете знать, какие параметры с него можно считать и записать.

Для наглядности – приведём опять в качестве иллюстрации фрагмент карты регистров датчика ПВТ110–RS из [урока 2.1:]

ВНИМАНИЕ
Гайку кабельного ввода следует заворачивать до упора.
При несоблюдении данного условия производитель не может гарантировать соответствие стандарту IP65.

8.3 Назначение контактов клеммника

Схема подключения прибора приведена на *рисунке 8.2*.

ВНИМАНИЕ
Во время подключения источника питания требуется соблюдать полярность!
Неправильное подключение может привести к порче оборудования.

Рисунок 8.2 – Схема подключения

9 Эксплуатация

9.1 Включение и работа

Во время работы прибор проверяет исправность подключенного измерительного зонда. Состояние прибора индицируется светодиодом «Статус» и передается в регистре «Состояние прибора», см. *п. 9.2– 9.3*.

9.2 Работа по интерфейсу RS-485

Прибор работает в режиме Slave по протоколу ModBus RTU.

Список параметров, доступных по сети RS-485, приведен в таблице ниже.

Таблица 9.1 – Параметры прибора, доступные по RS-485

Наименование параметра	Номер первого регистра		Кол-во регистров	Тип	Допустимые значения*	Тип доступа
	DEC	HEX				
Общие параметры						
Название датчика	1000	3E8	6	STRING [12]	PVT110	RO
Версия ПО	1006	3EE	3	STRING[6]	01.00 … 99.99	RO
Заводской номер	1104	450	10	STRING [20]	XXXXXXXXXXXXXXXXX	RO
Состояние датчика	1300	514	1	UC8	см. *регистр 0x0514*	RO
Управление прибором						
Команда управления	1400	578	1	UC8	bit[0] = 1 – программная перезагрузка прибора; bit[1] = 1 – сброс всех настроек на заводские	WO
Оперативные параметры						
Значение влажности, %RH	2200	898	2	FLOAT32	0,00…100	RO
Значение температуры, °C	2250	8CA	2	FLOAT32	-40,00…80,00	RO

Прибор поддерживает выполнение следующих функций ModBus:
- **03** – чтение значений из нескольких регистров хранения;
- **06** – запись значения в один регистр хранения;
- **16** – запись значения в несколько регистров хранения.

Прибор поддерживает коды ошибок ModBus:
- **01** – принятый код функции не может быть обработан;
- **02** – адрес данных, указанный в запросе, не доступен;
- **03** – величина, содержащаяся в поле данных запроса, является недопустимой.

9.3 Индикация

Светодиод расположен внутри электронного блока прибора.

Таблица 9.2 – Назначение светодиода

Свето-диод	Статус	Значение	
○	Зеленый, непрерывно светится	Нормальная работа прибора	
●	Красный, непрерывно светится	Отсутствует связь с зондом	
☼	Зеленый, непрерывно мигает	Выход за верхний предел измерения температуры	
✹	Красный, непрерывно мигает	Ошибочная конфигурация переключателя четности	
☼	Зеленый, быстро мигает	**Мигает на протяжении 0,5 с**	**Мигает на протяжении 1 с**
		Успешный прием пакета по RS-485	Подтверждение смены ручных настроек
✹	Красный, быстро мигает на протяжении 0,5 с	Ошибка при приеме пакета по RS-485	

10 Техническое обслуживание

Во время выполнения работ по техническому обслуживанию прибора следует соблюдать требования безопасности из раздела 5.

Техническое обслуживание прибора следует проводить не реже одного раза в 6 месяцев. Техническое обслуживание включает в себя следующие процедуры:
- проверка качества крепления прибора;
- проверка качества подключения внешних связей;
- удаление пыли и грязи с корпуса и клеммника прибора.

Обнаруженные при осмотре недостатки следует немедленно устранить.

Межповерочный интервал прибора – 1 год.

11 Маркировка

На корпус прибора нанесены:
- товарный знак;
- наименование и исполнение прибора;
- степень защиты корпуса по ГОСТ 14254-2015;
- напряжение питания;
- потребляемая мощность;

Подведение итогов

Мы ознакомились с важными аспектами спецификации Modbus:

- принципом адресации slave–устройств;
- типами данных (которых в спецификации описано всего 2 – биты и регистры);
- функциями, которые соответствуют различным операциям над данными (чтению и записи);
- областям памяти slave–устройств, к которым применяются эти функции;
- моделям этих областей памяти.

Обсудили, какие проблемы создает отсутствие в протоколе полноценной типизации и неуточнение порядка регистров при передаче параметров, размер которых превышает один регистр.

Выяснили, что для опроса slave–устройства нам необходима его карта регистров.

Вскользь упомянули групповые запросы.

Теперь давайте применим эти знания на практике для опроса терморегулятора ТРМ201.

2.6 Практика: опрос ТРМ201

ТРМ201 от компании ОВЕН – это представитель линейки терморегуляторов ТРМ2xx, история которой уходит к началу 2000–х. Уже первые модели линейки имели на борту интерфейс RS–485, что позволяло подключать их к ПК (в те времена ещё активно использовалась аббревиатура ЭВМ) для передачи показаний в системы диспетчеризации. Правда, изначально ТРМ поддерживали только протокол ОВЕН; поддержка Modbus появилась лишь спустя несколько лет.

За прошедшие два десятилетия линейка ТРМ2xx получила несколько аппаратных и программных обновлений. В конце декабря 2023 года она была снята с производства. Тем не менее, посмотреть, как была реализована работа с протоколом Modbus в старых ТРМ, будет довольно интересным и полезным.

ТРМ201 имеет один универсальный аналоговый вход и один выход, тип которого определяется модификацией.

Назначение ТРМ201 – получать значение контролируемого параметра (обычно – температуры), измеренной с помощью датчика, подключенного ко входу ТРМ, и поддерживать значение этого параметра максимально близким к значению заданной в настройках ТРМ уставки с помощью изменения значения выхода (к которому подключен исполнительный механизм – например, вентилятор).

Схема подключения

Схема подключения практически не отличается от той, которую мы использовали в уроке 2.1:

- ТРМ201 подключен к ПК с помощью конвертера интерфейсов АС4–М.

Все приборы линейки ТРМ2xx имеют диапазон питания 90...245 В, что позволяет подключать их напрямую к сети питания 220 В; использование отдельного блока питания в данном случае не требуется.

Для получения значения на аналоговом входе потребуется подключить к нему датчик температуры (список поддерживаемых датчиков приведён в руководстве по эксплуатации ТРМ201) и выбрать тип этого датчика в настройках прибора.

Если же у вас нет в наличии датчика – вы можете выбрать в настройках тип датчика **ТХК (L)** [или любую другую термопару, в диапазон измерений которой попадает комнатная температура] и соединить проводом клеммы 10 и 11. Это приведёт к тому, что измеренное значение будет получено от встроенного термодатчика ТРМ (температура холодного спая) и будет приблизительно равно температуре в помещении.

Карта регистров (часть 1)

Документация на ТРМ201 доступна по ссылке:

https://owen.ru/product/trm201/documentation_and_software

Нас интересует документ, который называется «Краткая инструкция ТРМ201 по работе с Modbus».

Краткая инструкция по работе с измерителем-регулятором одноканальным ТРМ201 по интерфейсу RS-485

Работа по протоколам Modbus RTU и Modbus ASCII

Перечень поддерживаемых функций Modbus

Функция (hex)	Действие	Примечание
03	Получение текущего значения одного или нескольких регистров	
10	Запись значений в несколько регистров	Устанавливается ограничение на запись только одного регистра
08	Диагностика. Получение данных о состоянии линии связи.	Поддерживается только код 00 - Вернуть запрос, который используется для проверки соединения между Master и Slave

Перечень поддерживаемых стандартных кодов ошибок MODBUS

Код	Ошибка	Примечание
01	ILLEGAL FUNCTION	Принятый код функции не поддерживается
02	ILLEGAL DATA ADDRESS	Адрес данных (№ регистра), указанный в запросе, не используется
03	ILLEGAL DATA VALUE	Некорректные данные – принятое значение находится вне допустимого диапазона; – длина ответа превышает размер буфера связи; – количество реальных байт данных в пакете не соответствует указанной длине пакета
04	SLAVE DEVICE FAILURE	Невозможность выполнения команды Подробно характер ошибки можно узнать, считав значение регистра 0108H, которое должно соответствовать коду ошибки N.Err для протокола ОВЕН

Перечень регистров Modbus

Параметр Имя ОВЕН	Назначение	Адрес Modbus (hex)	Тип данных	Кол-во знаков после запятой	Диапазон значений (dec)
Группа LvoP. Оперативные параметры (только чтение: Modbus-функция 0x03)					
STAT	Регистр статуса	0x 0000	binary	–	16 бит [1]
PV	Измеренная величина	0x 0001	Signed Int16	*	
Группа LvoP. Рабочие параметры (чтение: Modbus-функция 0x03, запись: Modbus-функция 0x10)					
SP	Уставка регулятора	0x 0002	Signed Int16	*	
r-L	Перевод канала на внешнее управление	0x 0003	Int16	0	0,1
r.out	Выходной сигнал	0x 0004	Int16	3	0,1 (ВУ ключевого типа) 0.000 … 1.000 (ВУ аналогового типа)
Группа LvoP. Оперативные параметры (только чтение: Modbus-функция 0x03)					
DEV	Тип прибора	0x 1000 0x 1001 0x 1002 0x 1003	Char[8]	–	ТРМ201
VER	Версия прибора	0x 1004 0x 1005 0x 1006 0x 1007	Char[8]	–	V03.00xx
STAT	Регистр статуса	0x 1008	binary	–	16 бит [1]
PV	Измеренная величина	0x 1009 0x 100A	Float32	–	
SP	Уставка регулятора	0x100B 0x100C	Float32	–	

Параметр Имя ОВЕН	Назначение	Адрес Modbus (hex)	Тип данных	Кол-во знаков после запятой	Диапазон значений (dec)
Группа Adv. Параметры индикации (чтение: Modbus-функция 0x03 / запись: Modbus-функция 0x10)					
rESt	Время выхода из режима программирования	0x 0300	Int16	0	5…100
Группа LvoU.Настройки регулирования и регистрации (чтение: Modbus-функция 0x03 / запись: Modbus-функция 0x10)					
SL.L	Нижняя граница задания уставки	0x 0400	Signed Int16	*	диапазон изм. датчика
SL.H	Верхняя граница задания уставки	0x 0401	Signed Int16	*	диапазон изм. датчика
CmP	Тип логики компаратора	0x 0402	Int16	0	0 – выкл; 1 – нагреватель; 2 – холодильник; 3 – П-образная, 4 – U-образная
HYS	Гистерезис для компаратора	0x 0403	Int16	*	0…9999
don	Задержка включения компаратора	0x 0404	Int16	0	0 … 250
doF	Задержка выключения компаратора	0x 0405	Int16	0	0 … 250
ton	Минимальное время удерживания компаратора во вкл. состоянии	0x 0406	Int16	0	0 … 250
toF	Минимальное время удерживания компаратора в выкл. состоянии	0x 0407	Int16	0	0 … 250
oEr	Состояние выхода в режиме «ошибка»	0x 0408	Int16	0	0 – выкл; 1 – вкл

Он включает в себя:

1. Список поддерживаемых прибором функций Modbus

ТРМ201 поддерживает:

- функцию чтения 0x03 (Read Holding Registers);
- функцию записи 0x10 (Write Multiple Registers);
- специфическую функцию 0x08 (Diagnostics).

Обратите внимание, что функция 0x10 **в рамках одного запроса позволяет записать только один регистр**.

Это ограничение прибора, и оно может создавать определённые неудобства: например, чтобы записать значения всех настроек аналогового входа, нам бы потребовалось отправить 10 запросов (по одному на каждый из 10 параметров входа). При отсутствии ограничения нам хватило бы одного запроса – и, конечно же, его выполнение заняло бы меньше времени.

Функция **0x08** относится к числу редко используемых функций; мы рассмотрим её в уроке 2.15.

В реализации ТРМ201 эта функция позволяет выполнить лишь одну задачу – проверить наличие связи с прибором; в принципе, для этой цели можно воспользоваться и функцией **0x03**.

2. Перечень кодов ошибок, которые может вернуть ТРМ201 в ответ на запрос

О кодах ошибок мы поговорим в уроке 2.8.

3. Список параметров ТРМ201

Для каждого параметра указан:

- адрес начального регистра (**обратите внимание** – все адреса указаны в **HEX** – т.е. шестнадцатеричной системе счисления);
- тип данных (описание типов приведено после таблицы);
- диапазон возможных значений.

В названии групп параметров указано, какие из них поддерживают функцию записи **0x10** (соответственно, остальные параметры доступны только для чтения).

Для некоторых параметров важен столбец «Количество знаков после запятой».

Дело в том, что ТРМ201 имеет только два параметра типа **Float32** (значение с плавающей точкой), и они доступны только для чтения.

Другие параметры имеют тип **Int16** (беззнаковое целое) или **Signed Int16** (знаковое целое).

Например, уставка регулятора (**SP**) как раз имеет тип **Signed Int16**, для которого используется «смещение десятичной точки», величина которого определяется значением параметра **dP**.

«Реальное» (обрабатываемое логикой прибора) значение **SP** вычисляется по формуле:

$$\frac{\text{Значение регистра } SP}{10^{dP}}$$

То есть если **dP** = **2**, и в регистр **SP** записано значение **3556** – то «реальное» значение **SP** составит **35.56** (десятичная точка смещается на два знака влево).

Карта регистров (часть 2)

Давайте определим список параметров, которые нас интересуют.

Мы будем считывать следующие параметры:

- DEV – модификация прибора;
- VER – версия прошивки;
- STAT – регистр статуса;
- PV – измеренная величина на входе прибора;
- SP – уставка.

> *PV означает «process value» (измеренное значение), а SP – «setpoint» (уставка).*
>
> *Остальные сокращения означают DEVICE, VERSION и STATUS.*
>
> *Сокращение названий параметров связано с тем, что дисплей ТРМ2хх является 4–сегментным и, соответственно, может отобразить только 4 символа.*

Мы можем считать все эти параметры одним запросом, потому что их адреса регистров (0x1000 – 0x100C) размещены «вплотную» друг к другу.

Мы будем записывать только параметр SP – уставку регулятора.

> *Обратите внимание, что считывать значение уставки мы будем в виде значения типа Float32, а записывать – в виде значения типа Signed16. Это связано с тем, что значение уставки в виде Float32 доступно только для чтения из–за особенностей ПО прибора.*

Краткая инструкция по работе с измерителем-регулятором одноканальным ТРМ201 по интерфейсу RS-485

Работа по протоколам Modbus RTU и Modbus ASCII

Перечень поддерживаемых функций Modbus

Функция (hex)	Действие	Примечание
03	Получение текущего значения одного или нескольких регистров	
10	Запись значений в несколько регистров	Устанавливается ограничение на запись только одного регистра
08	Диагностика. Получение данных о состоянии линии связи.	Поддерживается только код 00 - Вернуть запрос, который используется для проверки соединения между Master и Slave

Перечень поддерживаемых стандартных кодов ошибок MODBUS

Код	Ошибка	Примечание
01	ILLEGAL FUNCTION	Принятый код функции не поддерживается
02	ILLEGAL DATA ADDRESS	Адрес данных (№ регистра), указанный в запросе, не используется
03	ILLEGAL DATA VALUE	Некорректные данные – принятое значение находится вне допустимого диапазона; – длина ответа превышает размер буфера связи; – количество реальных байт данных в пакете не соответствует указанной длине пакета
04	SLAVE DEVICE FAILURE	Невозможность выполнения команды Подробно характер ошибки можно узнать, считав значение регистра 0108H, которое должно соответствовать коду ошибки N.Err для протокола ОВЕН

Перечень регистров Modbus

Параметр Имя ОВЕН	Назначение	Адрес Modbus (hex)	Тип данных	Кол-во знаков после запятой	Диапазон значений (dec)
Группа LvoP. Оперативные параметры (только чтение: Modbus-функция 0x03)					
STAT	Регистр статуса	0x 0000	binary	–	16 бит [1]
PV	Измеренная величина	0x 0001	Signed Int16	*	
Группа LvoP. Рабочие параметры (чтение: Modbus-функция 0x03, запись: Modbus-функция 0x10)					
SP	Уставка регулятора	0x 0002	Signed Int16	*	
r-L	Перевод канала на внешнее управление	0x 0003	Int16	0	0,1
r.out	Выходной сигнал	0x 0004	Int16	3	0,1 (ВУ ключевого типа) 0.000 … 1.000 (ВУ аналогового типа)
Группа LvoP. Оперативные параметры (только чтение: Modbus-функция 0x03)					
DEV	Тип прибора	0x 1000 0x 1001 0x 1002 0x 1003	Char[8]	–	TPM201
VER	Версия прибора	0x 1004 0x 1005 0x 1006 0x 1007	Char[8]	–	V03.00xx
STAT	Регистр статуса	0x 1008	binary	–	16 бит [1]
PV	Измеренная величина	0x 1009 0x 100A	Float32	–	
SP	Уставка регулятора	0x100B 0x100C	Float32	–	

Параметр Имя ОВЕН	Назначение	Адрес Modbus (hex)	Тип данных	Кол-во знаков после запятой	Диапазон значений (dec)
Группа Adv. Параметры индикации (чтение: Modbus-функция 0x03 / запись: Modbus-функция 0x10)					
rESt	Время выхода из режима программирования	0x 0300	Int16	0	5…100
Группа LvoU.Настройки регулирования и регистрации (чтение: Modbus-функция 0x03 / запись: Modbus-функция 0x10)					
SL.L	Нижняя граница задания уставки	0x 0400	Signed Int16	*	диапазон изм. датчика
SL.H	Верхняя граница задания уставки	0x 0401	Signed Int16	*	диапазон изм. датчика
CmP	Тип логики компаратора	0x 0402	Int16	0	0 – выкл; 1 – нагреватель; 2 – холодильник; 3 – П-образная, 4 – U-образная
HYS	Гистерезис для компаратора	0x 0403	Int16	*	0…9999
don	Задержка включения компаратора	0x 0404	Int16	0	0 … 250
doF	Задержка выключения компаратора	0x 0405	Int16	0	0 … 250
ton	Минимальное время удерживания компаратора во вкл. состоянии	0x 0406	Int16	0	0 … 250
toF	Минимальное время удерживания компаратора в выкл. состоянии	0x 0407	Int16	0	0 … 250
oEr	Состояние выхода в режиме «ошибка»	0x 0408	Int16	0	0 – выкл; 1 – вкл

Настройка ТРМ201

Чтобы настроить связь с прибором – нам потребуется узнать значения его сетевых настроек.

К ним относятся:

- протокол обмена данными (**Prot**);
- скорость обмена (**bPS**);
- адрес ТРМ как slave–устройства (**Addr**).

Изменить параметры можно с помощью дисплея и кнопок ТРМ. Как это сделать – описано в руководстве по эксплуатации:

https://owen.ru/product/trm201/documentation_and_software

После изменения настроек нужно перезагрузить прибор по питанию, чтобы они вступили в силу.

Наш ТРМ201 имеет следующие настройки:

- Протокол – **Modbus RTU**;
- Скорость обмена = **115200**;
- Адрес = **16**.

Обратите внимание, что из настроек COM–порта прибора для редактирования доступна только скорость. Все остальные параметры имеют фиксированные значения:

- количество бит данных = **8**;
- режим контроля чётности = **нет**;
- количество стоп–бит = **2**.

Параметр **A.Len** относится к протоколу ОВЕН – нас он не интересует.

Параметр **rSdl** (Response Delay) определяет время задержки, которую ТРМ выдерживает между получением запроса и отправкой ответа. Эта задержка позволяет master–устройству успеть переключить свой последовательный порт из режима приёма в режим передачи. Мы обсудим её более подробно в уроке 2.12.

7.6 Настройка обмена данными через интерфейс RS-485

Настройка обмена данными осуществляется параметрами группы *Conn*:

- *Prot* – протокол обмена данными (ОВЕН, ModBus-RTU, ModBus-ASCII);
- *bPS* – скорость обмена в сети, допустимые значения – 2400, 4800, 9600, 14400 19200, 28800, 38400, 57600, 115200 бит/с;
- *Addr* – базовый адрес прибора, диапазон значений:
  - 0…255 при *Prot* = *oUEn* и *ALEn* = 8;
  - 0…2047 при *Prot* = *oUEn* и *ALEn* = 11;
  - 1…247 при *Prot* = *n.rtU* или *n.ASC*.
- *ALEn* – длина сетевого адреса (8 или 11 бит);
- *rSdL* – задержка ответа прибора по RS-485 (1–45 мс).

Значения параметров обмена, которые не отображаются на цифровом индикаторе, т. к. их нельзя изменить вручную, перечислены в таблице 7.1.

Таблица 7.1 – Фиксированные параметры обмена данными

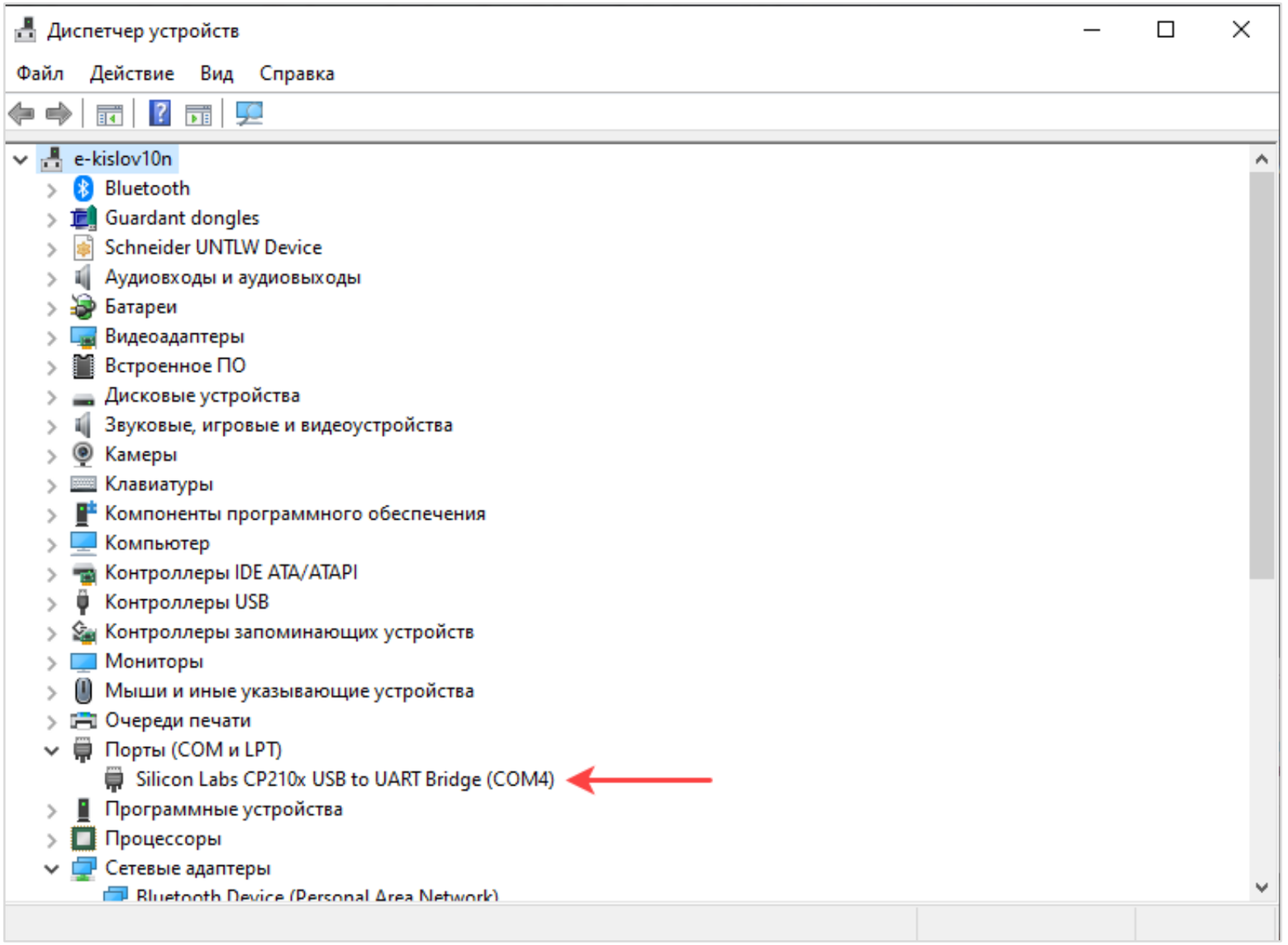
Параметр	Имя	Протокол		
		ОВЕН	ModBus RTU	ModBus ASCII
Количество стоп-бит	*Sbit*	1	2	2
Длина слова данных	*LEn*	8 бит	8 бит	7 бит
Контроль четности	*PrtY*	нет	нет	нет

ВНИМАНИЕ
Новые значения параметров обмена вступают в силу только после перезапуска прибора (снятия и подачи питания) или перезапуска по RS-485.

ПРЕДУПРЕЖДЕНИЕ
Минимальный период опроса параметров по протоколу ОВЕН для приборов с ВУ аналогового типа должен быть не менее 0,5 секунды.

Итак, мы подключили ТРМ201 к ПК с помощью конвертера интерфейсов АС4–М.

Напомним, что номер виртуального COM–порта, создаваемого конвертером, можно посмотреть в Диспетчере устройств Windows:


Диспетчер устройств
Файл
Действие
Вид
Справка
e-kislov10n
Bluetooth
Guardant dongles
Schneider UNTLW Device
Аудиовходы и аудиовыходы
Батареи
Видеоадаптеры
Встроенное ПО
Дисковые устройства
Звуковые, игровые и видеоустройства
Камеры
Клавиатуры
Компоненты программного обеспечения
Компьютер
Контроллеры IDE ATA/ATAPI
Контроллеры USB
Контроллеры запоминающих устройств
Мониторы
Мыши и иные указывающие устройства
Очереди печати
Порты (COM и LPT)
Silicon Labs CP210x USB to UART Bridge (COM4)
Программные устройства
Процессоры
Сетевые адаптеры

MasterOPC Universal Modbus Server

В шаге 2.1 мы настроили опрос датчика ПВТ110–RS с помощью программы Owen OPC Server.

В этом и следующих практических уроках вместо неё мы будет использовать MasterOPC Universal Modbus Server, разработанный компанией МПС софт – схожую программу с расширенным функционалом.

Её демонстрационную версию можно загрузить с сайта производителя:

https://masteropc.ru/download

Доступны варианты для ОС Windows и Linux.

Существуют две демонстрационные версии:

- с ограничением на количество опрашиваемых параметров (тегов) – 32 штуки, но без ограничения на время работы;
- с ограничением на время работы (1 час), но без ограничения на количество опрашиваемых тегов. Спустя час непрерывной работы программа будет остановлена. Вы сможете сразу же перезапустить её.

Если вы хотите избавиться от ограничений – следует приобрести лицензию (см. ссылки внизу страницы):

https://masteropc.ru/price

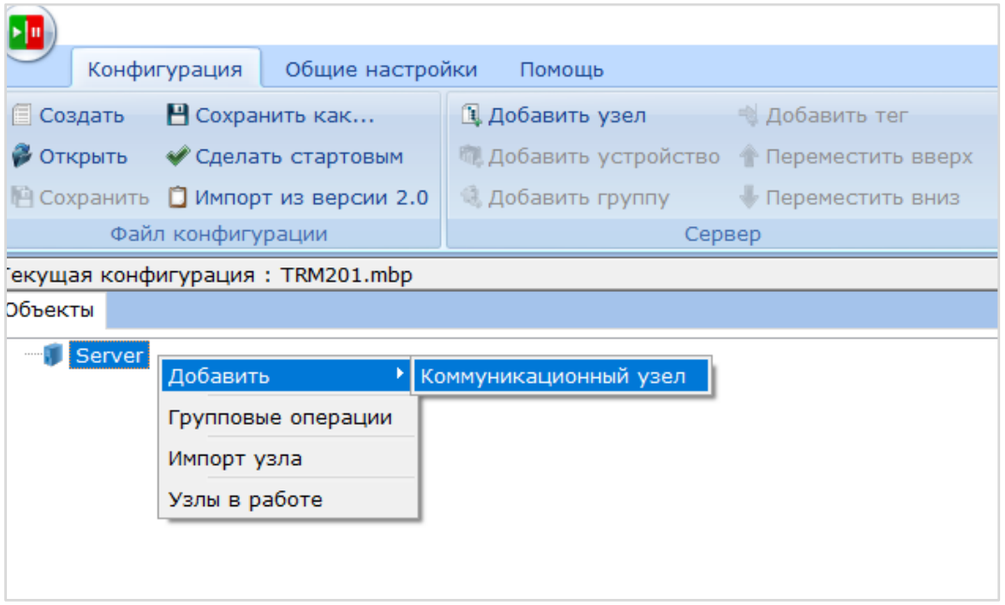
В рамках данного учебного курса будет достаточно демонстрационной версии; мы рекомендуем использовать версию с ограничением на время работы – так как в ней нет ограничений на количество тегов, то вы сможете использовать её даже для опроса устройств со значительным количеством параметров.

Загрузите утилиту, установите её на свой ПК и запустите.

Настройка опроса в MasterOPC Universal Modbus Server (часть 1)

Нажмите правой кнопкой мыши на узел **Server** и добавьте коммуникационный узел:

> *После первого запуска OPC–сервера может автоматически открыться демонстрационная конфигурация, в дереве проекта которой будут присутствовать различные узлы. Вы можете создать новую пустую конфигурацию (с помощью кнопки **Создать**) или удалить уже присутствующие узлы – нажимайте на них правой кнопкой мыши и используйте команду **Удалить**.*

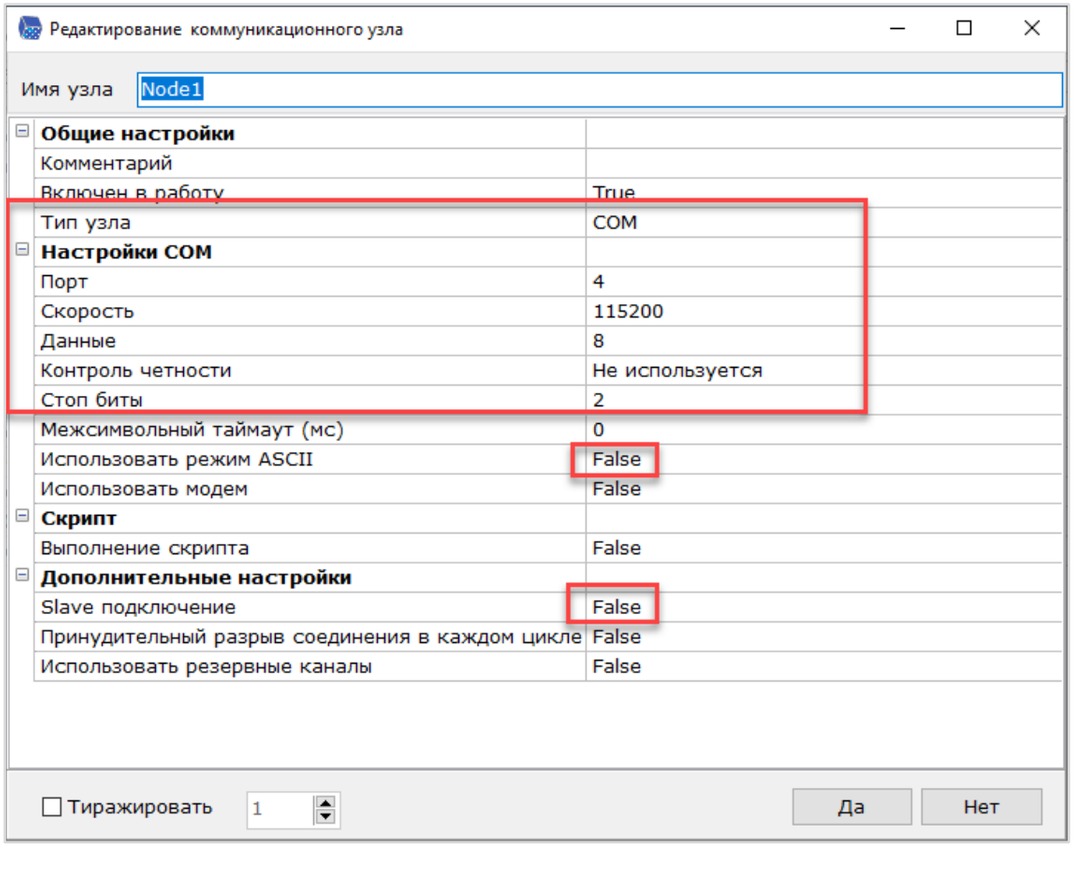


Этот узел соответствует сетевому интерфейсу компьютера. В конфигурации OPC–сервера их может быть несколько – чтобы вести опрос по различным интерфейсам одновременно. Нам хватит и одного.

В настройках узла укажите:

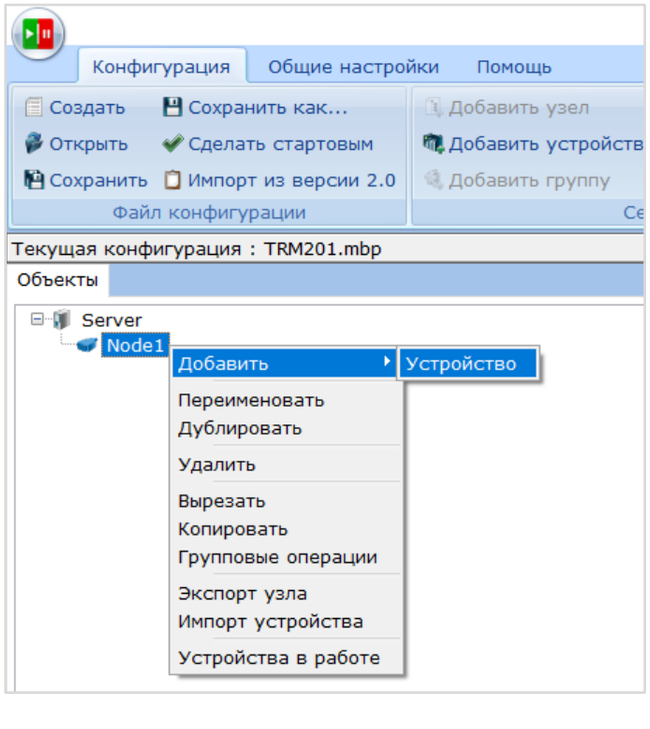
- тип интерфейса (мы выбираем **COM**, так как используем для обмена последовательный порт);
- номер COM–порта (у нас он равен **4** – он был показан два шага назад);
- сетевые настройки COM–порта (они были указаны три шага назад).

Убедитесь, что параметры **Использовать режим ASCII** и **Slave подключение** имеют значение **FALSE** – потому что в нашем случае мы используем OPC–сервер в режиме Modbus RTU Master.


Редактирование коммуникационного узла
Имя узла
Node1
Общие настройки
Комментарий
Включен в работу
True
Тип узла
COM
Настройки COM
Порт
4
Скорость
115200
Данные
8
Контроль четности
Не используется
Стоп биты
2
Межсимвольный таймаут (мс)
0
Использовать режим ASCII
False
Использовать модем
False
Скрипт
Выполнение скрипта
False
Дополнительные настройки
Slave подключение
False
Принудительный разрыв соединения в каждом цикле
False
Использовать резервные каналы
False
Тиражировать
1
Да
Нет

Настройка опроса в MasterOPC Universal Modbus Server (часть 2)

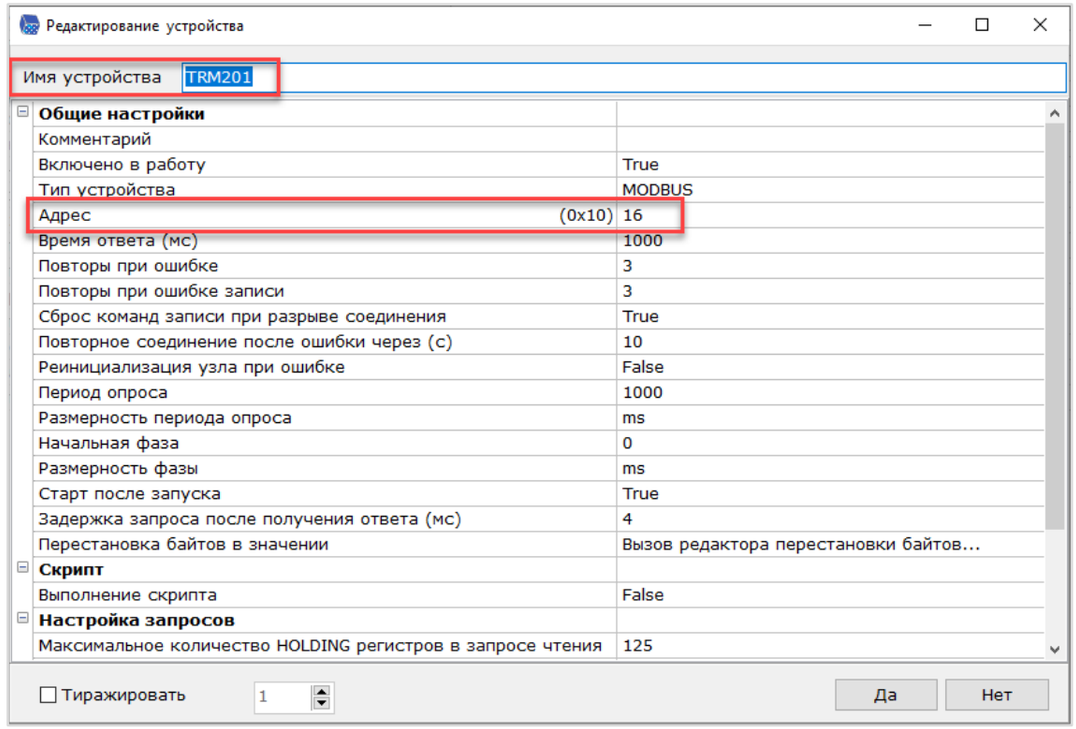
Нажмите правой кнопкой мыши на коммуникационный узел и добавьте **Устройство**:



Этот узел соответствует одному опрашиваемому slave–устройству. В конфигурации OPC–сервера их может быть несколько, но так как мы опрашиваем только один прибор – ТРМ201 – нам достаточно одного узла.

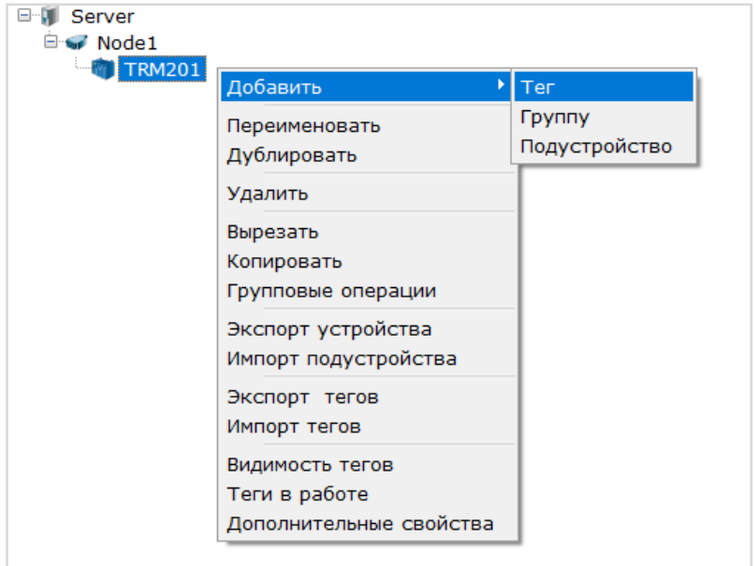
В настройках узла укажите его название и адрес slave–устройства (напомним, наш ТРМ201 имеет адрес **16**). Большинство других доступных настроек мы обсудим в модуле 6.

> *В пределах последовательной линии связи каждое slave–устройство должно иметь **уникальный** адрес.*
>
> *Вы можете вспомнить, что датчик ПВТ110–RS из урока 2.1 тоже имел адрес **16**.*
>
> *Что ж, мы не сможем совместно использовать их на одной линии связи!*
>
> *Но, строго говоря, это не получилось бы, даже если бы датчик имел другой адрес – потому что у него другие настройки последовательного порта (в частности, заданная скорость обмена датчика = 9600). Так что для успешного совместного опроса обоих устройств потребовалось бы изменить и их тоже.*


Редактирование устройства
Имя устройства
TRM201
Общие настройки
Комментарий
Включено в работу
True
Тип устройства
MODBUS
Адрес (0x10)
16
Время ответа (мс)
1000
Повторы при ошибке
3
Повторы при ошибке записи
3
Сброс команд записи при разрыве соединения
True
Повторное соединение после ошибки через (с)
10
Реинициализация узла при ошибке
False
Период опроса
1000
Размерность периода опроса
ms
Начальная фаза
0
Размерность фазы
ms
Старт после запуска
True
Задержка запроса после получения ответа (мс)
4
Перестановка байтов в значении
Вызов редактора перестановки байтов...
Скрипт
Выполнение скрипта
False
Настройка запросов
Максимальное количество HOLDING регистров в запросе чтения
125
Тиражировать
1
Да
Нет

Настройка опроса в MasterOPC Universal Modbus Server (часть 3)

Нажмите правой кнопкой мыши на узел **Устройство** (в прошлом шаге мы дали ему имя **TRM201**) и добавьте тег. Каждый тег соответствует одному параметру ТРМ – поэтому нам потребуется добавить 6 тегов.



Ниже приведены требуемые настройки для каждого из тегов – задайте их.

Все остальные настройки следует оставить в значениях по умолчанию.

При вводе адреса регистра – вводите его с префиксом **0x** (например, **0x1000**).

После ввода в таком формате OPC–сервер автоматически отобразит адрес регистра в десятичной системе (для **0x1000** отобразится **4096** и т. д.)

Имя тега	Регион	Адрес	Тип данных в устройстве	Тип данных в сервере	Количество байт для строкового типа	Тип доступа
DEV	HOLDING_ REGISTERS	0x1000	STRING	STRING	8	ReadOnly
VER	HOLDING_ REGISTERS	0x1004	STRING	STRING	8	ReadOnly

STAT	HOLDING_ REGISTERS	0x1008	Uint16	Uint32	–	ReadOnly
PV	HOLDING_ REGISTERS	0x1009	Float	Float	–	ReadOnly
SP_read	HOLDING_ REGISTERS	0x100B	Float	Float	–	ReadOnly
SP_write	HOLDING_ REGISTERS	0x0002	Int16	Int32	–	ReadWrite

Значения этих настроек взяты из документа «Краткая инструкция ТРМ201 по работе с Modbus», который мы рассмотрели на шаге 2.

У вас может возникнуть вопрос насчёт типов данных – как можно догадаться, что для параметров **DEV** и **VER**, для которых в руководстве указан тип **char[8]** нужно задать тип **STRING** с настройкой **Количество байт для строкового типа** = **8**?

Действительно, в документах и ПО разных производителей (и даже одного и того же производителя) одни и те же типы данных могут обозначаться по–разному. Настроив обмен с десятками различных устройств – вы поймёте, как все эти обозначения соотносятся между собой. Применительно к ТРМ201 – в руководстве по Modbus после таблицы параметров есть комментарий по типам, и для char[8] в описании типа используется термин «**string**»:

Типы данных

Тип данных	Описание
Int16	Двухбайтовое целое. На каждый параметр типа Integer отводится один регистр Modbus. Для параметров, значения которых могут иметь отрицательное значение (Signed Int16), отрицательные числа представляются в дополнительном коде. Передача данных осуществляется в формате $X*10^{-n}$, где X – передаваемое целое число, n – непередаваемая степень 10 (для каждого параметра она указывается в столбце «Кол-во знаков после запятой»).
float32	Четырехбайтовое с плавающей точкой. На каждый параметр типа Float отводится 2 соседних регистра Modbus. В регистре с младшим номером хранится старшая часть числа (high word), в регистре с большим номером – младшая часть числа (low word). Передача числа осуществляется по принципу «старшим вперед» (high byte first – high word first).
Char[8]	Строка из 8 символов. На каждый параметр типа String отводится 4 соседних регистра Modbus. В регистре с младшим номером хранятся первые два символа строки, в регистре с большим номером – последние. Для данных типа String, в отличие от протокола ОВЕН, используется прямой порядок следования символов (первым передается первый символ из строки).
Hex word	Двухбайтовое число в шестнадцатеричном формате
Binary	Двухбайтовое число в двоичном формате. При передаче первым следует пятнадцатый бит, последним – нулевой.

Ещё несколько возможных вопросов связаны с параметрами **STAT** и **SP_Write**.

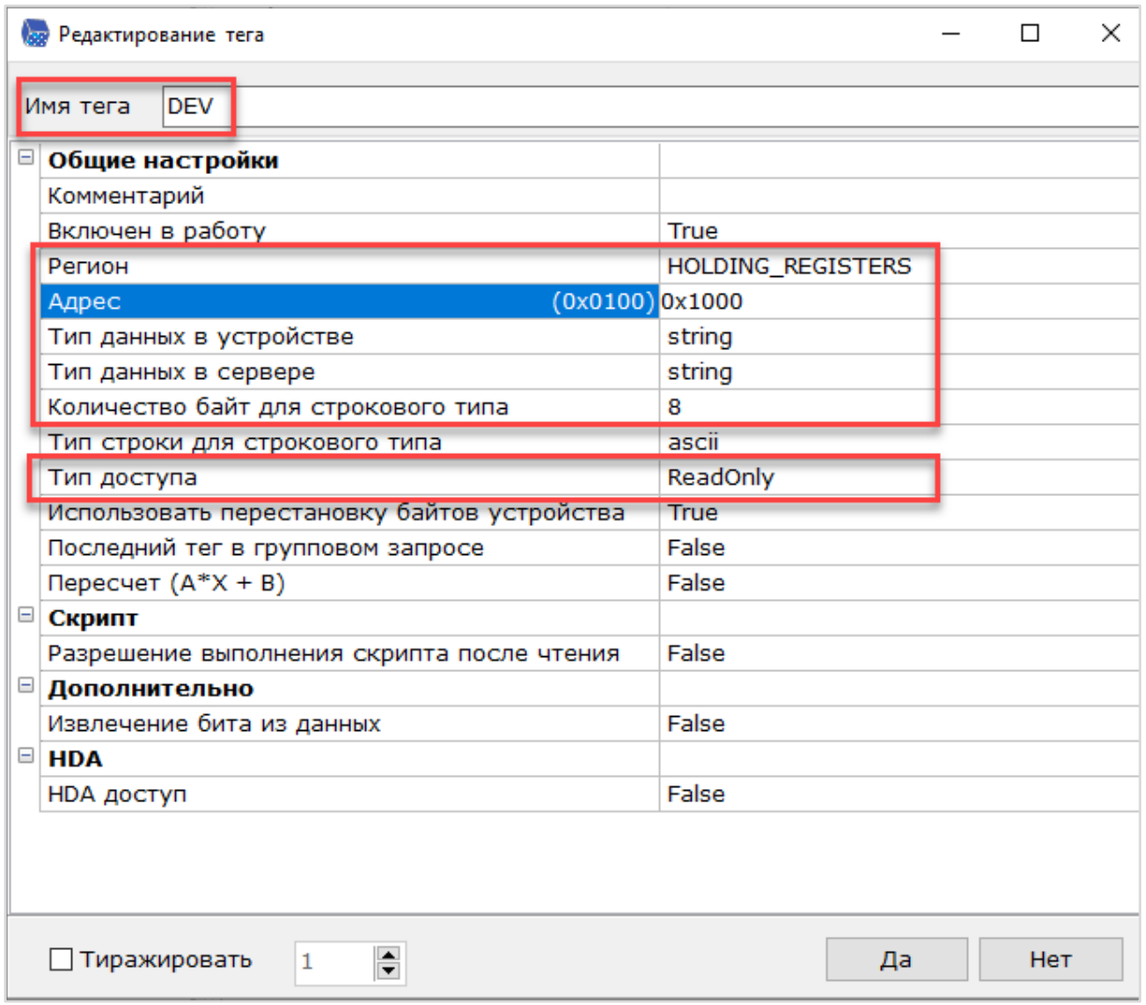
- почему у них «**Тип данных в сервере**» отличается от «**Типа данных в устройстве**»?
- и чем отличаются эти настройки?

Настройка **Тип данных в устройстве** позволяет OPC–серверу определить, сколько регистров занимает параметр, и как его правильно интерпретировать для отображения пользователю. Эта настройка **влияет** на формирование запроса, который OPC–сервер отправляет прибору.

Настройка **Тип данных в сервере** позволяет задать тип, который OPC–сервер будет использовать для передачи данных OPC–клиенту. Эта настройка **не влияет** на формирование запроса, который OPC–сервер отправляет прибору.

Вообще, оба типа должны совпадать, но из–за нюансов реализации OPC–сервера – в нём попросту нет типов данных **Uint16** и **Int16**. Поэтому для параметров этих типов в настройке **Тип данных в сервере** следует указать соответственно **Uint32** и **Int32**.

Пример настроек тега **DEV** приведён ниже.

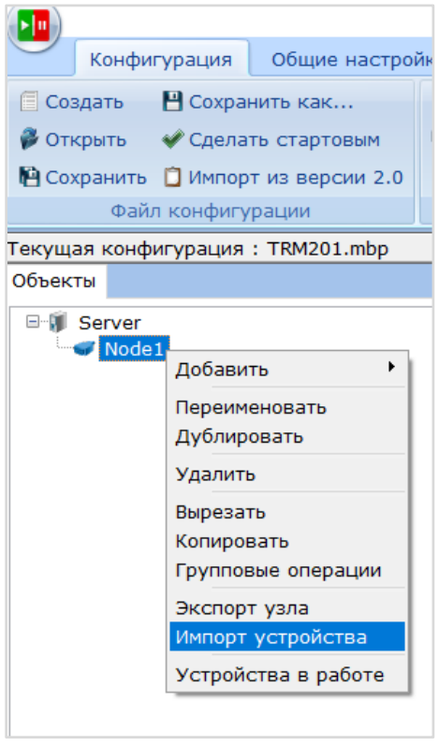


На всякий случай – мы подготовили и выложили файл готового устройства со всеми настроенными тегами (аналог шаблона из Owen OPC Server).

Загрузить его можно по ссылке:

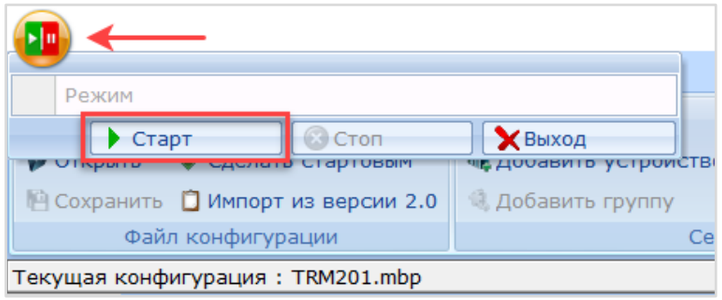
https://ftp.owen.ru/ModbusCourse/01_OpcTemplates/TRM201.sdv

Для импорта устройства нужно нажать правой кнопкой мыши на коммуникационный узел и использовать команду **Импорт устройства**:



Проверка обмена

Запустите OPC–сервер. Для этого нажмите на круглую кнопку, расположенную в верхнем левом углу, и используйте команду **Старт**.

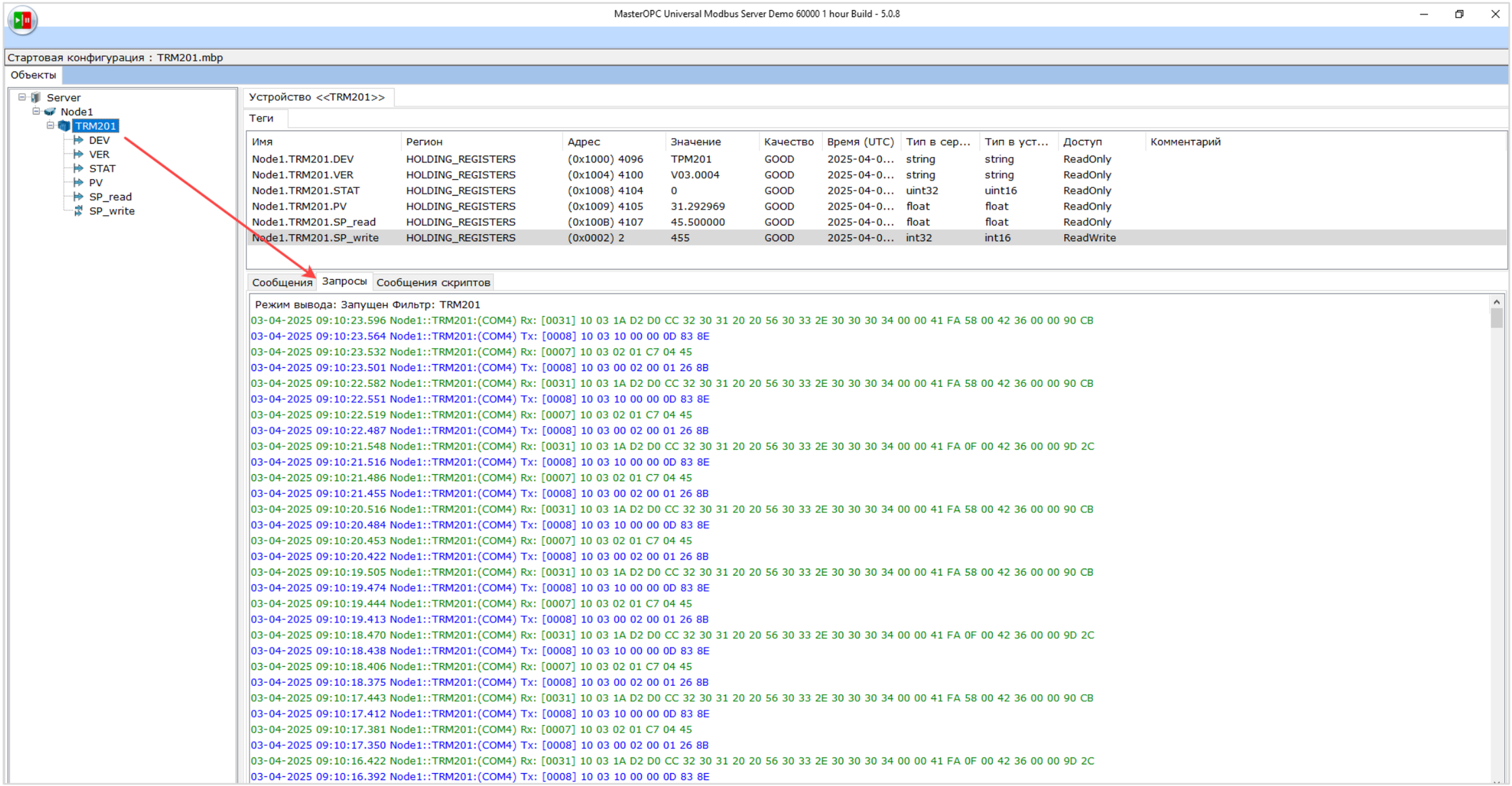


Если всё настроено и подключено корректно, то у всех тегов будет отображаться статус **Good**.

Выделите в конфигурации узел TRM201 и в центре экрана перейдите на вкладку **Запросы**, чтобы увидеть лог обмена между OPC–сервером и прибором. Этот лог мы будем анализировать в течение следующего урока.

Обратите внимание, как соотносятся между собой значения тегов **SP_read** и **SP_Write**:

- значение **SP_Write** в 10 раз больше значения **SP_Read**. Это связано с тем, что упомянутый ранее параметр **dP** (положение десятичной точки) у нашего TPM201 имеет значение **1**.



В логе синим цветом выделены запросы OPC–сервера, а зелёным – ответы от ТРМ.

Лог нужно читать снизу вверх – то есть самый первый запрос будет находиться в самом низу лога.

Обратите внимание, что OPC–сервер поочерёдно отправляет два запроса:

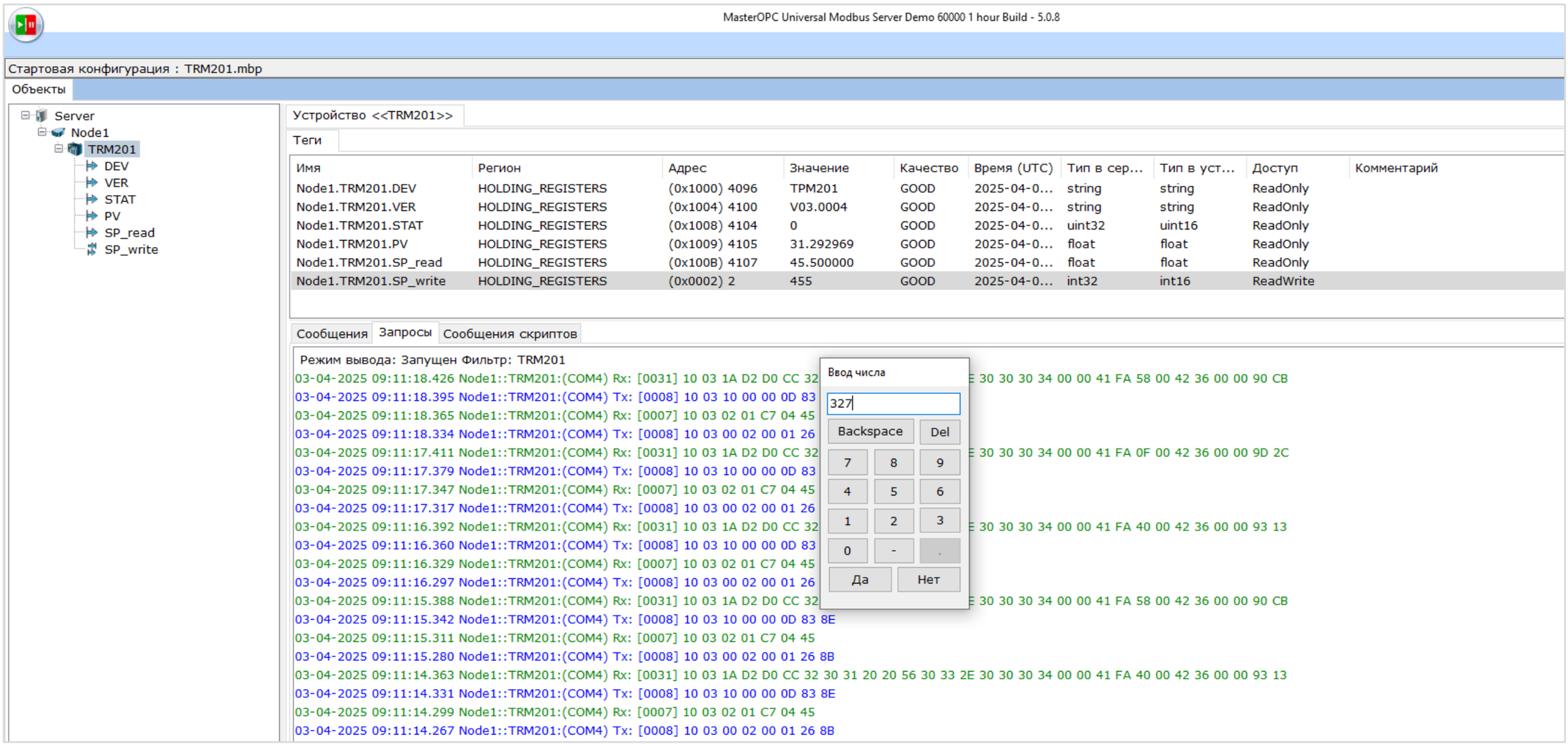
> *Запрос: 10 03 10 00 00 0D 83 8E*
>
> *Ответ: ... (длинный, каждый раз – разный)*

> *Запрос: 10 03 00 02 00 01 26 8B*
>
> *Ответ: ... (короткий, повторяющийся)*

Первый запрос – это запрос на чтение первых 5 тегов. OPC–сервер «понял», что адреса их регистров расположены последовательно, и объединил их чтение в один групповой запрос. Ответ на этот запрос каждый раз разный, потому что в состав этих тегов входит измеренное значение температуры, в котором знаки после запятой «плавают» из–за эффекта флуктуации.

Второй запрос – это запрос чтения значения уставки в формате **Int16**. Строго говоря, чтение этого значения нам не очень интересно, потому что мы уже получили значение уставки в формате с плавающей точкой (**SP_read**).

Но этот тег позволяет нам осуществить запись значения уставки. Для этого нажмите левой кнопкой мыши на текущее значение тега (на скриншоте выше оно равно **455**) и введите новое значение – например, **327**; как вы уже знаете, оно будет соответствовать уставке **32.7**. Для окончания ввода в диалоге нажмите клавишу **Да**.



Вы сразу увидите, что значения тегов **SP_write** и **SP_read** изменились – при очередных запросах чтения произошло получение только что введённого вами нового значения уставки.

Вы можете заметить, что тег **SP_read** имеет значение 32.700**195**, а не 32.7.

Эти последние цифры связаны с его типом – Float32 – использованным в ТРМ и OPC–сервере для данного параметра. Их наличие связано с погрешностями округления, которыми характеризуется этот тип данных.

Устройство <<TRM201>>

Теги

Имя	Регион	Адрес	Значение	Качество	Время (UTC)	Тип в сер...	Тип в уст...	Доступ	Комментарий
Node1.TRM201.DEV	HOLDING_REGISTERS	(0x1000) 4096	TPM201	GOOD	2025-04-0...	string	string	ReadOnly	
Node1.TRM201.VER	HOLDING_REGISTERS	(0x1004) 4100	V03.0004	GOOD	2025-04-0...	string	string	ReadOnly	
Node1.TRM201.STAT	HOLDING_REGISTERS	(0x1008) 4104	0	GOOD	2025-04-0...	uint32	uint16	ReadOnly	
Node1.TRM201.PV	HOLDING_REGISTERS	(0x1009) 4105	31.257324	GOOD	2025-04-0...	float	float	ReadOnly	
Node1.TRM201.SP_read	HOLDING_REGISTERS	(0x100B) 4107	32.700195	GOOD	2025-04-0...	float	float	ReadOnly	
Node1.TRM201.SP_write	HOLDING_REGISTERS	(0x0002) 2	327	GOOD	2025-04-0...	int32	int16	ReadWrite	

В момент ввода нового значения уставки в логе появятся следующие строки:

```
03-04-2025 09:14:10.325 Node1::TRM201:(COM4) Rx: [0008] 10 10 00 02 00 01 A3 48
03-04-2025 09:14:10.280 Node1::TRM201:(COM4) Tx: [0011] 10 10 00 02 00 01 02 01 47 26 40
```

Это запрос записи с кодом функции **0x10** (**Write Multiple Registers**), который OPC–сервер отправил прибору и ответ, который прибор отправил OPC–серверу.

Для работы с логом удобно использовать две команды контекстного меню, которое появляется при нажатии правой кнопкой мыши на любую область лога:

- команда **Приостановить** прекращает вывод в лог, позволяя спокойно его прокручивать и анализировать;
- команда **Копировать в буфер обмена** позволяет сохранить лог в буфер обмена операционный системы, чтобы затем вставить его в документ, e–mail и т. д.

```
Режим вывода: Запущен Фильтр: TRM201
03-04-2025 09:14:24.189 Node1::TRM201:(COM4) Rx: [0031] 10 03 1A D2 D0 CC 32 30 31 20 20 56 30 33 2E 30 30 30 34 00 00 41 FA 40 00 42 02 CD 00 86 4D
03-04-2025 09:14:24.157 Node1::TRM201:(COM4) Tx: [0008] 10 03 10 00 00 0D 83 8E
03-04-2025 09:14:24.126 Node1::TRM201:(COM4) Rx: [0007] 10 03 02 01 47 05 E5
03-04-2025 09:14:24.095 Node1::TRM201:(COM4) Tx: [0008] 10 03 00 02 00 01 26 8B
03-04-2025 09:14:23.175 Node1::TRM201:(COM4) Rx: [0031] 10 03 1A D2 D0 CC 32 30 31 20 20 56 30 33                                  02 CD 00 97 E7
03-04-2025 09:14:23.144 Node1::TRM201:(COM4) Tx: [0008] 10 03 10 00 00 0D 83 8E
03-04-2025 09:14:23.113 Node1::TRM201:(COM4) Rx: [0007] 10 03 02 01 47 05 E5
03-04-2025 09:14:23.082 Node1::TRM201:(COM4) Tx: [0008] 10 03 00 02 00 01 26 8B
03-04-2025 09:14:22.158 Node1::TRM201:(COM4) Rx: [0031] 10 03 1A D2 D0 CC 32 30 31 20 20 56 30 33 2E 30 30 30 34 00 00 41 FA 40 00 42 02 CD 00 86 4D
03-04-2025 09:14:22.126 Node1::TRM201:(COM4) Tx: [0008] 10 03 10 00 00 0D 83 8E
03-04-2025 09:14:22.094 Node1::TRM201:(COM4) Rx: [0007] 10 03 02 01 47 05 E5
```

Приостановить
Очистить
Копировать в буфер обмена

Регистры статуса. Понятие битовой маски

Среди считанных нами параметров есть **STAT** – регистр статуса.

В ходе нашего опроса он всегда имел значение **0**, и мы о нём ещё не упоминали.

Самое время — это сделать.

Регистр статуса содержит информацию о событиях, которые могут возникнуть в ходе работы ТРМ.

Для их представления используется битовая маска – то есть каждый бит значения регистра соответствует определённому событию. Если этот бит имеет значение TRUE (1) – то в данный момент это событие активно, если FALSE (0) – то неактивно.

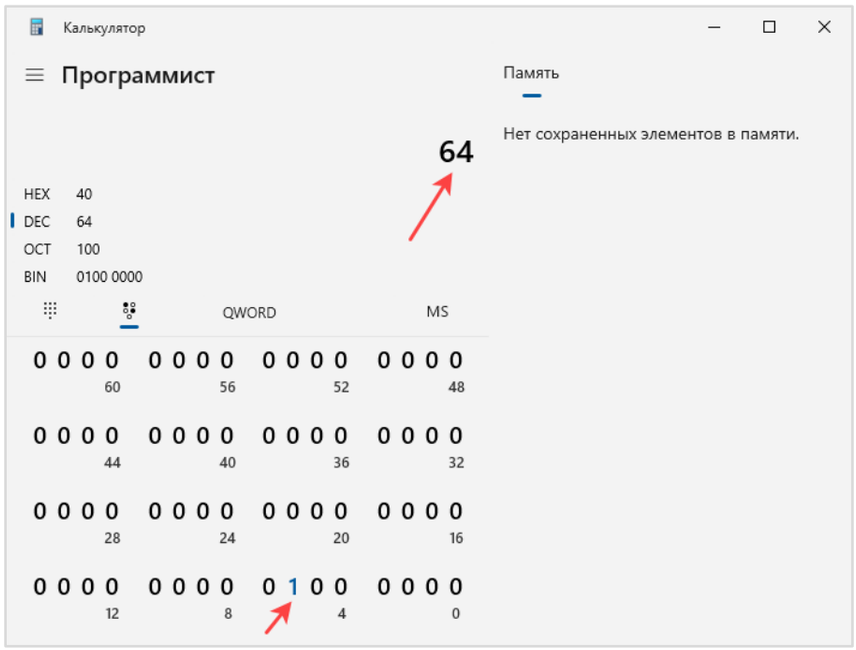
Список битов регистра статуса приведён в уже не раз использованном нами документе «Краткая инструкция ТРМ201 по работе с Modbus»:

Назначение битов регистра STAT

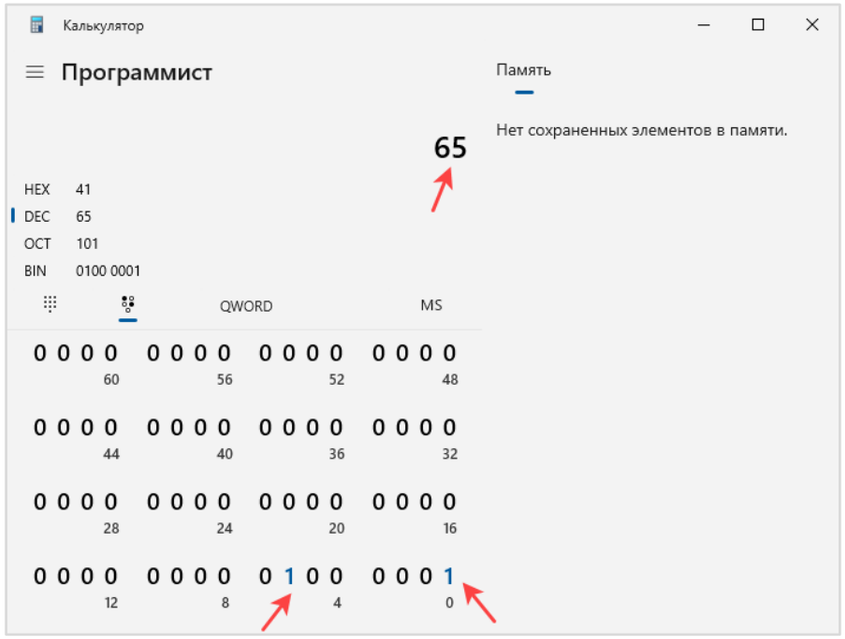
Номер бита	Описание
0	Ошибка на входе
3	Прочая ошибка, несовместимая с работой прибора (например, Er.Ad, Er.64)
4	Срабатывание реле
6	Дистанционное управление регулятором (**r-L**)
1, 2, 5, 7 - 15	В этом бите всегда 0

Для ТРМ201 в маске используются всего 4 бита; в других модификациях ТРМ2хх их может быть больше.

Предположим, регулятор находится в дистанционном режиме управления. В этом случае 6–й бит (в нумерации с 0) регистра статуса имеет значение TRUE (1), и, соответственно сам регистр статуса имеет значение 64 (в десятичной системе счисления):



Если одновременно с этим будет детектирована ошибка аналогового входа (например, из–за обрыва датчика), то нулевой бит будет установлен в значение TRUE (1) и, соответственно, регистр статуса примет значение 65:



Битовые маски широко используются при обмене по Modbus для компактного размещения группы битовых параметров.

Мы ещё не раз встретимся с ними по ходу курса.

Особенности и неудачи реализации Modbus в ТРМ2хх

Некоторые параметры ТРМ должны сохранять свои значения после выключения и повторного включения питания прибора – например, это касается настроек аналогового входа, уставки и т. д. Такие параметры называются **энергонезависимыми**.

В ТРМ2хх энергонезависимая память реализована в виде микросхемы EEPROM, для которой максимально допустимое количество цикло перезаписи довольно невелико и составляет примерно несколько тысяч раз.

В реализации Modbus в данной линейке приборов каждый запрос записи любого энергонезависимого параметра приводит к перезаписи EEPROM. В прошлом шаге это произошло, когда мы ввели новое значение уставки.

Если реализовать циклическую запись значения уставки и записывать её значение (даже если это значение каждый раз будет одним и тем же), например, раз в секунду – то в течение дня совершенно спокойно можно «перетереть» память прибора, и его настройки перестанут сохраняться после отключения и включения.

Это неудачная (и, что ещё хуже, не описанная в руководстве) реализация, которая была исправлена в следующих линейках ТРМ.

Существуют разные подходы к решению этой проблемы. Вот некоторые из них:

- использовать для хранения энергонезависимых параметров микросхему, которая поддерживает количество циклов перезаписи, которого с запасом должно хватить на предполагаемый срок службы прибора (MRAM, FRAM и т. д.);
- использовать встроенный аккумулятор, который позволит прибору проработать достаточное количество времени после пропадания питания, чтобы сохранить свои энергонезависимые переменные. Соответственно, в этом случае достаточно сохранять значения энергонезависимых параметров именно в момент пропадания питания;
- не сразу записывать полученное по Modbus значение в энергонезависимую память, а периодически или по команде пользователя (путём записи определённого значения в специально выделенный под это регистр).

***

С опросом ТРМ2хх по Modbus RTU связана ещё одна потенциально возможная и крайне неприятная ситуация.

Спустя какое–то время (которое может измеряться неделями и даже месяцами) интерфейс прибора может зависнуть, и при этом сделать неработоспособной всю линию связи, к которой он подключен; для устранения проблемы потребуется перезагрузить прибор по питанию.

Эта проблема не проявляется при использовании протокола Modbus ASCII.

В современных линейках ТРМ подобная проблема отсутствует – они стабильно работают как при использовании протокола Modbus RTU, так и Modbus ASCII.

Подведение итогов

Мы организовали опрос ТРМ201 с помощью MasterOPC Universal Modbus Server и получили лог обмена, увидев в нём, что из себя представляет групповой запрос. Этот лог мы используем в следующем уроке – на примере запросов и ответов из него рассмотрим структуру пакетов Modbus.

Также мы познакомились с битовыми масками и выяснили, какие неудачные решения могут быть допущены в процессе разработки прибора, поддерживающего протокол Modbus.

2.7 Тестовые задания

Ответы приведены в п. 9.

1. На какой архитектуре основан протокол Modbus?
   - Peer-to-peer
   - Master/Slave (Client/Server)
   - Publisher/Subscriber
   - Manager/Agent

2. Сколько master–устройств может быть подключено к последовательной линии связи?
   - 255
   - 247
   - 1
   - 2

3. Какие типы данных определены в спецификации Modbus?
   - Uint32
   - Int32
   - Бит
   - Float
   - String
   - Uint16
   - Int16
   - Регистр

4. Укажите размер регистра в битах.

5. Вставьте пропущенную фразу.
   "Спецификация Modbus определяет <....> при передаче данных".
   - Порядок регистров
   - Порядок байт и регистров
   - Типы данных
   - Порядок байт

6. Сопоставьте функцию Modbus с её категорией.

0x03 (Read Holding Registers)	Диагностика
0x14 (Read File Record)	Работа с битами
0x05 (Write Single Coil)	Работа с файлами
0x07 (Read Exception Status)	Работа с регистрами

7. Укажите максимальное количество объектов (битов или регистров), которое может быть размещено в одной области памяти slave–устройства.

2.8 Основы Modbus, часть 2

Введение

В следующих шагах мы обсудим форматы запросов и ответов протокола Modbus RTU.

В ходе обсуждения мы будем использовать лог обмена с терморегулятором ТРМ201, который получили в предыдущем уроке.

И запросы, и ответы состоят из набора параметров, которые мы будем называть **полями**.

Во многих случаях для представления значений полей мы будем использовать шестнадцатеричную систему счисления (HEX). Значения, записанные в этой системе, будут сопровождаться префиксом **0x** или постфиксом **h**. Примеры:

- **0x10** – значение, записанное в HEX;
- **10h** – это же значение, записанное в HEX, но с другим принципом обозначения;
- **16** – это же значение, но записанное в десятичной системе счисления.

Когда мы будем рассказывать про какие–то универсальные характеристики, касающиеся и запросов, и ответов, то будем объединять их (запросы и ответы) с помощью термина «**пакет**» (в спецификации Modbus используется термин «**frame**», часто заменяемый в русскоязычной документации на «фрейм»).

Первое, что стоит упомянуть – согласно спецификации [1, п. 2.5.1] максимальный размер пакета Modbus RTU составляет **256 байт**. На этом же рисунке видна общая структура пакета Modbus RTU, которую мы подробнее рассмотрим в следующих шагах.

MODBUS over serial line specification and implementation guide V1.02

Frame description :

Slave Address	Function Code	Data	CRC
1 byte	1 byte	0 up to 252 byte(s)	2 bytes CRC Low, CRC Hi

Figure 12: RTU Message Frame

➔ The maximum size of a MODBUS RTU frame is 256 bytes.

Формат запроса протокола Modbus RTU – функция 0x03 (Read Holding Registers)

Скопируем запрос с функцией **0x03** из лога обмена с ТРМ201 и рассмотрим каждое из его полей.

Обратите внимание, что в логах традиционно используется шестнадцатеричная система счисления (HEX). При расшифровке полей в скобках мы будем указывать их значения в десятичной системе, а также длину в байтах. В тексте описаний также используется десятичная система.

> *10 03 10 00 00 0D 83 8E*

- 10h (16) – адрес slave–устройства (1 байт);
- 03h (3) – код функции (1 байт);
- 1000h (4096) – адрес начального считываемого регистра (2 байта);
- 000Dh (13) – количество считываемых регистров (2 байта);
- 838Eh – контрольная сумма (2 байта).

Обсудим каждое из полей:

- **Адрес slave–устройства**: согласно спецификации Modbus – может принимать значение из диапазона **1...247.** Адрес **0** зарезервирован под широковещательный запрос (broadcast), который мы обсудим в уроке 2.12;
- **Код функции**: код **03h** соответствует функции **Read Holding Registers**. Таблица с кодами функций приведена здесь;
- **Адрес начального считываемого регистра** и их **количество**: мы используем групповой запрос для считывания всех 5 оперативных параметров ТРМ201, которые суммарно занимают 13 регистров:

Группа LvoP. Оперативные параметры (только чтение: Modbus-функция 0x03)					
DEV	Тип прибора	0x 1000 0x 1001 0x 1002 0x 1003	Char[8]	–	TPM201
VER	Версия прибора	0x 1004 0x 1005 0x 1006 0x 1007	Char[8]	–	V03.00xx
STAT	Регистр статуса	0x 1008	binary	–	16 бит [1)]
PV	Измеренная величина	0x 1009 0x 100A	Float32	–	
SP	Уставка регулятора	0x100B 0x100C	Float32	–	

Возможно, вы заметили, что поле **Количество считываемых регистров** занимает 2 байта. Это может привести к мысли, что максимальное количество регистров, которое можно считать запросом – **65535** (таково максимальное значение, которое можно уместить в 2 байта). Но это не так; как мы упоминали в предыдущем шаге – размер пакета Modbus RTU не может быть больше **256 байт**.

С учётом того, что несколько байт используется на служебные поля (адрес, код функции, контрольная сумма и т. д.) – **максимальное количество регистров, которое можно считать одним запросом, составляет 125.**

Но возникает другой вопрос – зачем тогда под поле **Количество считываемых регистров** выделено 2 байта, ведь значение **125** спокойно помещается в один?

Это связано с «битовыми» функциями **0x01 (Read Coils)** и **0x02 (Read Discrete Inputs)**. Вы помните, что регистр занимает 16 бит – и, соответственно, максимальное количество бит, которое можно считать одним запросом, составляет **125·16 = 2000**. Для хранения этого значения одного байта недостаточно – поэтому выделено 2.

***

Контрольная сумма используется для контроля целостности пакета. При получении пакета устройство (как master, так и slave) выделяет из него контрольную сумму, а потом рассчитывает её для всего полученного пакета (с*амо поле контрольной суммы при расчёте не используется*) и сравнивает эти два значения. Если они не совпадают – значит в процессе передачи один или несколько байт пакета были повреждены; такой пакет считает некорректным и не обрабатывается.

В протоколе Modbus RTU для расчёта контрольной суммы используется алгоритм CRC–16–IBM с полиномом $x^{16} + x^{15} + x^2 + 1$Теоретическую информацию о CRC можно получить в статье на Википедии.

Важно отметить, что поле контрольной суммы передаётся в протоколе Modbus RTU младшим байтом вперёд [1, п. 6.2.2] – в отличие от всех остальных полей, которые должны передаваться старшим байтом вперёд.

Так как под поле контрольной суммы выделено 2 байта – то существует всего 65536 её вариантов, что существенно меньше возможных вариантов пакетов Modbus RTU. В связи с этим могут существовать (и существуют) разные пакеты, для которых будет вычислена одна и та же контрольная сумма

> *Некоторые другие аспекты надёжности контрольной суммы Modbus рассмотрены в п. 2.3 статьи Заметки о Modbus.*

Формат запроса протокола Modbus RTU – функция 0x04 (Read Input Registers)

...совершенно аналогичен тому, который мы рассмотрели в предыдущем шаге.

Единственное отличие – в поле **Код функции** будет использоватья значение **04h**, а не **03h**.

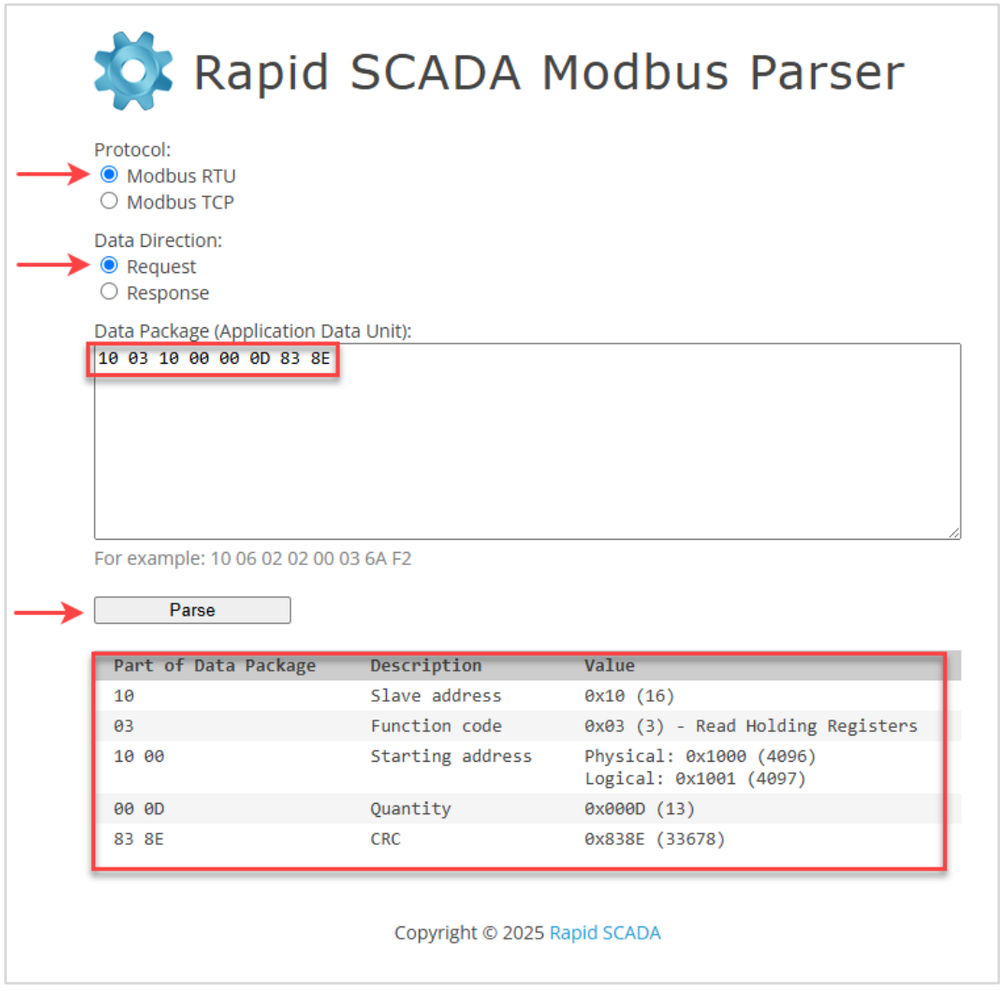
Онлайн–парсер пакетов Modbus

Для отладки обмена по Modbus очень удобно использовать онлайн–парсер пакетов Modbus, созданный разработчиками SCADA–системы RapidSCADA. Передаём огромную благодарность его разработчикам!

https://rapidscada.net/modbus/

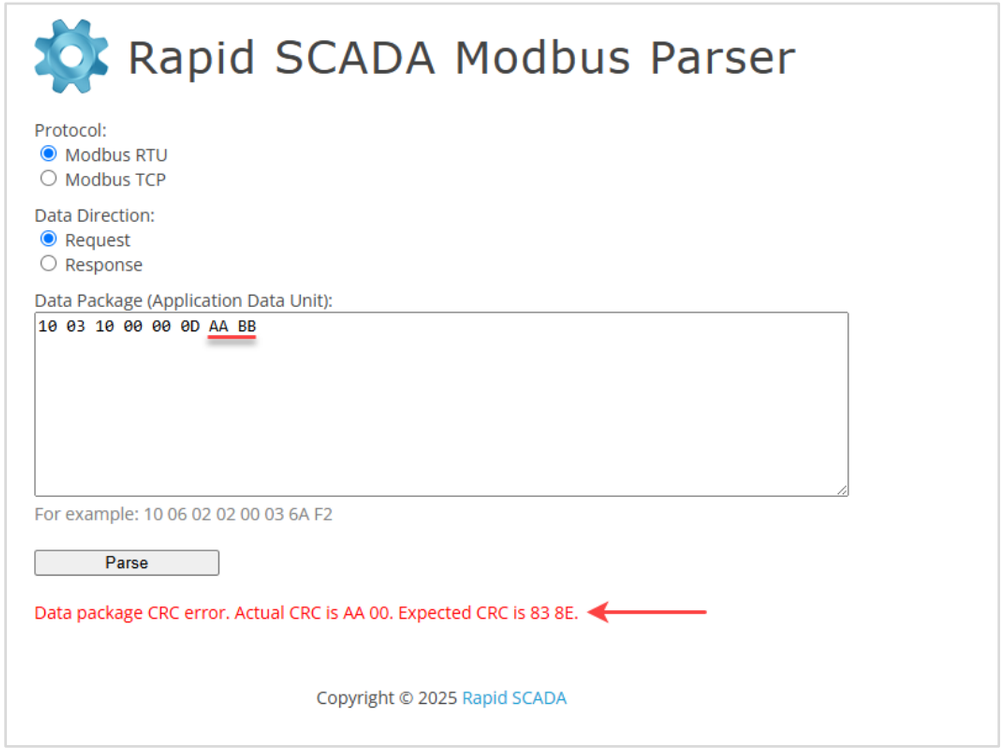
Достаточно ввести в него содержимое пакета (в HEX), выбрать протокол (Modbus RTU или Modbus TCP) и тип пакета (**request** – запрос от мастера к slave–устройству, **response** – ответ от slave–устройства мастеру) и нажать кнопку **Parse**.

Давайте сделаем это для запроса, который мы недавно рассмотрели:



> *Для поля **Starting address** (адрес начального считываемого регистра) отображается два значения с пометками «**Physical**» и «**Logical**». Они связаны с различными системами адресации. Мы поговорим об этом в уроке 7.2. Пока что, пожалуйста, не обращайте внимание на «Logical»–значения.*

С использованием парсера можно «конструировать» интересующие вас Modbus–пакеты (*вы сделаете это в одном из практических заданий в уроке 2.10*). Единственный вопрос, который может возникнуть – как рассчитать контрольную сумму для пакетов Modbus RTU? Проще всего попросить парсер рассчитать её за вас – для этого достаточно ввести вместо контрольной суммы два любые байта (например, AA BB). Парсер отобразит сообщение об ошибке контрольной суммы и укажет корректную контрольную сумму для данного пакета:



Формат ответа протокола Modbus RTU – функция 0x03 (Read Holding Registers)

Теперь рассмотрим ответ, который ТРМ201 отправил нашему OPC–серверу на запрос, изученный нами несколько шагов назад.

> *10 03 1A **D2 D0 CC 32 30 31 20 20 56 30 33 2E 30 30 30 34 00 00 41 FA 28 00 42 36 00 00** 9B FB*

Его структура крайне близка к структуре запроса:

- 10h (16) – адрес slave–устройства (1 байт);
- 03h – код функции или признак ошибки (1 байт). Возможные ошибки обсудим в следующем шаге;
- 1Ah (26) – количество байт данных в ответе (1 байт);
- **данные ответа** (от 2 до 250 байт в зависимости от количества считываемых в запросе регистров);
- 9BFBh – контрольная сумма (2 байта).

Обратите внимание, что в запросе указывается количество регистров, которые мастер хочет считать из slave–устройства, а в ответе slave–устройство указывает количество байт данных, которые входят в состав ответа – это значение должно быть в два раза больше количества запрошенных регистров, так регистр занимает 2 байта.

Байты данных нашего ответа выделены жирным шрифтом.

Чтобы правильно интерпретировать значения параметров, которые в них закодированы, нам нужно знать список этих параметров и их типы данных. Они приведены в документе «Краткая инструкция ТРМ201 по работе с Modbus», который мы рассматривали в предыдущем уроке.

Группа LvoP. Оперативные параметры (только чтение: Modbus-функция 0x03)

DEV	Тип прибора	0x 1000 0x 1001 0x 1002 0x 1003	Char[8]	–	ТРМ201
VER	Версия прибора	0x 1004 0x 1005 0x 1006 0x 1007	Char[8]	–	V03.00xx
STAT	Регистр статуса	0x 1008	binary	–	16 бит [1)]
PV	Измеренная величина	0x 1009 0x 100A	Float32	–	
SP	Уставка регулятора	0x100B 0x100C	Float32	–	

Типы данных

Тип данных	Описание
Int16	Двухбайтовое целое. На каждый параметр типа Integer отводится один регистр Modbus. Для параметров, значения которых могут иметь отрицательное значение (Signed Int16), отрицательные числа представляются в дополнительном коде. Передача данных осуществляется в формате $X*10^{-n}$, где X – передаваемое целое число, n – непередаваемая степень 10 (для каждого параметра она указывается в столбце «Кол-во знаков после запятой»).
float32	Четырехбайтовое с плавающей точкой. На каждый параметр типа Float отводится 2 соседних регистра Modbus. В регистре с младшим номером хранится старшая часть числа (high word), в регистре с большим номером – младшая часть числа (low word). Передача числа осуществляется по принципу «старшим вперед» (high byte first – high word first).
Char[8]	Строка из 8 символов. На каждый параметр типа String отводится 4 соседних регистра Modbus. В регистре с младшим номером хранятся первые два символа строки, в регистре с большим номером – последние. Для данных типа String, в отличие от протокола ОВЕН, используется прямой порядок следования символов (первым передается первый символ из строки).
Hex word	Двухбайтовое число в шестнадцатеричном формате
Binary	Двухбайтовое число в двоичном формате. При передаче первым следует пятнадцатый бит, последним – нулевой.

Давайте «расшифруем» данные ответа самостоятельно, воспользовавшись онлайн–конвертерами.

> *D2 D0 CC 32 30 31 20 20*

Это строка из 8 символов в кодировке Win1251, содержащая значение «ТРМ201 » (именно так, с двумя пробелами в конце).

> *56 30 33 2E 30 30 30 34*

Тоже строка. Её значение – "V03.0004".

> *00 00*

Значение регистра статуса. Мы говорили о нём в прошлом уроке.

> *41 FA 28 00*

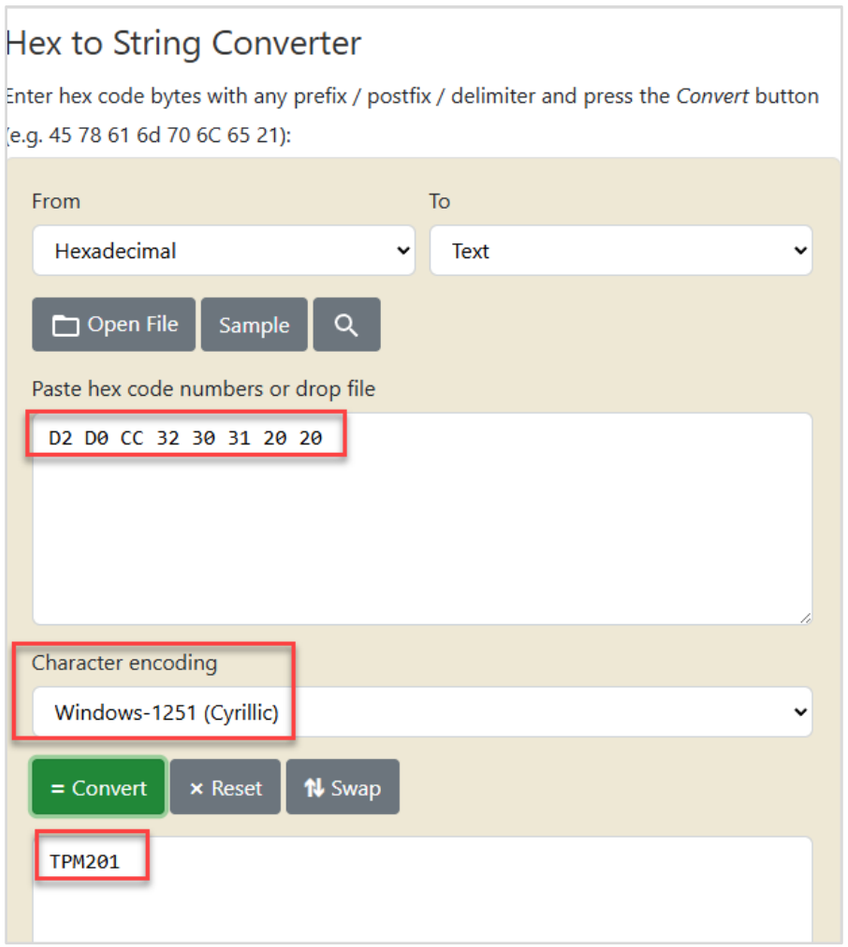
Измеренное значение на аналоговом входе – «31.269531».

> *42 36 00 00*

Заданное значение уставки – «45.5».

Ниже приведены скриншоты из онлайн–конвертеров, которые подтверждают нашу расшифровку.

https://www.rapidtables.com/convert/number/hex–to–ascii.html



From

Hexadecimal

To

Text

Open File | Sample

Paste hex code numbers or drop file

```
56 30 33 2E 30 30 30 34
```

Character encoding

Windows-1251 (Cyrillic)

= Convert | × Reset | ⇅ Swap

```
V03.0004
```

https://www.h–schmidt.net/FloatConverter/IEEE754.html

Quick links

- IEEE754
- FileHasher
- World sat image

Latest article

- Zone-based firewalling on Mikrotik routers

Contact

IEEE-754 Floating Point Converter

Translations: de

This page allows you to convert between the decimal representation of a number (like "1.02") and the binary format used by all modern CPUs (a.k.a. "IEEE 754 floating point").

IEEE 754 Converter, 2024-02

	Sign	Exponent	Mantissa
Value:	+1	2^4	1 + 0.954345703125
Encoded as:	0	131	8005632
Binary:	☐	☑☐☐☐☐☐☑☑	☑☑☑☑☐☑☐☐☐☑☐☑☐☐☐☐☐☐☐☐☐☐☐☐

Decimal Representation	31.269531
Value actually stored in float:	31.26953125
Error due to conversion:	0.00000025
Binary Representation	01000001111110100010100000000000
Hexadecimal Representation	41fa2800

1 -1

www.h-schmidt.net: Home | Articles | Tools | Archive | No Cookie Policy

Quick links

- IEEE754
- FileHasher
- World sat image

Latest article

- Zone-based firewalling on Mikrotik routers

Contact

IEEE-754 Floating Point Converter

Translations: de

This page allows you to convert between the decimal representation of a number (like "1.02") and the binary format used by all modern CPUs (a.k.a. "IEEE 754 floating point").

IEEE 754 Converter, 2024-02

	Sign	Exponent	Mantissa
Value:	+1	2^5	1 + 0.421875
Encoded as:	0	132	3538944
Binary:	☐	☑☐☐☐☐☑☐☐	☐☑☑☐☑☑☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Decimal Representation	45.5
Value actually stored in float:	45.5
Error due to conversion:	0
Binary Representation	01000010001101100000000000000000
Hexadecimal Representation	42360000

1 -1

Коды ошибок пакетов Modbus (часть 1)

Когда slave–устройство получает запрос от master–устройства – то выполняет его анализ. В рамках анализа проверяются следующие поля пакета:

- совпадает ли адрес slave–устройства, указанный в запросе, с адресом данного slave–устройства?
- если да – анализ запроса продолжается;
- если нет – то анализ запроса прекращается. На такой запрос slave–устройство не отправит ответ, потому что он предназначен другому устройству;
- совпадает ли выделенная из пакета контрольная сумма (CRC) с контрольной суммой, рассчитанной на основании байт пакета? Если да – анализ запроса продолжается. Если нет – то запрос считается повреждённым; его анализ прекращается. На такой запрос slave–устройство не отправит ответ;
- имеют ли все остальные поля запроса логически корректные значения?
- если да – то slave–устройство выполняет операцию, определяемую кодом функции запроса (подготавливает запрошенные мастером данные или применяет присланные мастером данные), после чего отправляет ответ master–устройству – мы видели пример такого пакета на предыдущем шаге;
- если нет – slave–устройство не выполняет операцию и отправляет master–устройству ответ с кодом ошибки, характеризующим детектированную им ошибку.

Предположим, мы отправили запрос чтения (HEX):

> *10 03 10 00 00 0D 83 8E*

не на ТРМ201, а на регулятор совсем другого производителя, у которого нет регистров **0x1000** и далее (но в настройках этого регулятора также задан адрес slave–устройства = **16**).

В ответ мы должны получить следующий пакет:

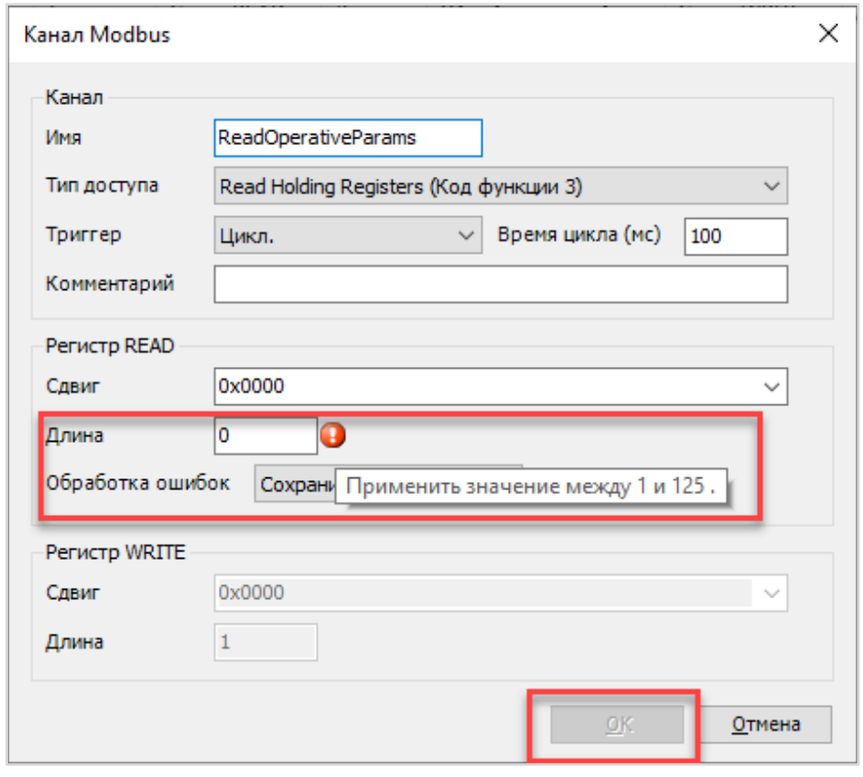
> *10 83 02 90 F4*

- 10h (16) – адрес slave–устройства (1 байт);
- 83h – код функции из запроса, к которому прибавлено значение 0x80 (или, иными словами, код функции из запроса с установленным старшим битом). Это является указанием на то, что данный пакет содержит код ошибки;
- 02h (2) – код ошибки (1 байт). Таблица с возможными кодами ошибок приведена ниже. В нашем случае мы видим код 02h – ILLEGAL DATA ADRESS – который свидетельствует о том, что в slave–устройстве отсутствует как минимум один из регистров, значение которого мы попытались прочитать в рамках нашего запроса;
- 90F4h – контрольная сумма (2 байта).

Категория	Код	Название	Описание
Стандартные	0x01	ILLEGAL FUNCTION	Функция, указанная в запросе, не поддерживается slave-устройством
	0x02	ILLEGAL DATA ADDRESS	Адрес объекта и/или число объектов, указанные в запросе, не является корректными для конкретного slave-устройства
	0x03	ILLEGAL DATA VALUE	**Функции 0x01, 0x02, 0x03, 0x04**: число объектов, указанное в запросе, выходит за ограничения спецификации Modbus **Функции 0x05, 0x06**: некорректное (с точки зрения спецификации) записываемое значение **Функции 0x08, 0x14, 0x16, 0x18**: специфические ситуации
Специфические (без примеров в спецификации)	0x04	SERVER DEVICE FAILURE	Непреодолимая (*unrecoverable*) ошибка в процессе выполнения операции – например, slave-устройству не удалось произвести обращение к своей памяти
	0x05	ACKNOWLEDGE	Операция начала выполняться, но это займет много времени
	0x06	SERVER DEVICE BUSY	Операция не может быть запущена из-за выполнения другой операции
	0x08	MEMORY PARITY ERROR	**Функции 0x14, 0x15**: размер запрошенным данных в байтах не кратен 2
Ошибки шлюзов Modbus TCP/ Modbus RTU	0x0A	GATEWAY PATH UNAVAILABLE	Не удалось построить маршрут до slave-устройства за шлюзом
	0x0B	GATEWAY TARGET DEVICE FAILED TO RESPOND	Устройство за шлюзом не ответило на запрос

Наиболее часто встречающимися являются три первые ошибки:

- **0x01 (ILLEGAL FUNCTION)** – код функции из запроса не поддерживается slave–устройством. Мы могли получить такой код ошибки от ТРМ201, если бы, например, отправили ему запрос чтения с кодом функции **0x04** или запрос записи с кодом функции **0x06**;
- **0x02 (ILLEGAL DATA ADDRESS)** – указанный в запросе адрес начального объекта (бита или регистра) и/или их количество является некорректным для данного slave–устройства. Именно эту ошибку мы получили в рассмотренной выше ситуации;
- **0x03 (ILLEGAL DATA VALUE)** – в большинстве случаев эта ошибка возвращается в ответ на запрос с функцией чтения, для которой указано некорректное (с точки зрения спецификации Modbus) число считываемых объектов. Вы можете её увидеть, если, например, каким–то образом присвоите в запросе полю **Количество считываемых регистров** значение **0** (что, очевидно, бессмысленно) или значение, превышающее **125** (что нарушает спецификацию Modbus). Большинство Master–устройств просто не дадут вам ввести такое значение – см., например, соответствующее предупреждающее сообщение в среде CODESYS V3.5:


Канал Modbus
Канал
Имя
ReadOperativeParams
Тип доступа
Read Holding Registers (Код функции 3)
Триггер
Цикл.
Время цикла (мс)
100
Комментарий
Регистр READ
Сдвиг
0x0000
Длина
0
Обработка ошибок
Сохрани
Применить значение между 1 и 125 .
Регистр WRITE
Сдвиг
0x0000
Длина
1
OK
Отмена

Коды ошибок пакетов Modbus (часть 2)

В некоторых устройствах код ошибки с кодом **0x03 (ILLEGAL DATA VALUE)** отправляется в тех случаях, когда master–устройство пытается записать в параметр «некорректное» с точки зрения прибора значение (например, если мощность регулятора должна принадлежать диапазону 0...100%, а мастер пробует записать в соответствующий параметр значение 200).

Это является отступлением от спецификации Modbus...

03	ILLEGAL DATA VALUE	A value contained in the query data field is not an allowable value for server. This indicates a fault in the structure of the remainder of a complex request, such as that the implied length is incorrect. It specifically does NOT mean that a data item submitted for storage in a register has a value outside the expectation of the application program, since the MODBUS protocol is unaware of the significance of any particular value of any particular register.

...но достаточно «общепринятым» и, в целом, разумным отступлением.

Ошибки с кодами 0x04, 0x05, 0x06 и 0x08 встречаются крайне редко.

В ТРМ201 ошибка **0x04 (SLAVE DEVICE FAILURE)** характеризует, согласно руководству по эксплуатации, «невозможность выполнения полученной команды», но в чём именно заключается эта «невозможность» – руководство не поясняет, равно как и не приводит примеры подобных ситуаций.

Ошибки с кодами **0x0A** и **0x0B** могут быть возвращены исключительно шлюзами Modbus TCP/Modbus Serial. Мы обсудим их в уроке 5.3.

> *Возможно, вы заметили, что в списке пропущены ошибки с кодами **0x07** и **0x09**.*
>
> *Ошибка **0x09** отсутствует в принципе, а вот ошибка **0x07** описывалась в старой версии спецификации, но была исключена в «современной». Она означает **NEGATIVE ACKNOWLEDGE**, и в древние времена могла быть отправлена в ответ на запрос с кодом функции **0x0D (Program Controller)** и **0x0E (Poll Controller)**, которые поддерживались только контроллерами Modicon (эти функции также теперь исключены из спецификации протокола).*

Формат запроса протокола Modbus RTU – функция 0x10 (Write Multiple Registers)

Скопируем запрос с функцией **0x10** из лога обмена с ТРМ201 и рассмотрим каждое из его полей.

Мы использовали эту функцию для записи в ТРМ значения уставки.

> *10 10 00 02 00 01 02 01 2C 67 AF*

- 10h (16) – адрес slave–устройства (1 байт);
- 10h (16) – код функции (1 байт);
- 0002h (2) – адрес начального записываемого регистра (2 байта);
- 0001h (1) – количество записываемых регистров, от 1 до **123** (2 байта);
- 02h (2) – количество записываемых байт данных (1 байт);
- 012Ch – записываемые данные (от 2 до 246 байт в зависимости от количества записываемых в запросе регистров);
- 67AFh – контрольная сумма (2 байта).

Максимальное количество регистров в запросе записи с кодом функции 0x10 – **123**.

Значения поле данных (012Ch) в десятичной системе счисления соответствует значению **300**, которое мы записывали в ТРМ201 в качестве уставки.

Интересно, что в запросе размер передаваемых данных указывается как в виде количества регистров (0001h), так и в виде количества байт (02h). Цель такого дублирования в спецификации не раскрывается, но оно позволяет облегчить определение типа пакета (запрос или ответ) при его парсинге. Мы обсудим это в одном из следующих шагов.

Rapid SCADA Modbus Parser

Protocol:
- (•) Modbus RTU
- ( ) Modbus TCP

Data Direction:
- (•) Request
- ( ) Response

Data Package (Application Data Unit):

```
10 10 00 02 00 01 02 01 2C 67 AF
```

For example: 10 06 02 02 00 03 6A F2

Parse

Part of Data Package	Description	Value
10	Slave address	0x10 (16)
10	Function code	0x10 (16) - Write Multiple Registers
00 02	Starting address	Physical: 0x0002 (2) Logical: 0x0003 (3)
00 01	Quantity	0x0001 (1)
02	Byte count	0x02 (2)
01 2C	Registers value	0x012C (300)
67 AF	CRC	0x67AF (26543)

Copyright © 2025 Rapid SCADA

Формат ответа протокола Modbus RTU – функция 0x10 (Write Multiple Registers)

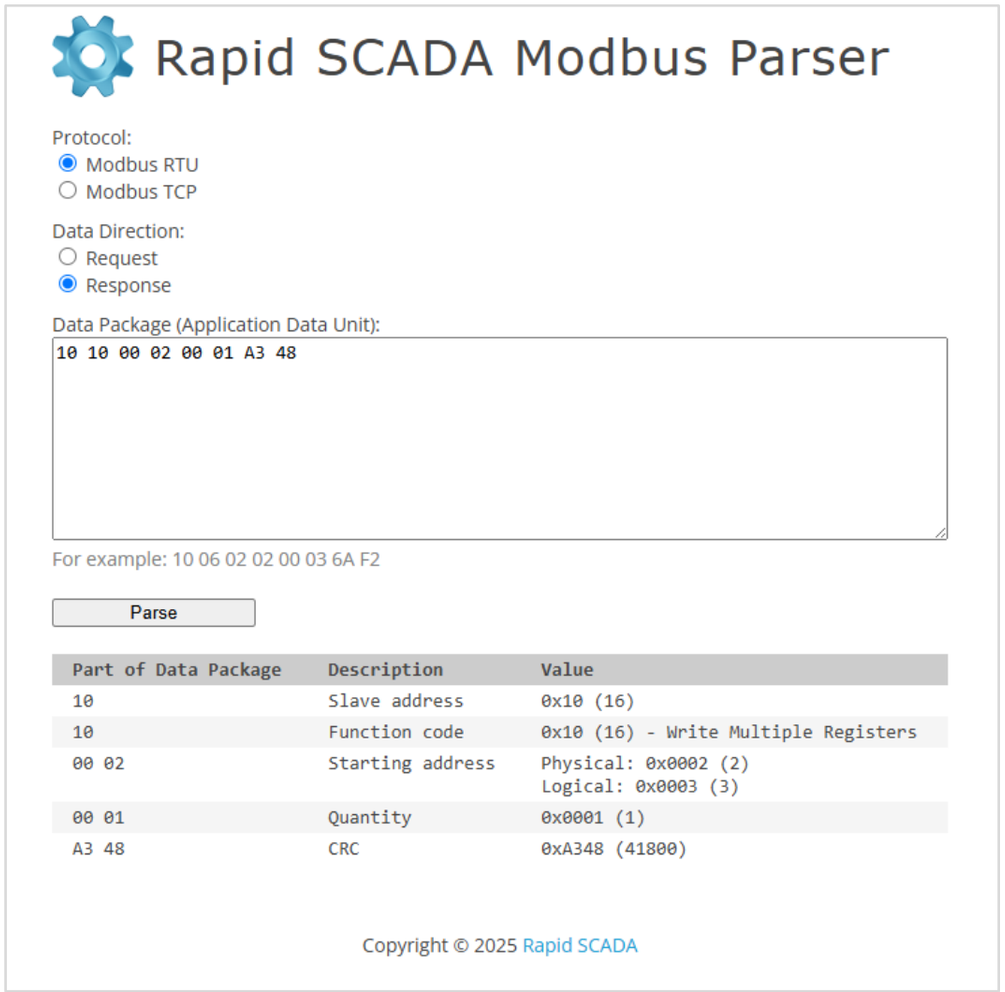
В ответ на запрос

> ***10 10 00 02 00 01*** *02 01 2C 67 AF*

TPM210 вернёт нам следующий ответ:

> ***10 10 00 02 00 01*** *A3 48*

То есть ответ представляет собой первые 6 байт запроса (адрес slave–устройства, код функции, адрес начального регистра, количество регистров), дополненные заново рассчитанной контрольной суммой.



Формат запроса и ответа протокола Modbus RTU – функция 0x06 (Write Single Register)

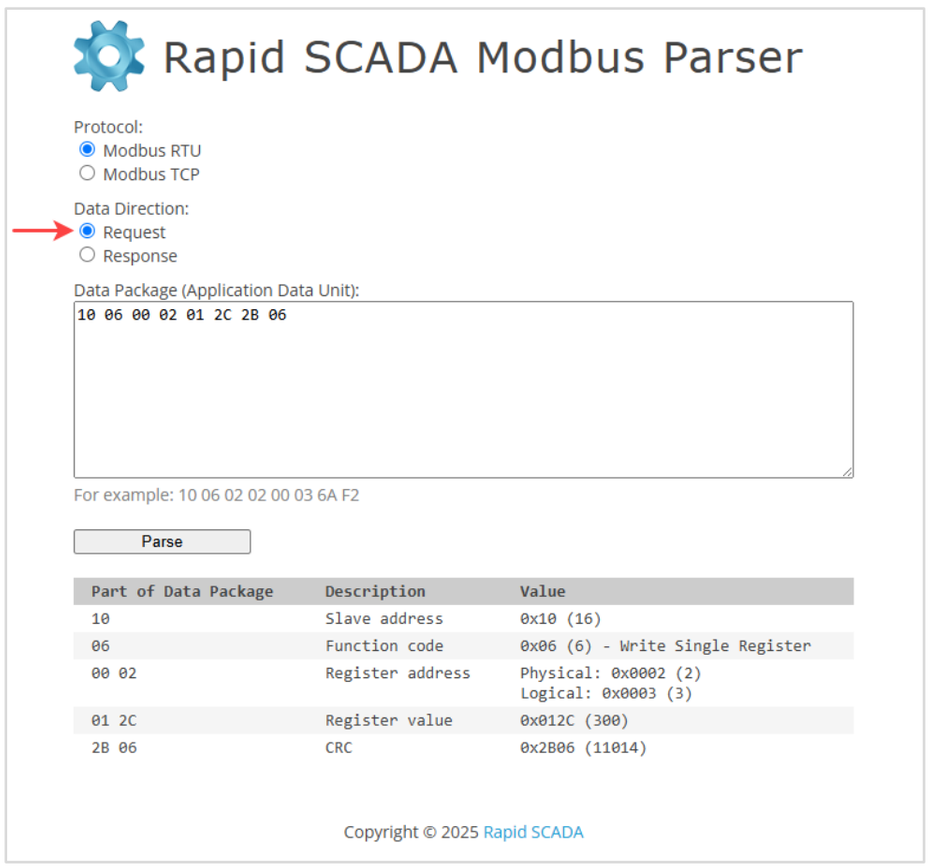
Функция **0x06** используется для записи одного регистра.

Целесообразность её наличия в спецификации может вызвать резонный вопрос – ведь один регистр можно записать и с помощью функции **0x10 (Write Multiple Registers)**; мы рассмотрели это в предыдущих шагах. Но, тем не менее, такая функция существует. Более того, вы можете встретить устройства, которые поддерживают её, но при этом не поддерживают функцию 0x10.

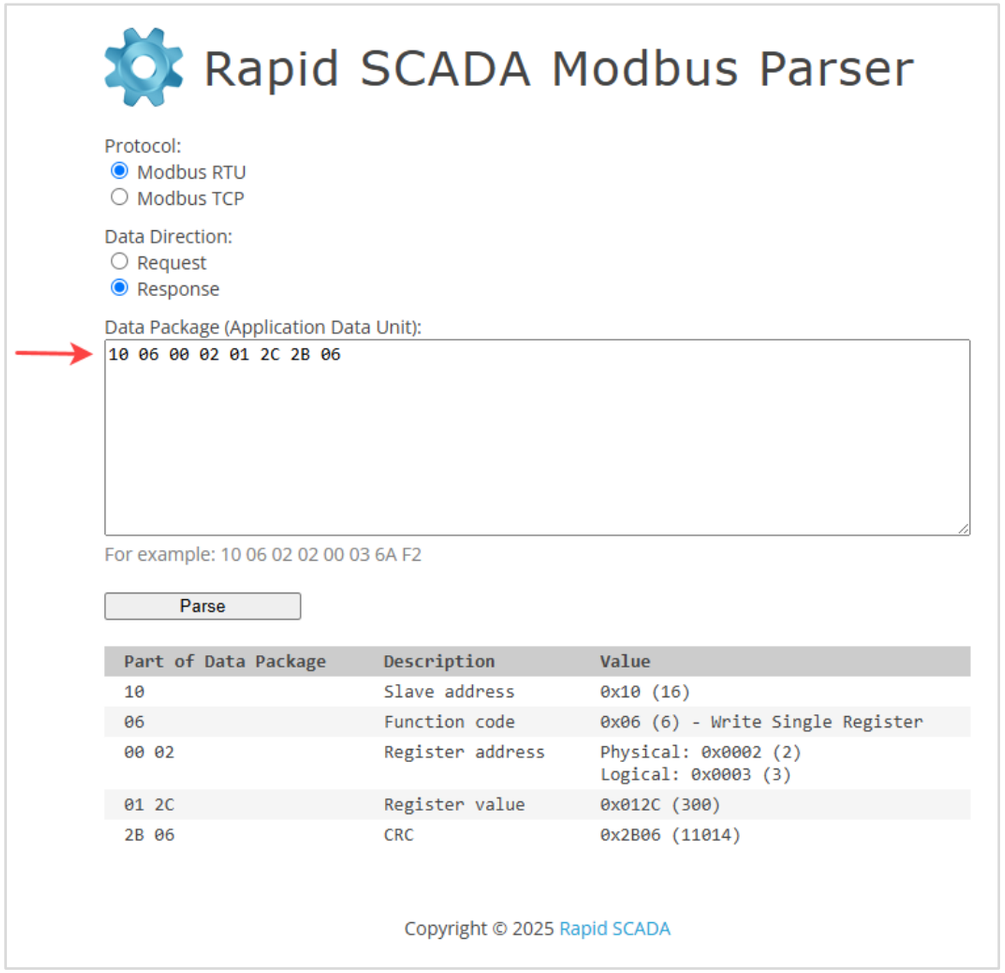
Примера запроса с этой функцией выглядит следующим образом:

> *10 06 00 02 01 2C 2B 06*

- 10h (16) – адрес slave–устройства (1 байт);
- 06h (6) – код функции (1 байт);
- 0002h (2) – адрес записываемого регистра (2 байта);
- 012Ch – значение записываемого регистра (2 байта);
- 2B06h – контрольная сумма (2 байта).

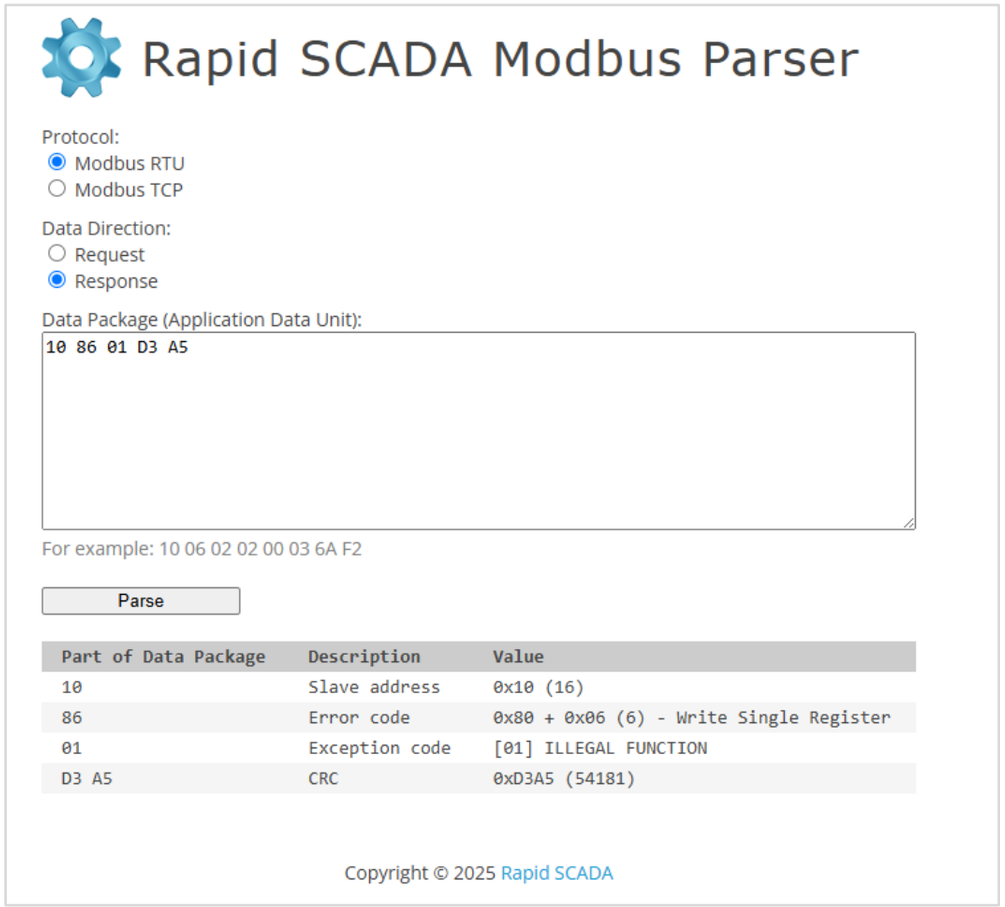


Что касается формата ответа – то в случае **успешной обработки** запроса с кодом функции **0x06** slave–устройство отправляет ответ, который **в точности** совпадает с этим запросом:



Если slave–устройство не может обработать запрос – то в ответ будет отправлен пакет с кодом ошибки. Например, TPM201, который не поддерживает функцию **0x06** в принципе, отправит в ответе код ошибки **0x01 (ILLEGAL FUNCTION)**:

> *10 86 **01** D3 A5*



Формат запроса и ответа протокола Modbus RTU – функции 0x01/0x02 (Read Coils/Read Discrete Inputs)

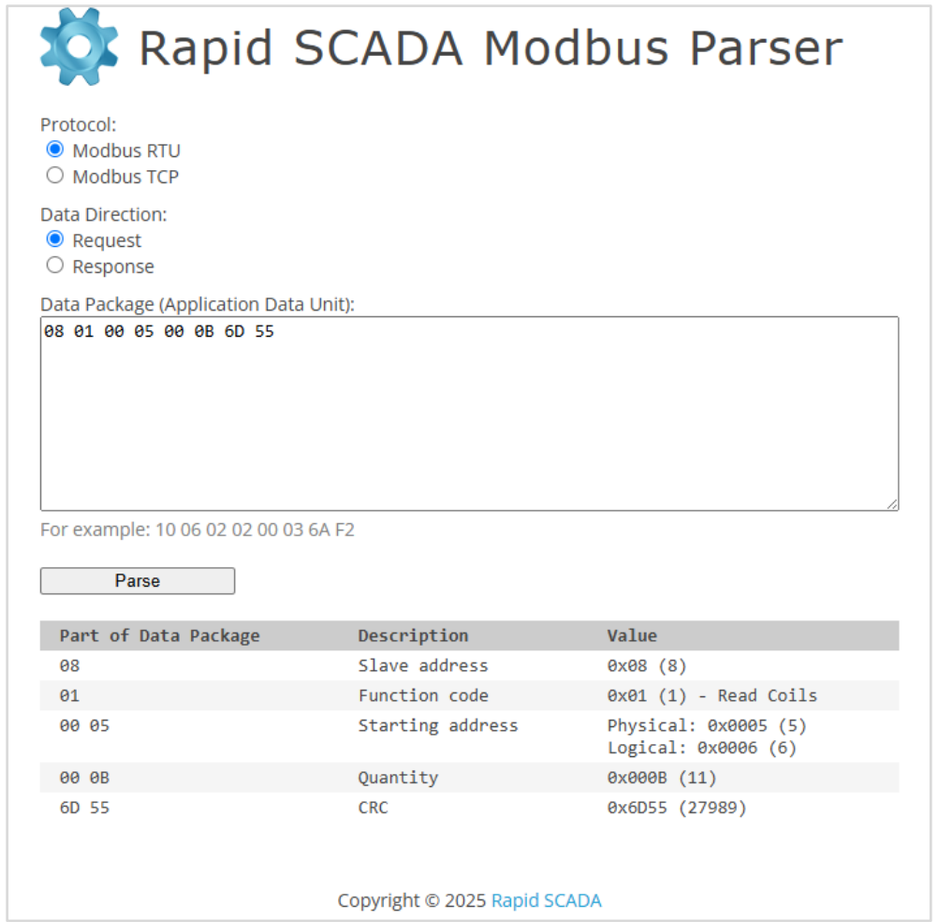
Формат запроса для функций чтения битов (**0x01** и **0x02**) аналогичен функциям чтения регистров (**0x03** и **0x04**).

Рассмотрим такую ситуацию: пусть у нас есть slave–устройство с адресом **8**, и мы хотим считать с него **11 бит** с помощью функции **0x01 (Read Coils)** начиная с **бита 5** (т. е. фактически мы хотим считать биты **5...15**).

Для этого нам потребуется отправить следующий запрос:

> *08 01 00 05 00 0B 6D 55*

- 08h (8) – адрес slave–устройства (1 байт);
- 01h (3) – код функции (1 байт);
- 0005h (5) – адрес начального считываемого бита (2 байта);
- 000Bh (11) – количество считываемых битов, от 1 до **2000** (2 байта);
- 6D55h – контрольная сумма (2 байта).

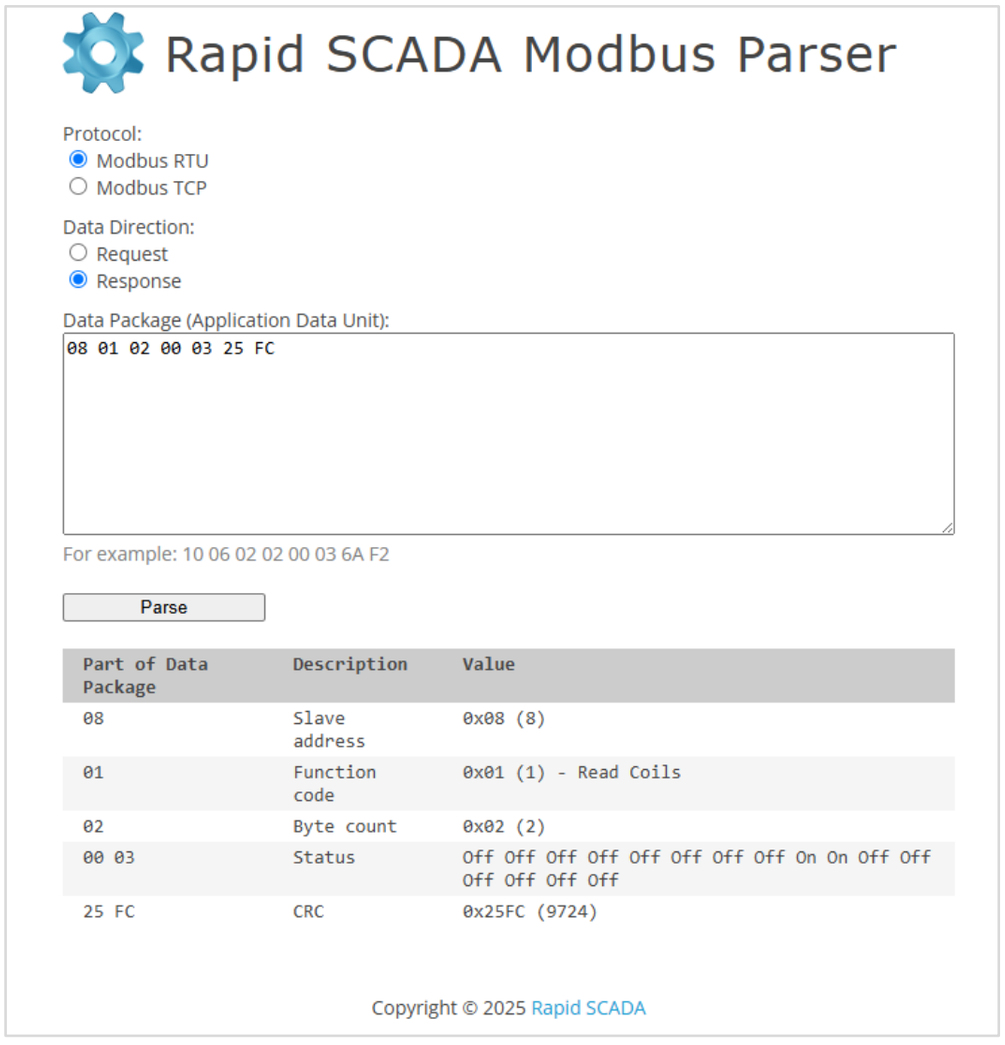


Предположим, slave–устройство действительно имеет адрес **8**, поддерживает функцию **0x01** и имеет биты с адресами **5...15**, и при этом биты с адресами **13** и **14** имеют значение **TRUE (1)**, а все остальные – **FALSE (0)**.

Тогда его ответ будет следующим:

> *08 01 02 00 03 25 FC*

- 08h (8) – адрес slave–устройства (1 байт);
- 01h (3) – код функции (1 байт);
- 0002h (2) – количество байт данных, в которых размещены запрошенные биты (2 байта);
- 0003h (11) – байты, содержащие запрошенные биты (от 2 до 250 байт в зависимости от количества бит);
- 25FCh – контрольная сумма (2 байта).



Запрошенные биты «упаковываются» в байты; **каждый байт содержит 8 бит**.

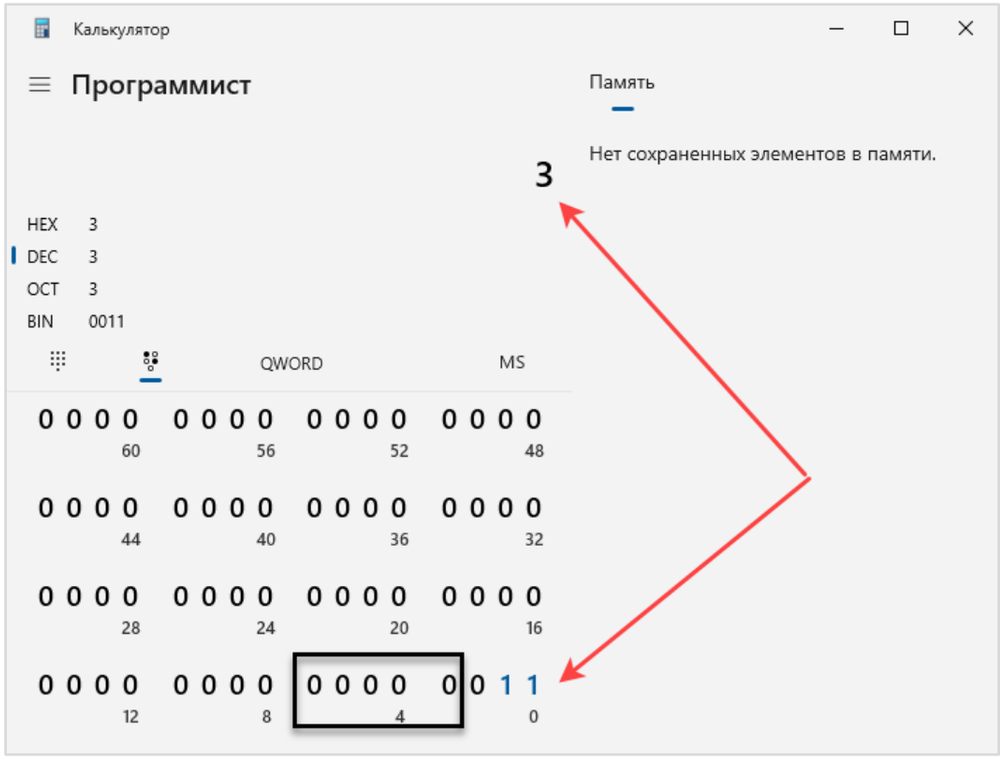
Младший бит первого байта соответствует биту с тем адресом, который был указан в запросе. Далее размещаются следующие за ним запрошенные биты.

Если количество запрошенных бит не кратно **8**, то «лишние» биты последнего байта считаются незначимыми и имеют значение **FALSE (0)**.

Мы запросили **11 бит** начиная с бита с адресом **5**. Мы знаем, что биты с адресами **13** и **14** имеют значение **TRUE (1)**, а все остальные – **FALSE (0)**. Полученные нами байты ответа подтверждают это:

- первый байт имеет значение **00h**. Поэтому все его 8 бит (которые соответствуют битам slave–устройства с адресами **5...12**) имеют значение **FALSE (0)**;
- второй байт имеет значение **03h**. Соответственно, два его младших бита (которые соответствуют битам slave–устройства с адресами **13...14**) имеют значение **TRUE (1)**, а все остальные биты имеют значение **FALSE (0)**. Из этих остальных бит один является «настоящим» и соответствует биту slave–устройства с адресом **15** (это последний из запрошенных нами битов), а все остальные биты являются незначимыми (master–устройство не будет анализировать их значения) и имеют значение **FALSE (0)**; их наличие связано лишь с тем, что передать меньше байта данных по протоколу Modbus невозможно. На последнем скриншоте (с калькулятором) эти биты выделены чёрной рамкой.

Байты данных ответа	Номер бита в байте	Modbus-адрес бита	Значение бита
00h	0 (младший)	5	FALSE (0)
	1	6	FALSE (0)
	2	7	FALSE (0)
	3	8	FALSE (0)
	4	9	FALSE (0)
	5	10	FALSE (0)
	6	11	FALSE (0)
	7 (старший)	12	FALSE (0)
03h	0 (младший)	13	TRUE (1)
	1	14	TRUE (1)
	2	15	FALSE (0)
	3	-	FALSE (0)
	4	-	FALSE (0)
	5	-	FALSE (0)
	6	-	FALSE (0)
	7 (старший)	-	FALSE (0)


Калькулятор
Программист
Память
Нет сохраненных элементов в памяти.
3
HEX 3
DEC 3
OCT 3
BIN 0011
QWORD
MS
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
60 56 52 48
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
44 40 36 32
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
28 24 20 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
12 8 4 0

Формат запроса протокола Modbus RTU – функция 0x0F (Write Multiple Coils)

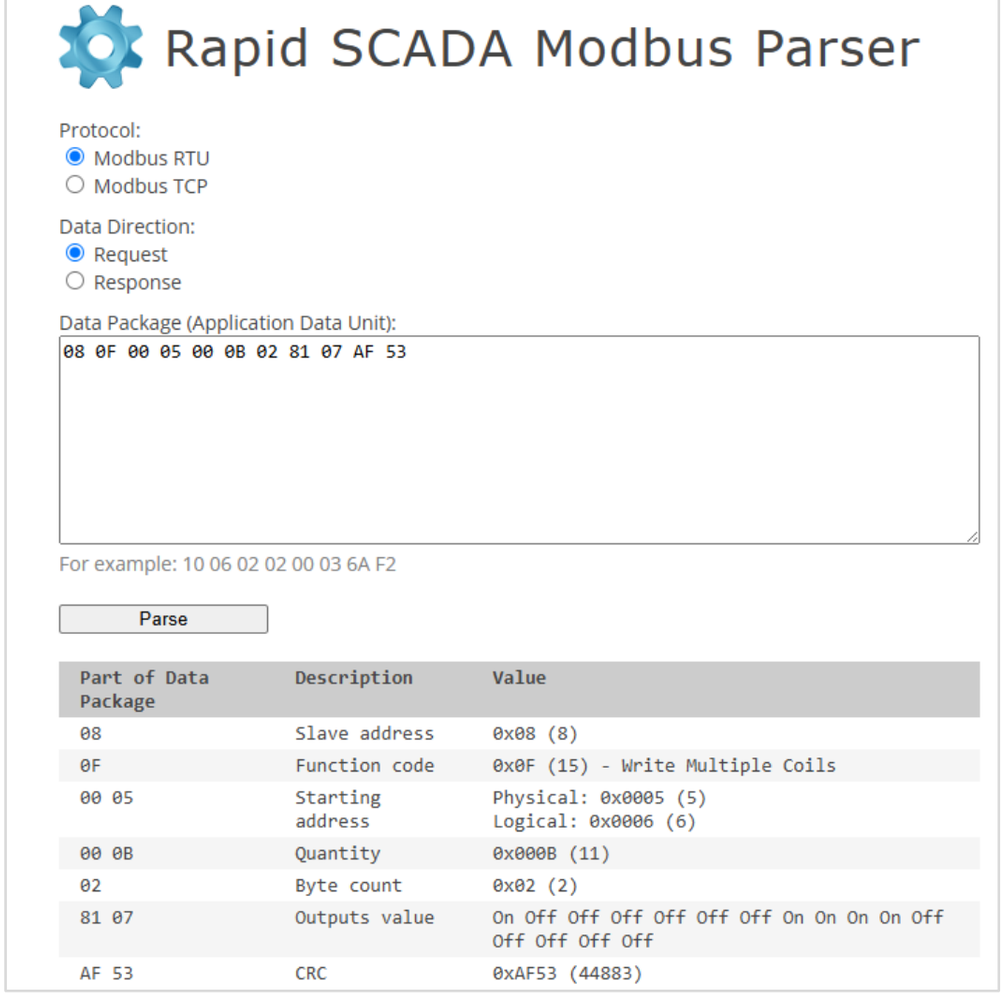
Теперь запишем несколько бит нашего гипотетического slave–устройства с адресом **8**.

Давайте отправим запрос на запись всё тех же бит – с адресами **5...15** – и установим биты с адресами **5**, **12** и **15** в значение **TRUE(1)**, не сбрасывая при этом уже установленные биты с адресами **13** и **14**.

Он будет выглядеть следующим образом:

> *08 0F 00 05 00 0B 02 81 07 AF 53*

- 08h (8) – адрес slave–устройства (1 байт);
- 0Fh (15) – код функции (1 байт);
- 0005h (2) – адрес начального записываемого бита (2 байта);
- 000Bh (11) – количество записываемых битов, от 1 до **1968** (2 байта);
- 02h (2) – количество записываемых байт данных (1 байт);
- 8107h – записываемые данные (от 2 до 246 байт в зависимости от количества записываемых в запросе битов);
- AF53h – контрольная сумма (2 байта).



Rapid SCADA Modbus Parser

Protocol:
- (•) Modbus RTU
- ( ) Modbus TCP

Data Direction:
- (•) Request
- ( ) Response

Data Package (Application Data Unit):

```
08 0F 00 05 00 0B 02 81 07 AF 53
```

For example: 10 06 02 02 00 03 6A F2

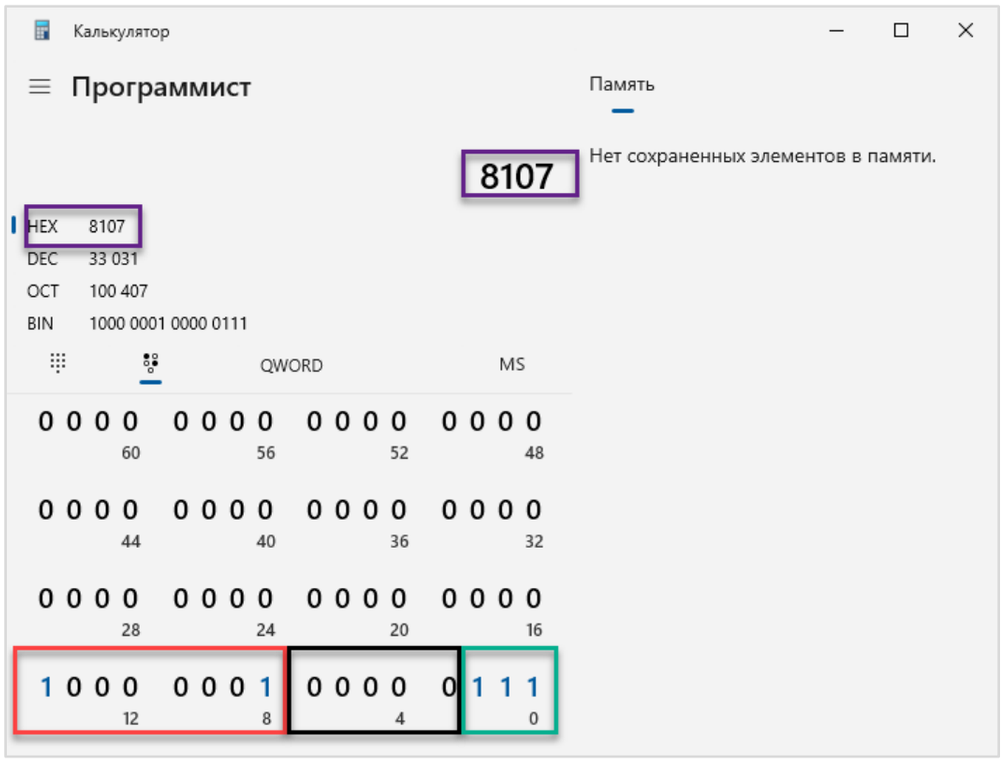
Parse

Part of Data Package	Description	Value
08	Slave address	0x08 (8)
0F	Function code	0x0F (15) - Write Multiple Coils
00 05	Starting address	Physical: 0x0005 (5) Logical: 0x0006 (6)
00 0B	Quantity	0x000B (11)
02	Byte count	0x02 (2)
81 07	Outputs value	On Off Off Off Off Off Off On On On On Off Off Off Off Off
AF 53	CRC	0xAF53 (44883)

Структура запроса идентична запросу с кодом функции **0x10 (Write Multiple Registers)**.

Интерес для нас представляет только поле записываемых данных.

Посмотрим его побитовое представление в калькуляторе Windows (или любом другом аналогичном калькуляторе):



Первый записываемый байт данных имеет значение **81h**. На скриншоте он выделен красным цветом.

В нём установлены младший бит (на скриншоте под ним написано число **8** – это порядковый номер бита во введённом в калькуляторе 2–байтовом значении), который соответствует биту с адресом **5** нашего slave–устройства, и старший (самый левый бит в выделенном красным фрагменте скриншота), который соответствует биту с адресом **12**.

Второй записываемый байт данных имеет значение **07h**.

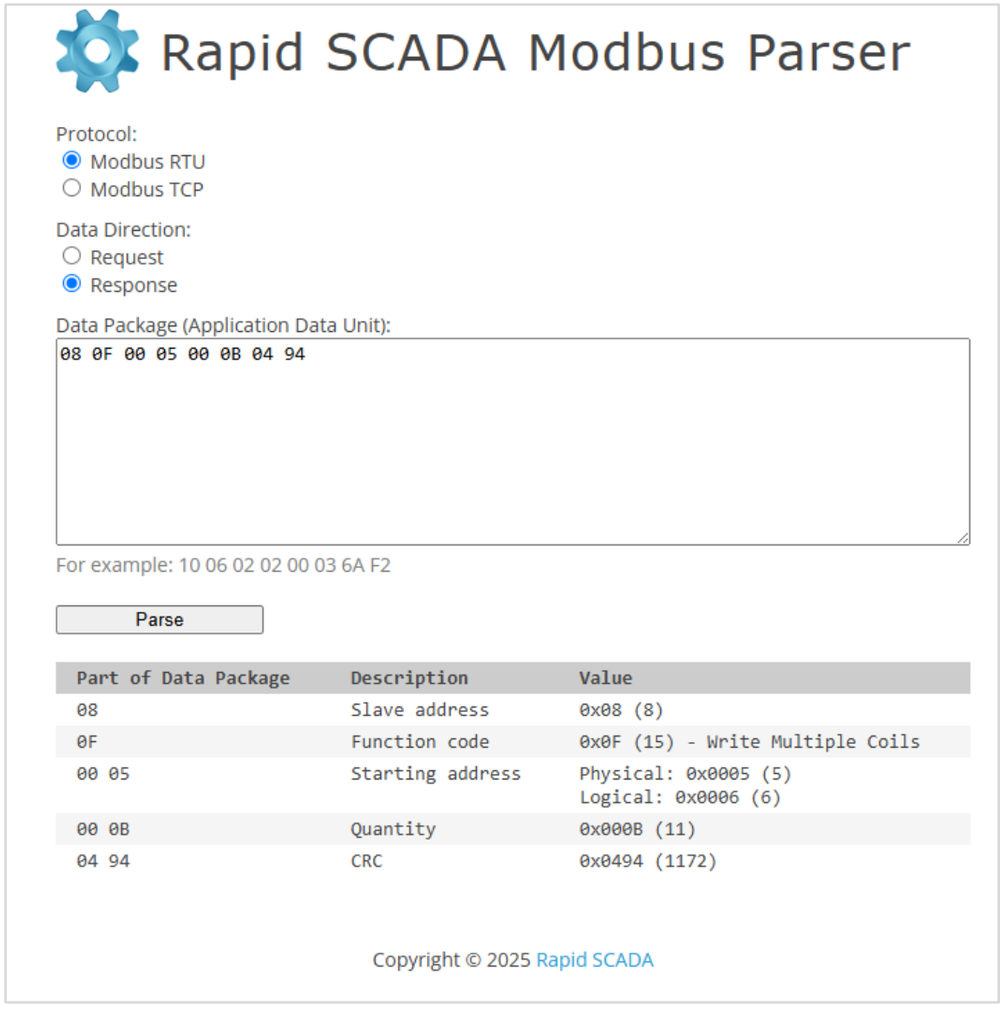
Два его младших бита имеют значение **TRUE (1)**, потому что они были установлены ранее; мы знаем это из предыдущего шага. Следующий за ними бит мы тоже установили в значение **TRUE**, потому что он соответствует нужному нам биту с адресом 15 нашего slave–устройства. Эти три бита выделены зелёным цветом. Остальные 5 бит байта являются незначимыми и выделены **чёрным** цветом; их наличие связано лишь с тем, что передать меньше байта данных по протоколу Modbus невозможно.

Формат ответа протокола Modbus RTU – функция 0x0F (Write Multiple Coils)

В ответ slave–устройство отправит мастеру следующий пакет данных:

> *08 0F 00 05 00 0B 04 94*

То есть ответ представляет собой первые 6 байт запроса (адрес slave–устройства, код функции, адрес начального регистра, количество регистров), дополненные заново рассчитанной контрольной суммой.

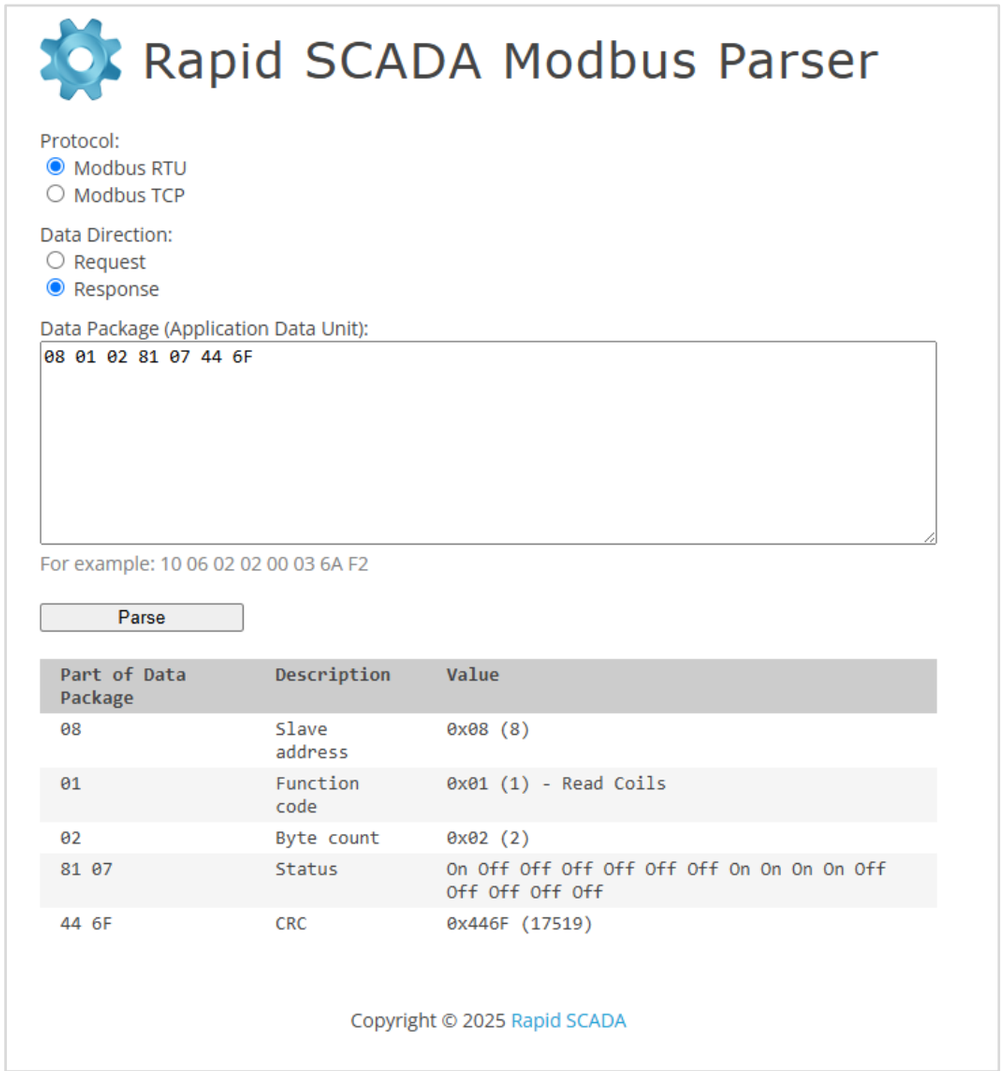


Если мы теперь повторно отправим устройству запрос на чтение этих же бит с помощью функции **0x01 (Read Coils)**:

> *08 01 00 05 00 0B 6D 55*

...то получим следующий ответ:

> *08 01 02 **81 07** 44 6F*

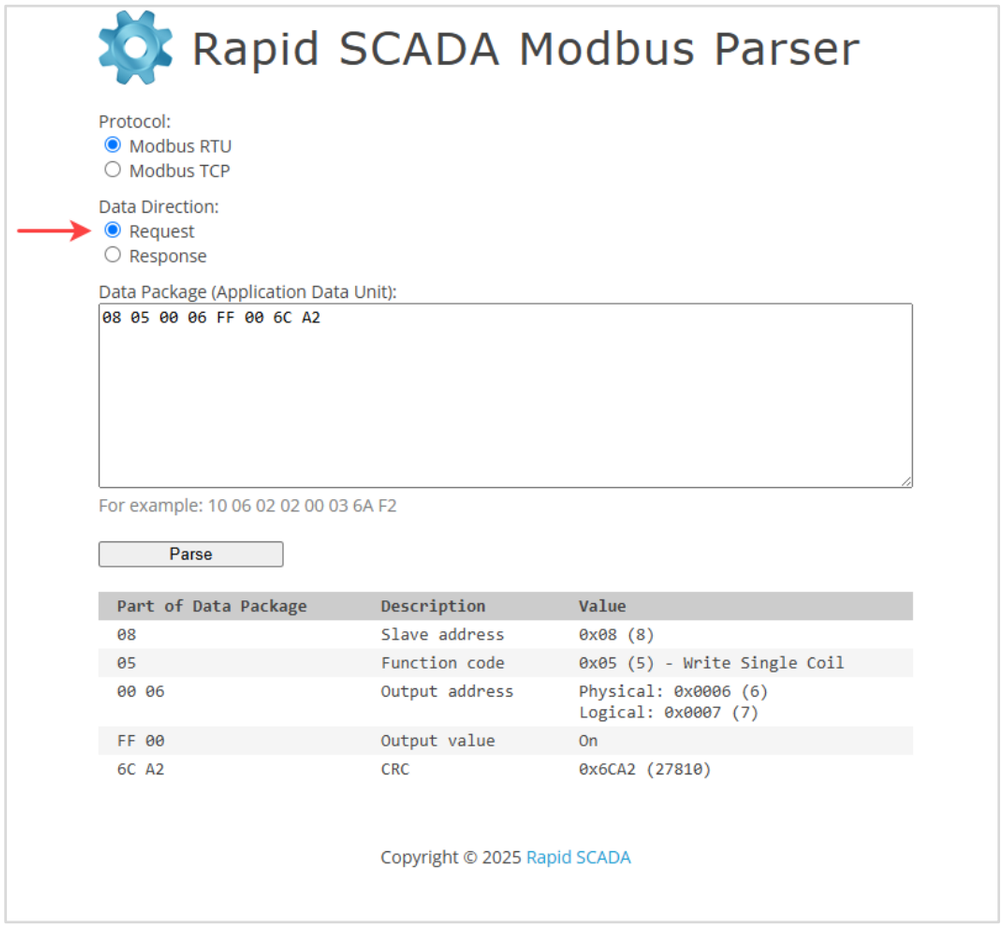


Формат ответа протокола Modbus RTU – функция 0x05 (Write Single Coil)

После рассмотрения предыдущих битовых функций – функция записи одного бита должна показаться вам предельно простой.

Установим в нашем slave–устройстве с адресом **8** бит с адресом **6** в значение **TRUE (1)**.

> *08 05 00 06* ***FF 00*** *6C A2*



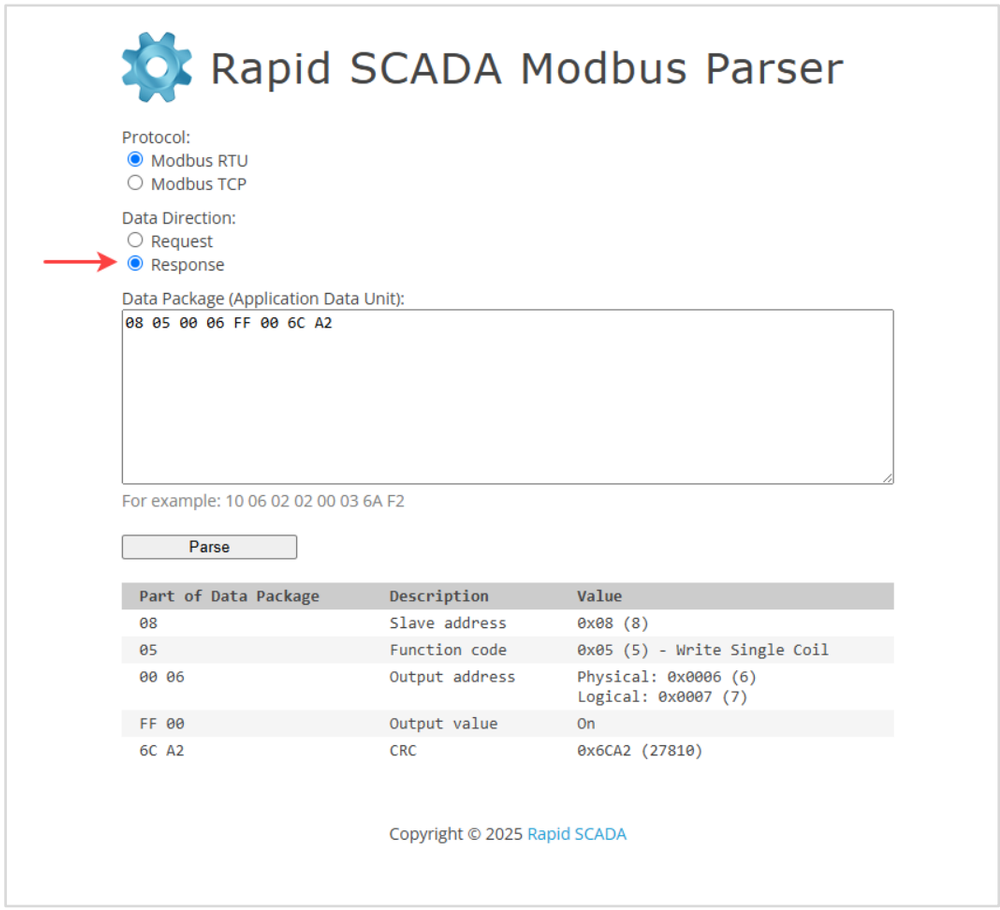
Выделенное **жирным** поле является полем значения. Если вы хотите записать в бит **TRUE** – то оно должно быть равным **FF00h**.

Если вы хотите сбросить бит в **FALSE** – то оно должно быть равным **0000h**.

В рамках функции **0x05** поле данных не может иметь иных значений. Если по каким–то причинам это случилось – то slave–устройство должно отправить ответ с кодом ошибки **0x03 (ILLEGAL DATA VALUE)**.

В случае получения корректного запроса на запись – ответ slave–устройства полностью идентичен запросу, как и в случае функции **0x06 (Write Single Register)**:

> *08 05 00 06 FF 00 6C A2*



Интересное наблюдение о размерах пакетов

Давайте вернёмся к запросу и ответу с кодом функции **0x03 (Read Holding Registers)**, с которых мы начали данный урок:

> *10 03 10 00 00 0D 83 8E*

> *10 03 1A D2 D0 CC 32 30 31 20 20 56 30 33 2E 30 30 30 34 00 00 41 FA 28 00 42 36 00 00 9B FB*

Обратите внимание на неочевидный момент – длина запроса для данной функции (и для аналогичной ей функции **0x04**) всегда будет составлять **8 байт**.

Длина ответа зависит от количества регистров, но из–за самой структуры ответа его размер в байтах всегда будет **нечётным**.

Для функции **0x10 (Write Multiple Registers)** размер запроса зависит от количества регистров, но из–за самой структуры ответа его размер в байтах всегда будет **нечётным**.

А вот ответ от slave–устройства для этой функции всегда будет составлять **8 байт**.

Выше речь шла о «хороших» ответах; длина ответа с кодом ошибки Modbus всегда будет составлять **5 байт**.

Имея эту информацию – можно написать программу–сниффер, которая «прослушивает» последовательную линию связи и анализирует проходящие по ней пакеты. Мы рассмотрим примеры подобных программ в уроке 2.17.

***

> *– А как дела обстоят с битовыми функциями?*

можете спросить вы.

Длина запроса на чтение с кодом функции **0x01** и **0x02** всегда составляет 8 байт.

Длина ответа на запрос записи с кодом функции **0x0F** тоже всегда составляет 8 байт.

А вот для ответов на запрос чтения и запрос записи вывести какую–то зависимость не получится, потому что для этих функций наименьшей порцией передаваемых данных является байт.

Более того – с битовыми функциями могут возникать крайне интересные ситуации. Например, пакет

> *10 01 03 01 00 18 6E C5*

одновременно является и **корректным запросом**, и **корректным ответом на этот запрос.** Поэтому анализировать такие запросы с помощью сниффера может быть крайне затруднительно.

Rapid SCADA Modbus Parser

Protocol:
- (•) Modbus RTU
- ( ) Modbus TCP

Data Direction:
- (•) Request
- ( ) Response

Data Package (Application Data Unit):

```
10 01 03 01 00 18 6E C5
```

For example: 10 06 02 02 00 03 6A F2

Parse

Part of Data Package	Description	Value
10	Slave address	0x10 (16)
01	Function code	0x01 (1) - Read Coils
03 01	Starting address	Physical: 0x0301 (769) Logical: 0x0302 (770)
00 18	Quantity	0x0018 (24)
6E C5	CRC	0x6EC5 (28357)

Copyright © 2025 Rapid SCADA

Rapid SCADA Modbus Parser

Protocol:
- (•) Modbus RTU
- ( ) Modbus TCP

Data Direction:
- ( ) Request
- (•) Response

Data Package (Application Data Unit):

```
10 01 03 01 00 18 6E C5
```

For example: 10 06 02 02 00 03 6A F2

Parse

Part of Data Package	Description	Value
10	Slave address	0x10 (16)
01	Function code	0x01 (1) - Read Coils
03	Byte count	0x03 (3)
01 00 18	Status	On Off Off Off Off Off Off Off Off Off Off Off Off Off Off Off Off Off Off On On Off Off Off
6E C5	CRC	0x6EC5 (28357)

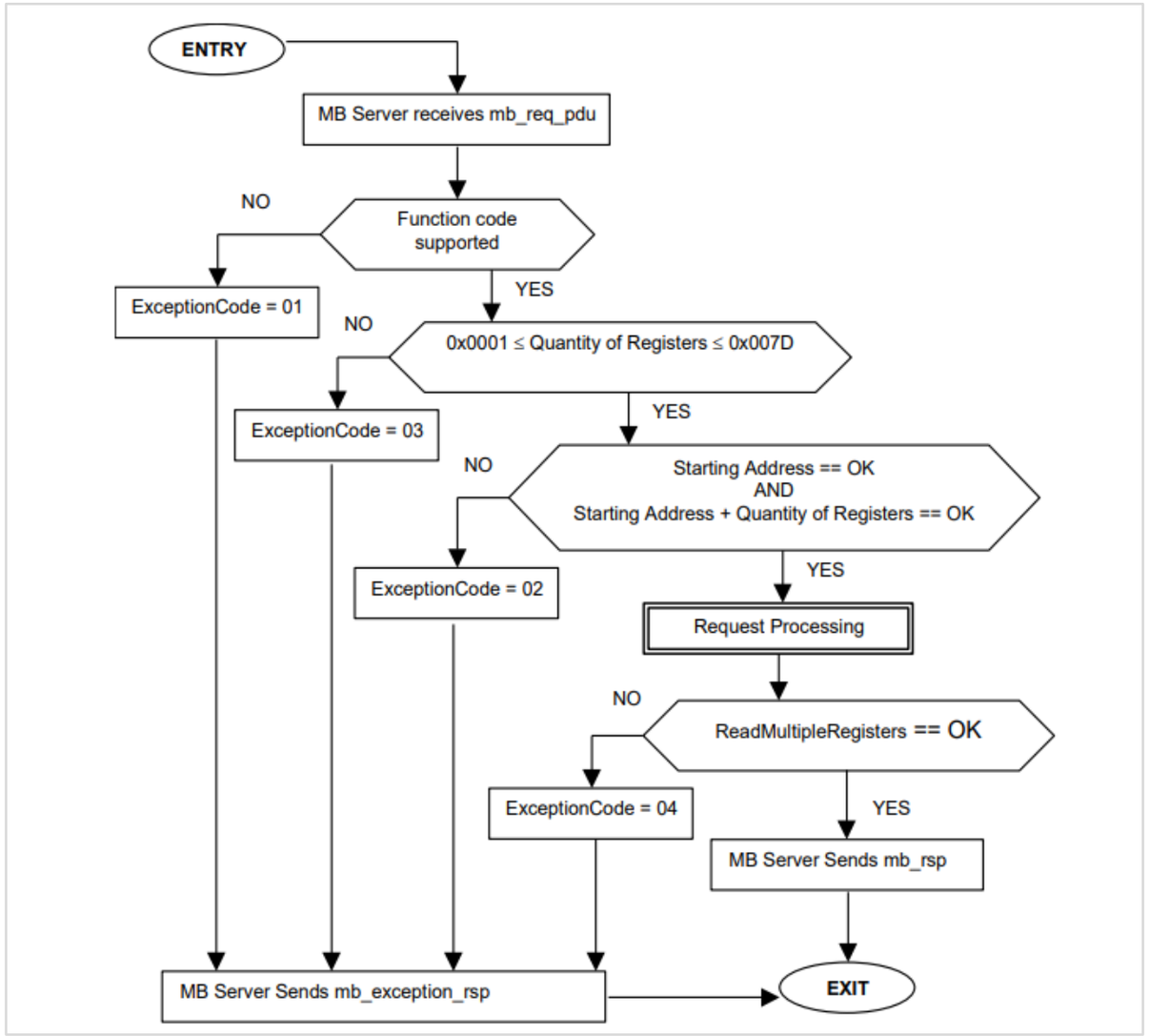
Copyright © 2025 Rapid SCADA

Диаграммы состояний для запросов и ответов

В спецификации Modbus для каждого запроса приведены диаграммы состояний, демонстрирующие, как именно slave–устройство должно анализировать запрос master–устройства, и какие коды ошибок (и в каких ситуациях) могут быть возвращены для каждой из функций.

> *В рамках диаграмм подразумевается, что проверки на адрес slave–устройства (совпадает ли адрес в запросе с адресом данного slave–устройства) и контрольную сумму уже проведены.*

Ниже приведена диаграмма состояний для функций **0x03 (Read Holding Registers)** и **0x04 (Read Inputs Registers).**



Реализации некоторых slave–устройств могут отступать от этой диаграммы. Например, в датчике ПД200–RS отправка пакетов с кодами ошибок может происходить в гораздо большем количестве ситуаций, чем предусмотрено спецификацией. В основном, эти ошибки связаны с особенностями конкретного датчика, но ошибка с кодом **0x00** будет оправлена в случае, если CRC, выделенное из пакета master–устройства, не сходится с рассчитанным. С точки зрения спецификации Modbus [3, п. 7] – в этой ситуации slave–устройство должно было проигнорировать запрос и не отправлять master–устройству никакого ответа.

Таблица 9.2 – Коды ошибок преобразователя с интерфейсом RS-485

Код	Описание ошибки
ERR00	Ошибка кода проверки CRC команды связи
ERR01	Ошибка функционального кода команды связи
ERR02	Неверный начальный адрес команды связи
ERR03	Неверное количество регистров команд связи/длина данных
ERR11	Ошибка данных вне диапазона отображения
ERR12	Выходной ток ниже установленного значения аварийного сигнала нижнего предела (на ЖК-дисплее отображается: OUT<AOLC)
ERR13	Выходной ток выше, чем установленное значение верхнего предела аварийного сигнала (OUT>AOHC)
ERR20	Данные внутренней калибровки зоны 1 продукта повреждены (No CAL)
ERR21	Данные внутренней калибровки зоны 2 продукта повреждены (No CAL)
ERR22	Внутренние резервные данные продукта повреждены (No BAK)
ERR051	Параметр LRV находится вне допустимого диапазона, должен соответствовать: -19999≤LRV≤99999
ERR052	Параметр URV находится вне допустимого диапазона, должен соответствовать: -19999≤URV≤99999
ERR056	Параметр KK находится вне допустимого диапазона и должен соответствовать: 1,000≤KK≤ 1,999
ERR057	Параметр FIXC находится вне допустимого диапазона и должен соответствовать: 0≤FIXC≤7
ERR058	Параметр AOLC находится вне допустимого диапазона и должен соответствовать: 3,500≤AOLC≤3,800
ERR059	Параметр AOHC находится вне допустимого диапазона и должен соответствовать: 20,800≤AOHC≤24,000
ERR060	Параметр BT находится вне допустимого диапазона и должен удовлетворять: 0≤BT≤5
ERR061	Параметр DE находится вне допустимого диапазона и должен соответствовать: 1≤DE≤ 247
ERR062	Параметр OddP находится вне допустимого диапазона и должен соответствовать: 0≤OddP≤2
ERR063	Параметр Stop находится вне допустимого диапазона и должен удовлетворять следующим требованиям: 0≤Stop≤ 1
ERR064	Параметр FFT находится вне допустимого диапазона и должен соответствовать: 0≤FFT≤3
ERR065	Параметр UUR находится вне допустимого диапазона и должен соответствовать: 0≤UUR≤22

Подведение итогов

Мы изучили на примерах формат запросов и ответов для всех основных функций Modbus и ознакомились с онлайн–сервисом Rapid SCADA Modbus Parser, который облегчает их анализ.

В реальной жизни «сборкой» запросов и анализом ответов будет заниматься используемое вами ПО – запущенное на ПК или устройствах системы автоматизации. Но умение читать логи обмена и понимать, что происходит на линии связи, является ценным навыком, существенно упрощающим отладку.

2.9 Практика: опрос ТРМ1–У3

В уроке 2.6 мы настроили опрос ТРМ201 с помощью MasterOPC Universal Modbus Server.

В тот раз мы упомянули, что к настоящему моменту линейка ТРМ2хх уже снята с продажи.

Её развитием является линейка ТРМ–Ух; аналогом ТРМ201 в этой линейке является ТРМ1.

Документация на прибор доступна по ссылке:

https://owen.ru/product/trm1/documentation

Мы будем использовать документ «Таблица регистров Modbus ТРМ1».

Современные ТРМ могут конфигурироваться не только с помощью экрана и кнопок, но и по интерфейсу USB.

Начиная с 2024 года для их подключения к ПК используется кабель USB Type–C; ранее использовался MicroUSB.

В принципе, прибор получит питание по этому кабелю, и через него же мы сможем организовать опрос по Modbus.

Но полученного питания не хватит для обеспечения работы аналогового входа – так что лучше подключить внешний источник питания.

Для получения значения на аналоговом входе потребуется подключить к нему датчик температуры (список поддерживаемых датчиков приведён в руководстве по эксплуатации) и выбрать тип этого датчика в настройках прибора.

Если же у вас нет в наличии датчика – вы можете выбрать в настройках тип датчика **ТХК (L)** [или любую другую термопару, в диапазон измерений которой попадает комнатная температура] и соединить проводом клеммы **X–2** и **X–3**. Это приведёт к тому, что измеренное значение будет получено от встроенного термодатчика ТРМ и будет приблизительно равно температуре в помещении.

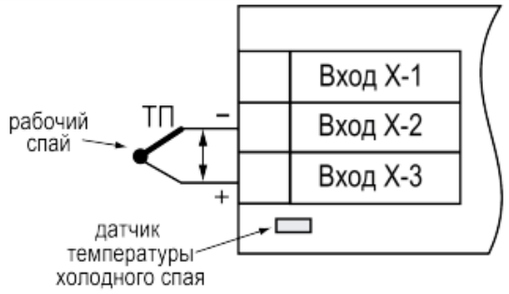


Рисунок 6 – Схема подключения термопары

Подключение к ТРМ1 с помощью OWEN Configurator (часть 1)

Утилита **OWEN Configurator** является универсальным конфигуратором для современных приборов ОВЕН.

С её помощью можно подключиться к прибору по интерфейсу USB, Ethernet и RS–485 (набор поддерживаемых интерфейсов зависит от конкретного прибора) и:

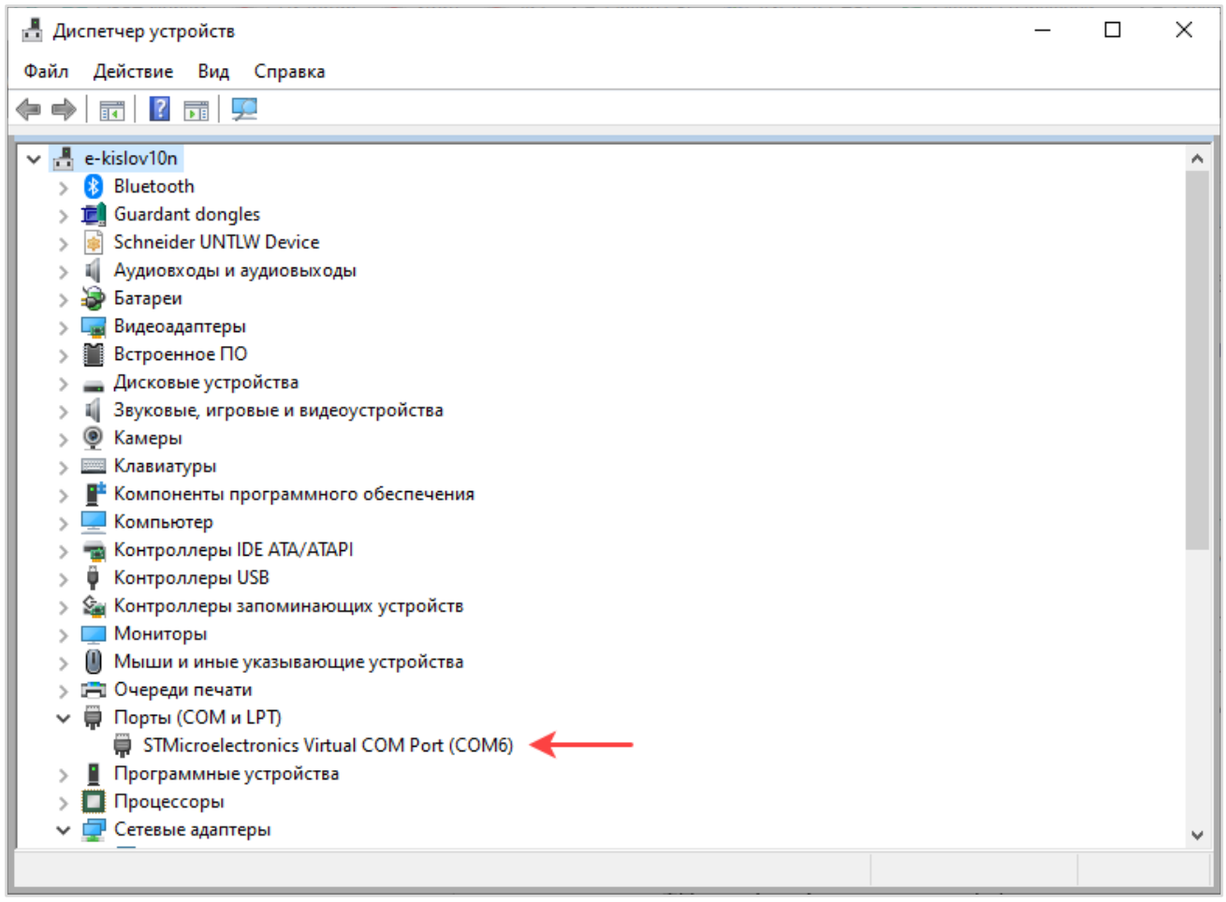
- посмотреть текущие значения параметров прибора и изменить их;
- обновить прошивку прибора;
- выполнить другие специфические операции, доступные для конкретного прибора (*например, провести его юстировку или выгрузить архив*).

Утилита доступна для загрузки по ссылке:

https://owen.ru/product/owen_configurator/software

Загрузите и установите её. В процессе установки будут также установлены драйверы виртуальных COM–портов для приборов ОВЕН.

После завершения установки подключите ТРМ1 к ПК с помощью кабеля USB Type–C (или MicroUSB, если у вас не совсем новый ТРМ). В Диспетчере устройств Windows появится новый виртуальный COM–порт. У нас он получил номер **6** – запомним это.



Теперь запустите OWEN Configurator.

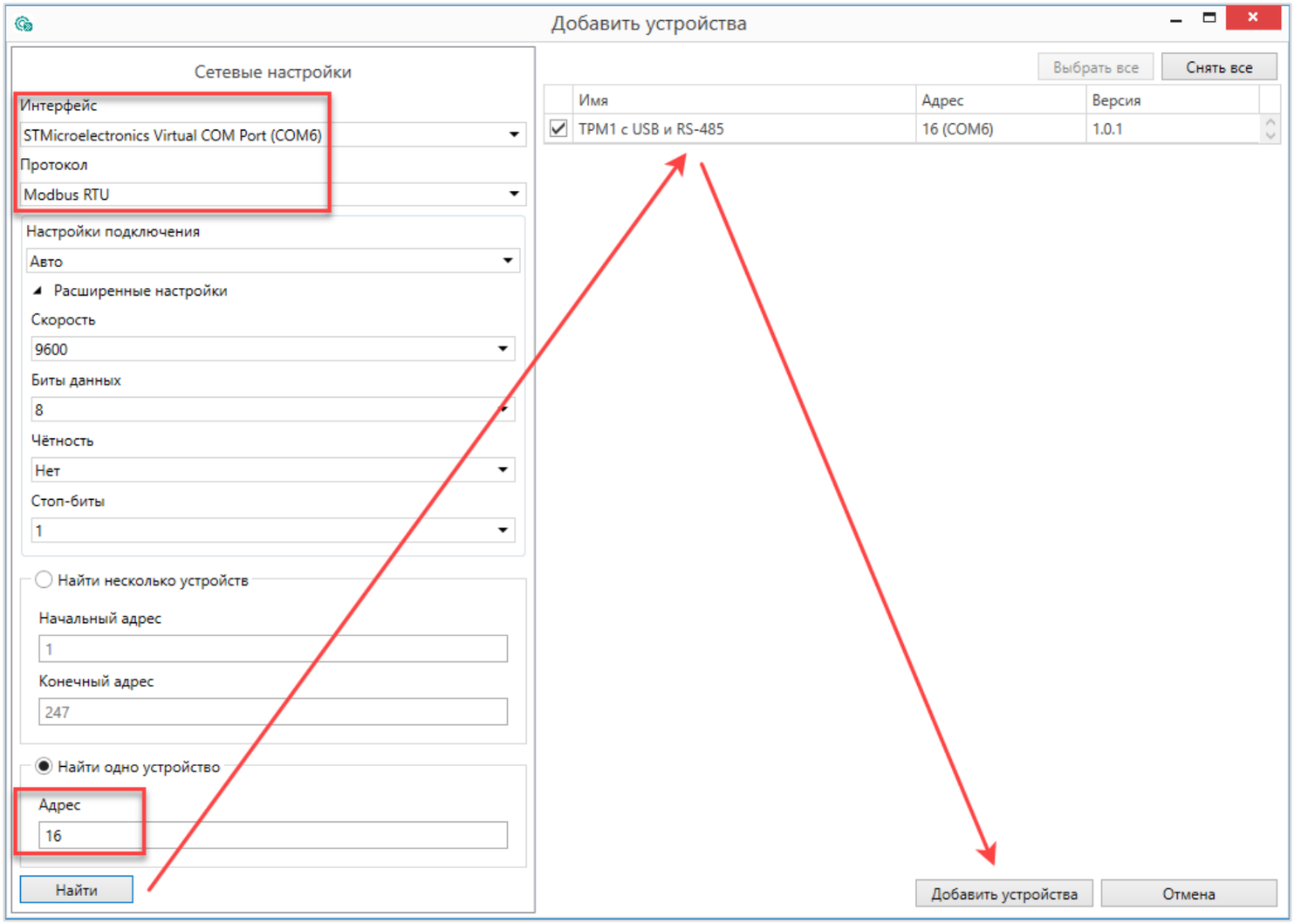
В качестве интерфейса выберите определённый чуть выше виртуальный COM–порт.

Задайте протокол **Modbus RTU**, режим **Найти одно устройство** и адрес – **16** (это адрес ТРМ по умолчанию).

> *Настройки COM–порта в данном случае не имеют значения, потому что мы используем виртуальный COM–порт.*

Нажмите кнопку **Найти**. Вы увидите найденное устройство в списке приборов.

Нажмите кнопку **Добавить устройства**.



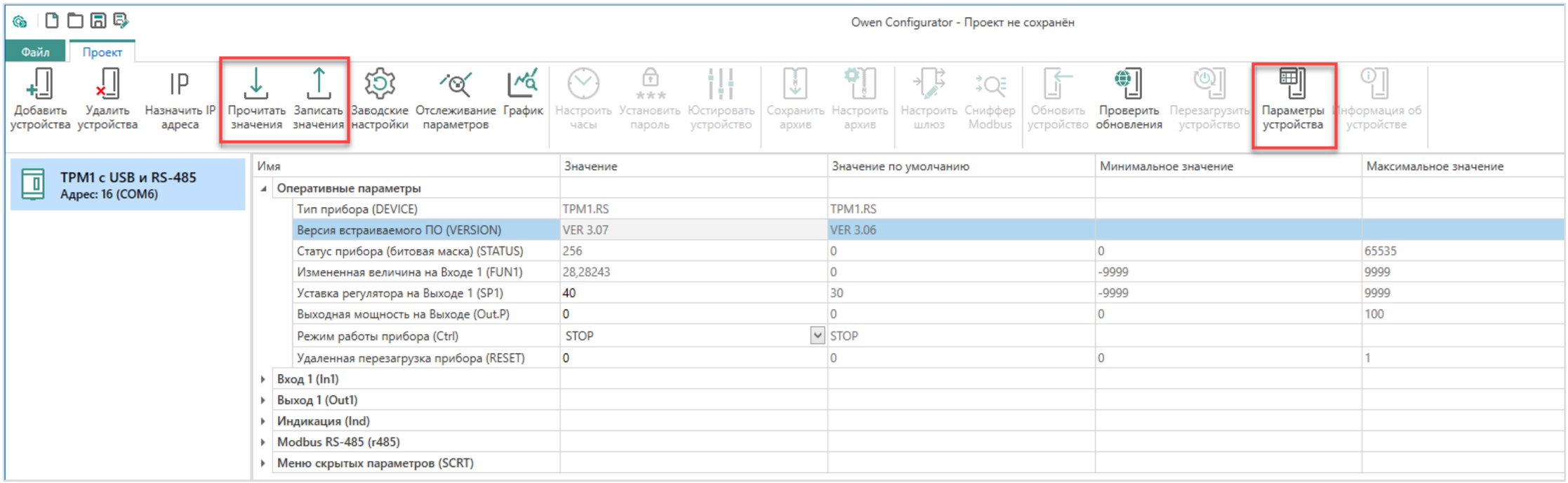
Подключение к ТРМ1 с помощью OWEN Configurator (часть 2)

В конфигураторе отображается список параметров прибора.

При нажатии на кнопку **Прочитать значения** – происходит их однократное считывание.

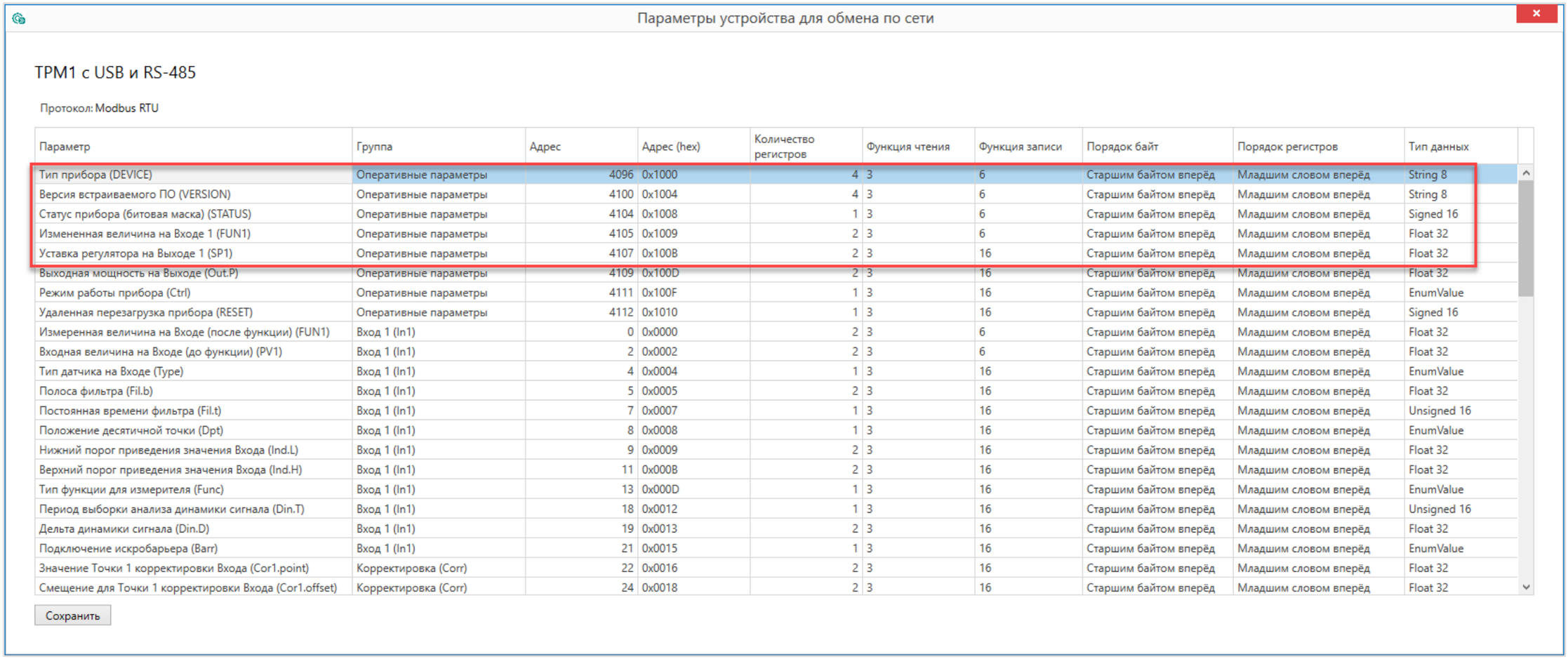
Вы можете изменить значения нужных вам параметров и нажать кнопку **Записать значения** для их однократной записи.

Обратите внимание, что применение большинства новых настроек происходит только после перезагрузки прибора по питанию.



Имя	Значение	Значение по умолчанию	Минимальное значение	Максимальное значение
Оперативные параметры				
Тип прибора (DEVICE)	TPM1.RS	TPM1.RS		
Версия встраиваемого ПО (VERSION)	VER 3.07	VER 3.06		
Статус прибора (битовая маска) (STATUS)	256	0	0	65535
Измененная величина на Входе 1 (FUN1)	28,28243	0	-9999	9999
Уставка регулятора на Выходе 1 (SP1)	40	30	-9999	9999
Выходная мощность на Выходе (Out.P)	0	0	0	100
Режим работы прибора (Ctrl)	STOP	STOP		
Удаленная перезагрузка прибора (RESET)	0	0	0	1
Вход 1 (In1)				
Выход 1 (Out1)				
Индикация (Ind)				
Modbus RS-485 (r485)				
Меню скрытых параметров (SCRT)				

С помощью кнопки **Параметры устройств**а вы можете увидеть карту регистров прибора:



Параметры устройства для обмена по сети

ТРМ1 с USB и RS-485

Протокол: Modbus RTU

Параметр	Группа	Адрес	Адрес (hex)	Количество регистров	Функция чтения	Функция записи	Порядок байт	Порядок регистров	Тип данных
Тип прибора (DEVICE)	Оперативные параметры	4096	0x1000	4	3	6	Старшим байтом вперёд	Младшим словом вперёд	String 8
Версия встраиваемого ПО (VERSION)	Оперативные параметры	4100	0x1004	4	3	6	Старшим байтом вперёд	Младшим словом вперёд	String 8
Статус прибора (битовая маска) (STATUS)	Оперативные параметры	4104	0x1008	1	3	6	Старшим байтом вперёд	Младшим словом вперёд	Signed 16
Измененная величина на Входе 1 (FUN1)	Оперативные параметры	4105	0x1009	2	3	6	Старшим байтом вперёд	Младшим словом вперёд	Float 32
Уставка регулятора на Выходе 1 (SP1)	Оперативные параметры	4107	0x100B	2	3	16	Старшим байтом вперёд	Младшим словом вперёд	Float 32
Выходная мощность на Выходе (Out.P)	Оперативные параметры	4109	0x100D	2	3	16	Старшим байтом вперёд	Младшим словом вперёд	Float 32
Режим работы прибора (Ctrl)	Оперативные параметры	4111	0x100F	1	3	16	Старшим байтом вперёд	Младшим словом вперёд	EnumValue
Удаленная перезагрузка прибора (RESET)	Оперативные параметры	4112	0x1010	1	3	16	Старшим байтом вперёд	Младшим словом вперёд	Signed 16
Измеренная величина на Входе (после функции) (FUN1)	Вход 1 (In1)	0	0x0000	2	3	6	Старшим байтом вперёд	Младшим словом вперёд	Float 32
Входная величина на Входе (до функции) (PV1)	Вход 1 (In1)	2	0x0002	2	3	6	Старшим байтом вперёд	Младшим словом вперёд	Float 32
Тип датчика на Входе (Type)	Вход 1 (In1)	4	0x0004	1	3	16	Старшим байтом вперёд	Младшим словом вперёд	EnumValue
Полоса фильтра (Fil.b)	Вход 1 (In1)	5	0x0005	2	3	16	Старшим байтом вперёд	Младшим словом вперёд	Float 32
Постоянная времени фильтра (Fil.t)	Вход 1 (In1)	7	0x0007	1	3	16	Старшим байтом вперёд	Младшим словом вперёд	Unsigned 16
Положение десятичной точки (Dpt)	Вход 1 (In1)	8	0x0008	1	3	16	Старшим байтом вперёд	Младшим словом вперёд	EnumValue
Нижний порог приведения значения Входа (Ind.L)	Вход 1 (In1)	9	0x0009	2	3	16	Старшим байтом вперёд	Младшим словом вперёд	Float 32
Верхний порог приведения значения Входа (Ind.H)	Вход 1 (In1)	11	0x000B	2	3	16	Старшим байтом вперёд	Младшим словом вперёд	Float 32
Тип функции для измерителя (Func)	Вход 1 (In1)	13	0x000D	1	3	16	Старшим байтом вперёд	Младшим словом вперёд	EnumValue
Период выборки анализа динамики сигнала (Din.T)	Вход 1 (In1)	18	0x0012	1	3	16	Старшим байтом вперёд	Младшим словом вперёд	Unsigned 16
Дельта динамики сигнала (Din.D)	Вход 1 (In1)	19	0x0013	2	3	16	Старшим байтом вперёд	Младшим словом вперёд	Float 32
Подключение искробарьера (Barr)	Вход 1 (In1)	21	0x0015	1	3	16	Старшим байтом вперёд	Младшим словом вперёд	EnumValue
Значение Точки 1 корректировки Входа (Cor1.point)	Корректировка (Corr)	22	0x0016	2	3	16	Старшим байтом вперёд	Младшим словом вперёд	Float 32
Смещение для Точки 1 корректировки Входа (Cor1.offset)	Корректировка (Corr)	24	0x0018	2	3	16	Старшим байтом вперёд	Младшим словом вперёд	Float 32

Сохранить

В ней есть вся нужна нам информация для настройки опроса в OPC–сервере.

Мы будем считывать те же самые 5 параметров, что и в уроке 2.6:

- Тип прибора;
- Версию встраиваемого ПО (прошивки);
- Статус;
- Измеренную величину на аналоговом входе;
- Уставку регулятора.

Уставку регулятора мы будем ещё и записывать.

Обратите внимание, что в отличие ТРМ201 мы сможем записать уставку прямо в формате **Float32** – так что не придётся добавлять два отдельных тега (SP_read и SP_write), как мы это делали для ТРМ201.

Напоследок отметим ряд следующих моментов:

- все параметры ТРМ1 могут быть считаны как функцией **0x03 (Read Holding Registers)**, так и **0x04 (Read Input Registers)**. Это указано в документации на прибор. Соответственно, в ТРМ1 используется общая модель памяти;

Таблица 1 – Чтение и запись параметров по протоколу Modbus

Операция	Функция
Чтение	0x03 или 0x04
Запись	0x10

- на скриншоте выше для параметров, которые не поддерживают запись (например, **Тип прибора**), указана поддержка функции записи **0x06 (Write Single Register)**. Это ошибка конфигуратора;
- параметры устройства со скриншота выше можно выгрузить в виде файла формата .csv с помощью кнопки **Сохранить**.

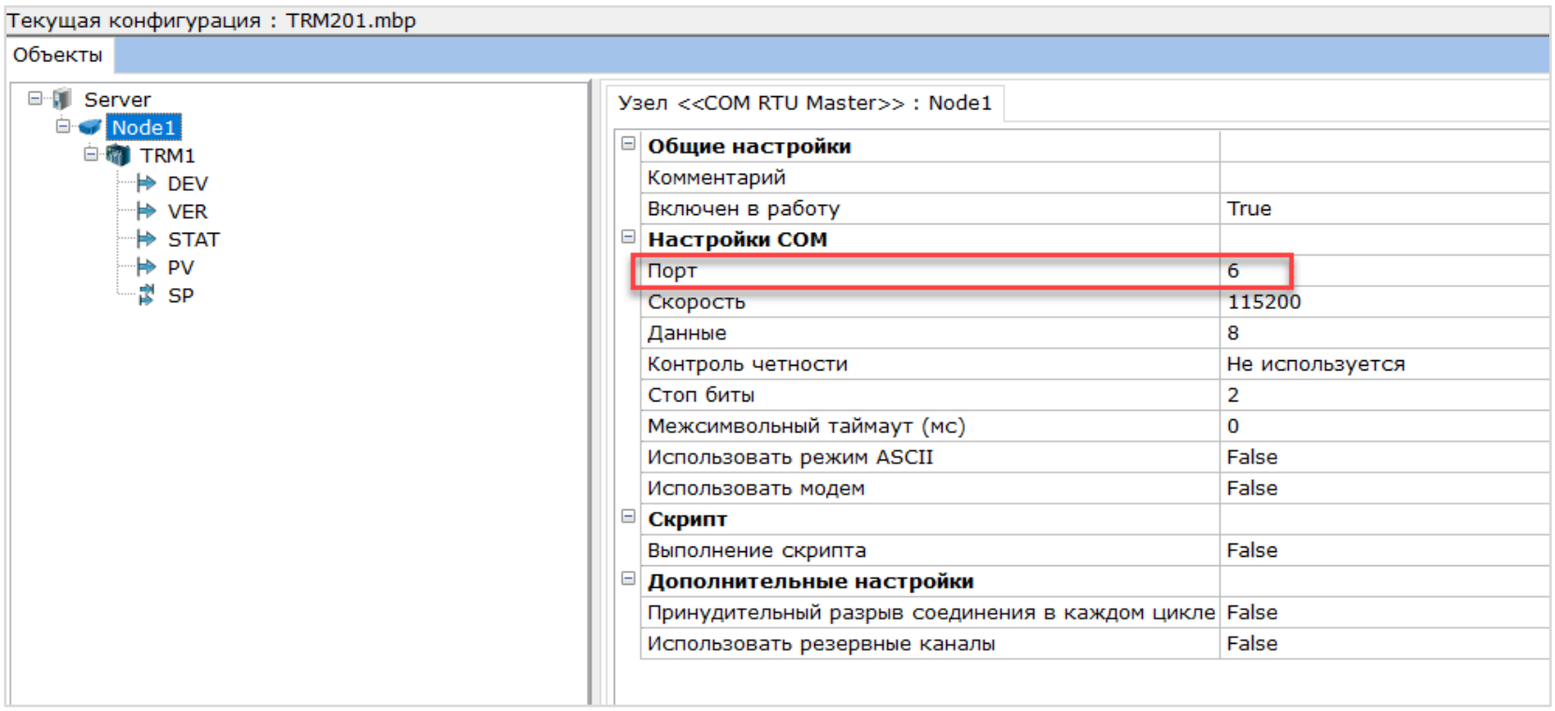
Настройка опроса в MasterOPC Universal Modbus Server

Мы уже упоминали, что ТРМ1 представляет собой современную версию отправленного на покой ТРМ201.

Поэтому во многих аспектах они совпадают – в том числе, в адресах регистров и типах интересующих нас оперативных параметров.

Мы можем открыть созданную нами ранее конфигурацию для ТРМ201 и внести в неё пару изменений:

- в настройках коммуникационного узла нужно исправить номер порта на **6** (потому что виртуальному COM–порту ТРМ у нас соответствует порт **COM6**).

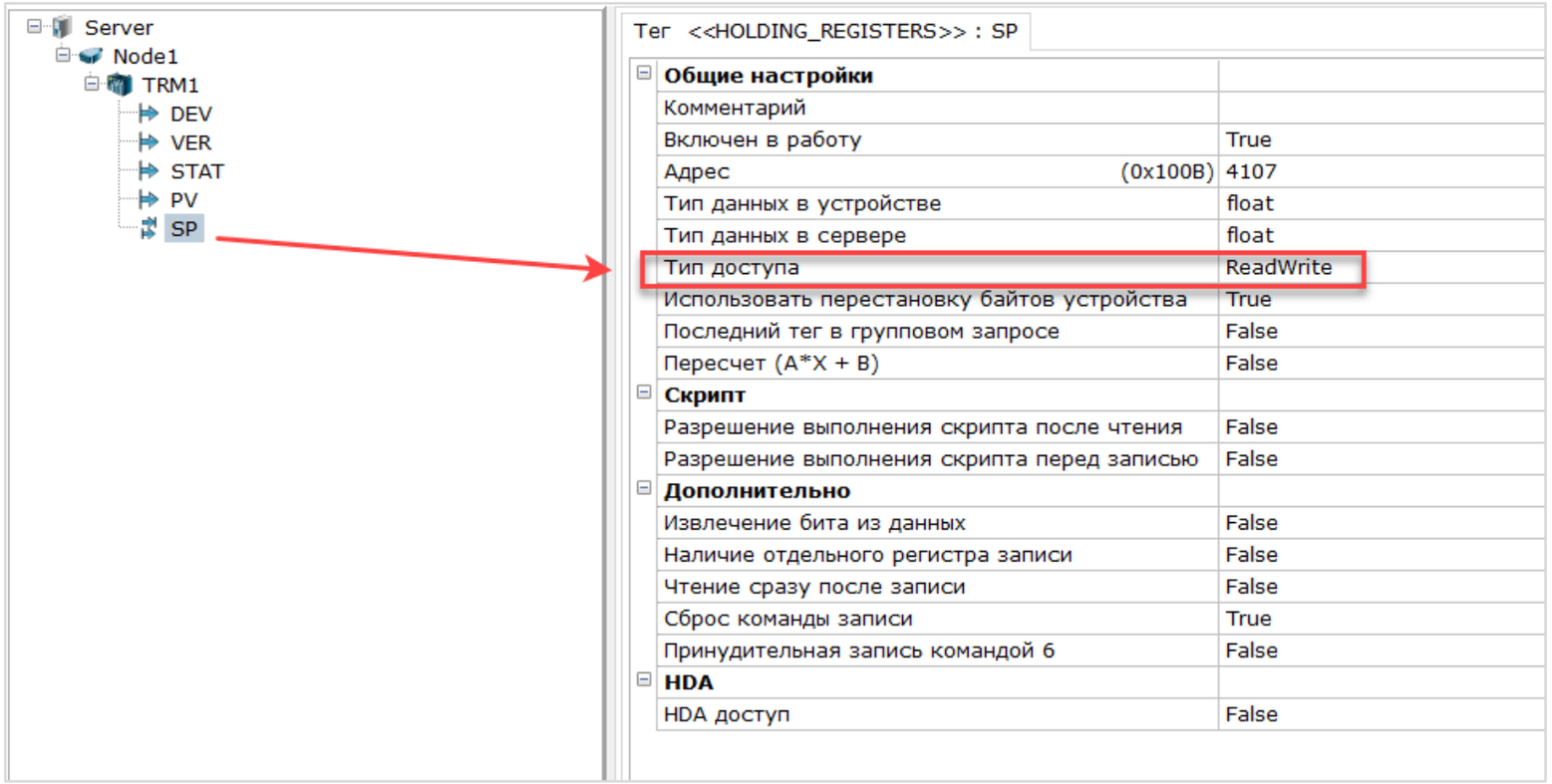


устройство **TRM201** переименовать в **TRM1**;

- переименовать параметр **SP_read** в **SP** и установить для него тип доступа **ReadWrite**, а параметр **SP_write** удалить – он больше не нужен.

Ссылка на готовый шаблон: https://ftp.owen.ru/ModbusCourse/01_OpcTemplates/TRM1.sdv

Для импорта шаблона нужно нажать правой кнопкой мыши на коммуникационный узел и использовать команду **Импорт устройства**.



Тег <<HOLDING_REGISTERS>> : SP

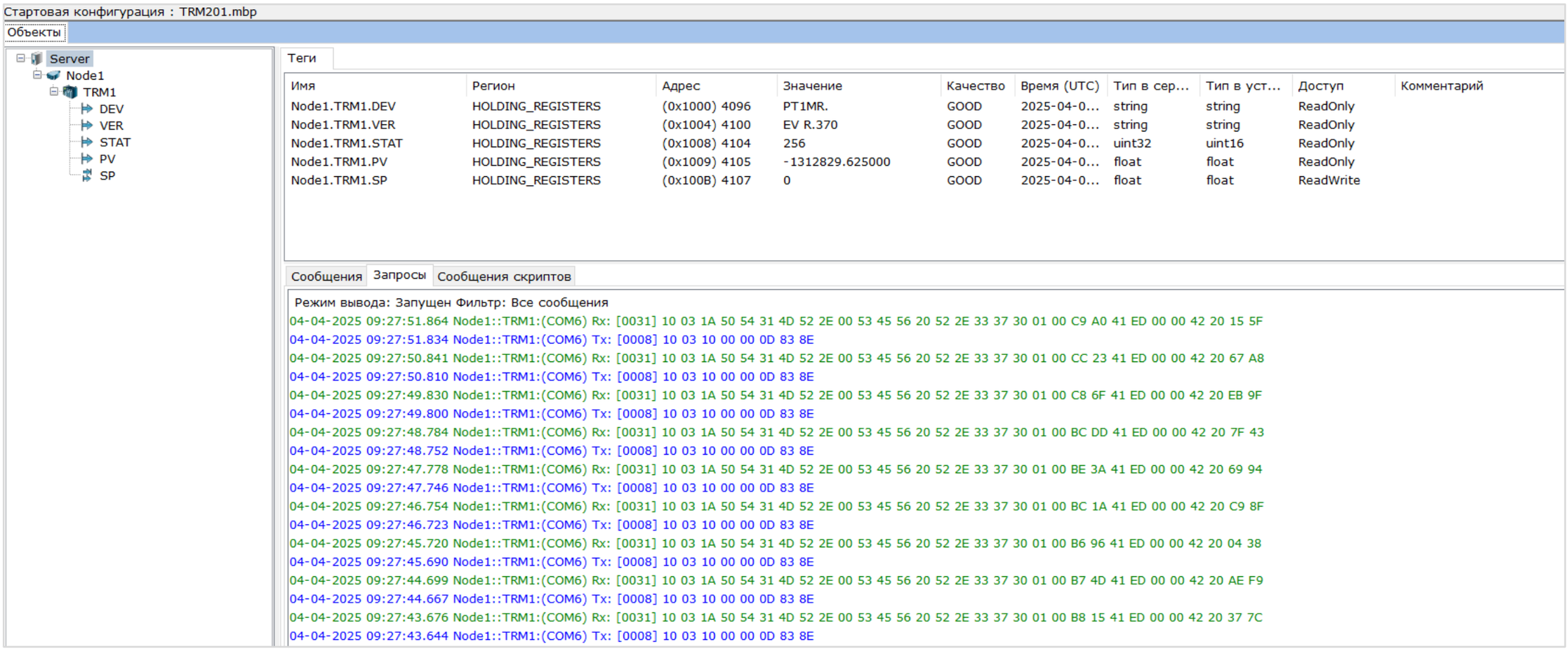
Общие настройки	
Комментарий	
Включен в работу	True
Адрес (0x100B)	4107
Тип данных в устройстве	float
Тип данных в сервере	float
Тип доступа	ReadWrite
Использовать перестановку байтов устройства	True
Последний тег в групповом запросе	False
Пересчет (A*X + B)	False
Скрипт	
Разрешение выполнения скрипта после чтения	False
Разрешение выполнения скрипта перед записью	False
Дополнительно	
Извлечение бита из данных	False
Наличие отдельного регистра записи	False
Чтение сразу после записи	False
Сброс команды записи	True
Принудительная запись командой 6	False
HDA	
HDA доступ	False

Проверка обмена

> *Перед запуском OPC–сервера убедитесь, что в данный момент OWEN Configurator не производит никаких операций с прибором; можно просто удалить прибор в конфигураторе. Это связано с тем, что и OWEN Configurator, и OPC–сервер используют один и тот же виртуальный COM–порт – и в каждый момент времени с ним должна работать только одна из этих программ.*

В ожидании триумфа запустите OPC–сервер.

Вы увидите приблизительно следующую картину.



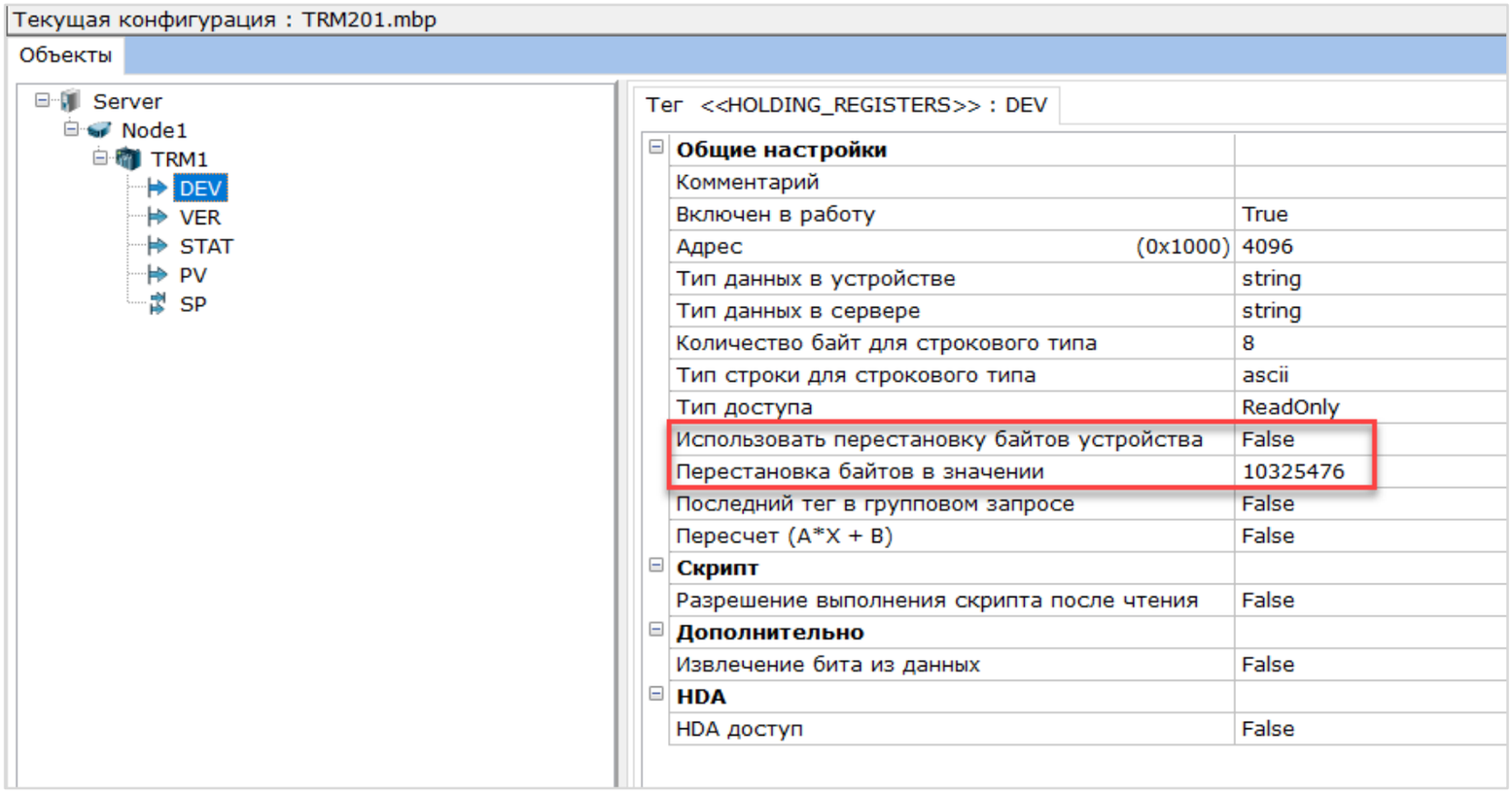
Значения параметров не похожи на те, которые мы ожидали, и это типичная ситуация при настройке опроса по Modbus.

Наиболее вероятная причина – не совпадает порядок байт и/или регистров в приборе и OPC–сервере.

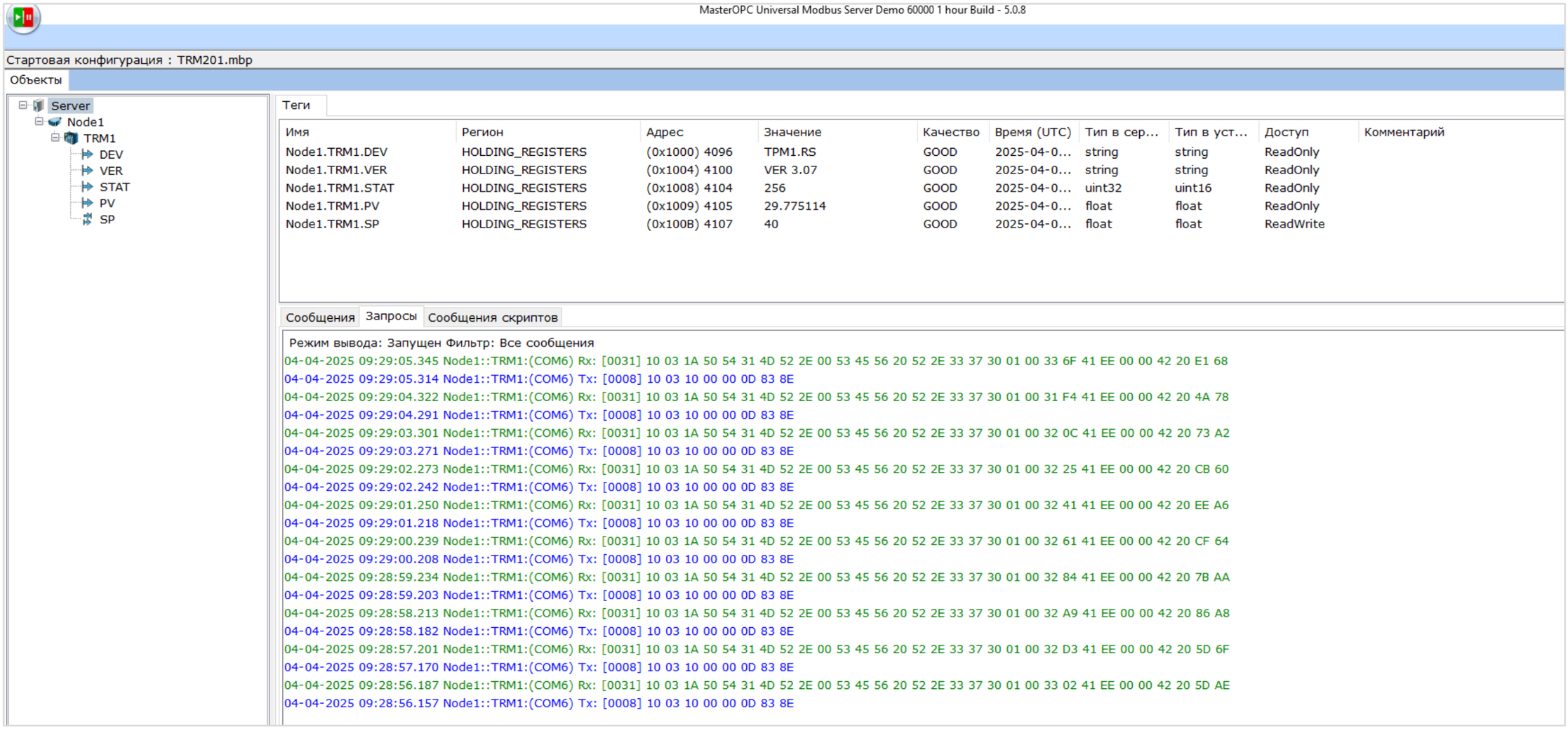
Помните, мы ещё в уроке 2.1 при опросе температуры датчика ПВТ110–RS устанавливали в OPC–сервере (*в то время мы ещё использовали Owen OPC Server*) настройку **Младшим регистром вперёд**?

Нам нужно опять это сделать.

В настройках каждого тега, кроме **STAT**, для параметр **Использовать перестановку байт устройства** установите значение **FALSE**. После этого автоматически добавится параметр **Перестановка байт в значении** – его мы не будем трогать.



Снова запустим OPC–сервер, и на этот раз увидим корректные значения параметров:



Давайте обсудим то, что сейчас произошло.

Обсуждение порядка байт и регистров

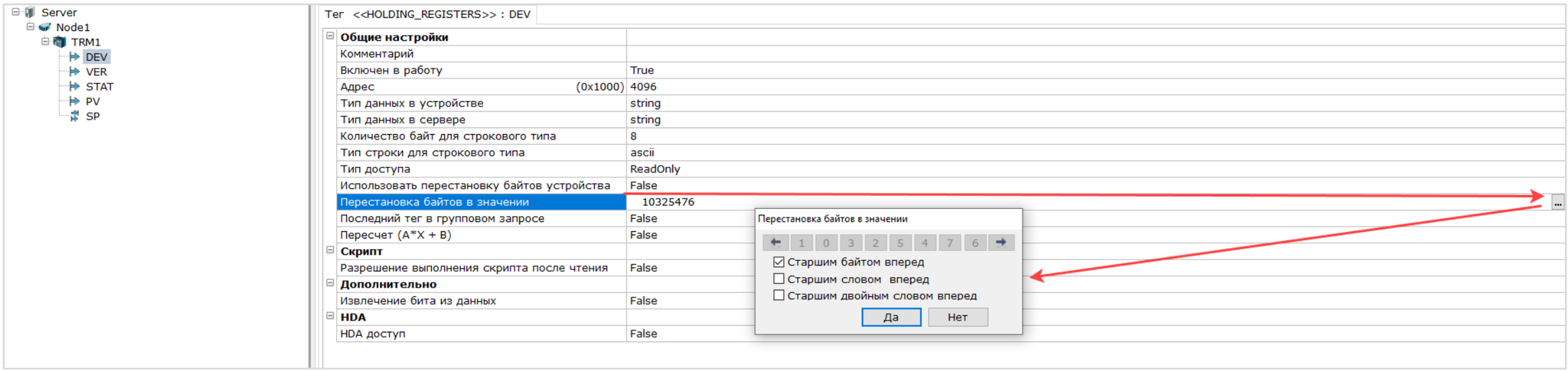
Как мы уже упоминали в уроке 2.5 – спецификация Modbus не определяет порядок регистров при передаче параметров, размер которых больше одного регистра.

Она определяет только порядок байт в регистре – «**Старшим байтом вперёд**» – но в некоторых приборах даже это требование спецификации игнорируется.

В предыдущем шаге не меняли настройки тега **STAT**, потому что этот тег занимает всего один регистр.

Для всех остальных тегов мы установили параметр **Использовать перестановку байт устройства** в значение **FALSE**. В этот момент у нас появилась возможность «подстраивать» порядок байт и регистров на стороне OPC–сервера с помощью параметра **Перестановка байт в значении**. По умолчанию значение этого параметра соответствует варианту **Старшим байтом вперёд/Младшим регистром вперёд**. В этом можно убедиться, нажав на кнопку «...», расположенную в конце строки параметра.

> *«Слово» тут используется как синоним термина «регистр».*



Для корректного представления значений параметров ТРМ1 потребовалось именно такое сочетание порядка байт и регистров. Можно ли было это как–то выяснить заранее?

Строго говоря – да.

В списке параметров устройства в OWEN Configurator была соответствующая информация:

Параметры устройства для обмена по сети

ТРМ1 с USB и RS-485

Протокол: Modbus RTU

Параметр	Группа	Адрес	Адрес (hex)	Количест регистро	Функция чтения	Функция записи	Порядок байт	Порядок регистров	Тип данных
Тип прибора (DEVICE)	Опера...	4096	0x1000	4	3	6	Старшим байтом вперёд	Младшим словом вперёд	String 8
Версия встраиваемого ПО (VERSION)	Опера...	4100	0x1004	4	3	6	Старшим байтом вперёд	Младшим словом вперёд	String 8
Статус прибора (битовая маска) (STATUS)	Опера...	4104	0x1008	1	3	6	Старшим байтом вперёд	Младшим словом вперёд	Signed 16
Измененная величина на Входе 1 (FUN1)	Опера...	4105	0x1009	2	3	6	Старшим байтом вперёд	Младшим словом вперёд	Float 32
Уставка регулятора на Выходе 1 (SP1)	Опера...	4107	0x100B	2	3	16	Старшим байтом вперёд	Младшим словом вперёд	Float 32

Если бы мы не нашли этой информации в OWEN Configurator и документации на прибор – нам бы потребовалось провести тесты со всеми возможными сочетаниями галочек **Старшим байтом вперёд** и **Старшим регистром вперёд**.

Галочка **Старшим двойным словом вперёд** нас не интересует. Она используется для попарной перестановки наборов из двух регистров в параметрах, размер которых составляет 4 регистра – например, параметров типа Double, Int64 и Uint64.

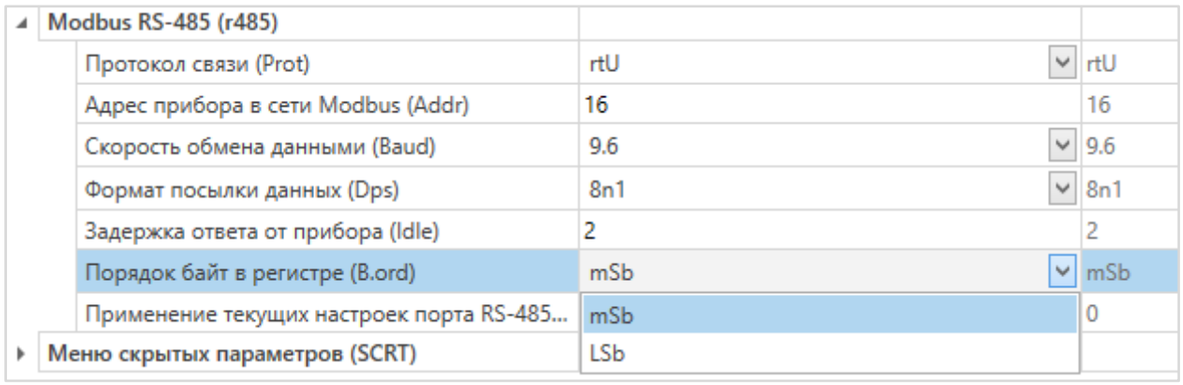
Можно ли было «подстроить» порядок байт и регистров на стороне ТРМ1, чтобы не менять настройки OPC–сервера?

Порядок байт – можно; у прибора есть соответствующий параметр. Порядок регистров – нет.

Это выглядит довольно неудачным решением, потому что порядок байт определён в спецификации Modbus, а порядок регистров – нет. Лучше было бы предоставить пользователю возможность менять как порядок байт, так и порядок регистров.

Modbus RS-485 (r485)		
Протокол связи (Prot)	rtU	rtU
Адрес прибора в сети Modbus (Addr)	16	16
Скорость обмена данными (Baud)	9.6	9.6
Формат посылки данных (Dps)	8n1	8n1
Задержка ответа от прибора (Idle)	2	2
Порядок байт в регистре (B.ord)	mSb	mSb
Применение текущих настроек порта RS-485...	mSb	0
Меню скрытых параметров (SCRT)	LSb	

Внезапная отсылка к флешфорварду



Ещё раз обратите внимание на 2 параметра:

- Порядок байт в регистре;
- Задержка ответа от прибора (Idle).

Параметр **Idle** определяет время задержки, которую ТРМ выдерживает между получения запроса и отправкой ответа. Это задержка позволяет master–устройству успеть переключить свой последовательный порт из режима передачи в режим приёма. У ТРМ201 аналогичный параметр назывался **rSdl**. Мы обсудим его более подробно в уроке 2.11.

Оба этих параметра присутствовали и у датчика ПВТ110–RS, с опроса которого мы начали наш курс.

В его документации они назывались «**Последовательность байт в двухбайтовых данных**» (звучит громоздко, но понятно) и «**Таймаут ответа**» (как мы увидим в уроке 2.11 – это крайне неудачный выбор названия для данного параметра).

Параметры интерфейса						
Последователь-ность байт в двухбайтовых данных	5601	15E1	1	UC8	**11** – старший байт первый 12 – младший байт первый	RW
Сетевой адрес	5602	15E2	1	UC8	1…**16**…99	RO
Скорость обмена (в бодах)	5603	15E3	1	UC8	2 – 2400 3 – 4800 **4 – 9600** 5 – 14400 6 – 19200 7 – 28800 8 – 38400 9 – 56000 10 – 57600 11 – 115200	RO
Количество бит данных	5604	15E4	1	UC8	7/**8**	RO
Контроль чётности	5605	15E5	1	UC8	**0** – нет 1 – чётный 2 – нечётный	RO
Количество стоп-битов	5606	15E6	1	UC8	**0** – 1 1 – 1,5 2 – 2	RW
Таймаут ответа, мс	5607	15E7	2	UC16	1...**100**...1000	RW

Ещё в конце урока 2.1 мы обещали пояснить назначение параметров датчика **Аварийное значение влажности** и **Аварийное значение температуры**.

Их роль заключается в следующем – в случае неисправности измерительного зонда (его обрыва или короткого замыкания) в параметры **Значение влажности** и **Значение температуры** будут подставлены соответствующие аварийные значения; одновременно с этим нулевой бит параметра **Состояние датчика** будет установлен в значение **TRUE (1)**; этот параметр представляет собой битовую маску.

Параметры измерителя						
Верхний предел измерения влажности, %RH	5302	14B6	2	FLOAT32	100	RO
Нижний предел измерения влажности, %RH	5304	14B8	2	FLOAT32	0,00	RO
Постоянная времени фильтра измерения влажности, с	5310	14BE	1	UC8	**0**...100	RW
Аварийное значение влажности, %RH	5313	14C1	2	FLOAT32	-100…**0**…100	RW
Верхний предел измерения температуры, °C	5352	14E8	2	FLOAT32	80,00	RO
Нижний предел измерения температуры, °C	5354	14EA	2	FLOAT32	-40,00	RO
Постоянная времени фильтра температуры, с	5360	14F0	1	UC8	**0**...100	RW
Аварийное значение температуры, °C	5363	14F3	2	FLOAT32	-100…**0**…100	RW

Аварийные значения позволяют избежать ситуации, при которой в случае неисправности измерительного зонда параметры влажности и температуры принимают нулевые значения; при этом в некоторых системах управления в некоторые моменты времени температура действительно может быть нулевой. Чтобы отличить эти две ситуации – можно присвоить аварийному значения температуры заведомо нереалистичное значение (например, –999.0). Это позволит обслуживающему персоналу оперативно определить неисправность датчика, увидев такие показания на экране панели оператора или в SCADA–системе.

2.10 Тестовые задания

Ответы приведены в п. 9.

1. Сопоставьте параметр и его значение.

Максимальный размер пакета Modbus RTU в байтах	8
Максимальное количество считываемых регистров для функций 0x3 и 0x04	123
Максимальное количество записываемых регистров для функции 0x10	125
Количество бит в байте	256

2. Рассмотрим следующий пакет протокола Modbus RTU:

> *4D 04 14 00 02 00 00 04 00 06 00 08 00 0A 00 0A 08 00 06 00 04 00 02 09 DA*

Это запрос или ответ?

- Запрос
- Ответ
- Может быть как запросом, так и ответом

3. Рассмотрим следующий пакет протокола Modbus RTU:

> *4D 04 14 00 02 00 00 04 00 06 00 08 00 0A 00 0A 08 00 06 00 04 00 02 09 DA*

Соотнесите поля пакета и соответствующие им байты.

Адрес slave–устройства	14
Код функции	09 DA
Количество байт данных	04
Данные	4D
Контрольная сумма	00 02 00 00 04 00 06 00 08 00 0A 00 0A 08 00 06 00 04 00 02

4. Рассмотрим следующий пакет протокола Modbus RTU:

> *4D 04 14 00 02 00 00 04 00 06 00 08 00 0A 00 0A 08 00 06 00 04 00 02 09 DA*

Предположим, что первый из считываемых регистров имеет адрес **100** (в DEC).
Введите запрос, на который был отправлен данный ответ (в HEX, с пробелами между байтами).

> ***Примечание**: вы можете использовать Rapid SCADA Modbus Parser для конструирования запроса.*

5. Рассмотрим следующий пакет протокола Modbus RTU:

> *4D 10 00 64 00 01 02 FF FF CB 07*

Предположим, что slave–устройство не поддерживает данную функцию Modbus.
Как будет выглядеть его ответ? (в HEX, с пробелами между байтами).

6. *Какой тип контрольной суммы используется в протоколе Modbus RTU?*
   - *CRC-64*
   - *CRC-8*
   - *CRC-32*
   - *CRC-16*
   - *LRC*

2.11 Настройки COM–порта. Протокол Modbus ASCII

В прошлых уроках мы обсуждали протокол Modbus RTU, в большинстве случаев используемый поверх интерфейса RS–485.

Настало время ознакомиться с конкретными деталями реализации этого интерфейса.

В этом и следующих уроках мы обсудим некоторые особенности RS–485, сфокусировавшись исключительно на тех моментах, представление о которых имеет ценность для инженера, использующего протокол Modbus. Гораздо более исчерпывающую информацию про RS–485, UART и т. д. можно получить из профессиональной литературы.

> *RS–485 (также называемый EIA/TIA–485) – это стандарт физического уровня, определяющий электрические характеристики передатчиков и приёмников, подключаемых к последовательной линии связи; первая версия стандарта появилась в 1983 году. Аббревиатура RS означает Recommended Standard (рекомендуемый стандарт).*

RS–485 тесно связан с понятием UART.

> *UART – узел вычислительных устройств, предназначенный для организации связи с другими цифровыми устройствами. Он выполняет функцию приёмопередатчика.*

В предыдущих уроках мы иногда использовали термин «последовательный порт» или COM–порт.

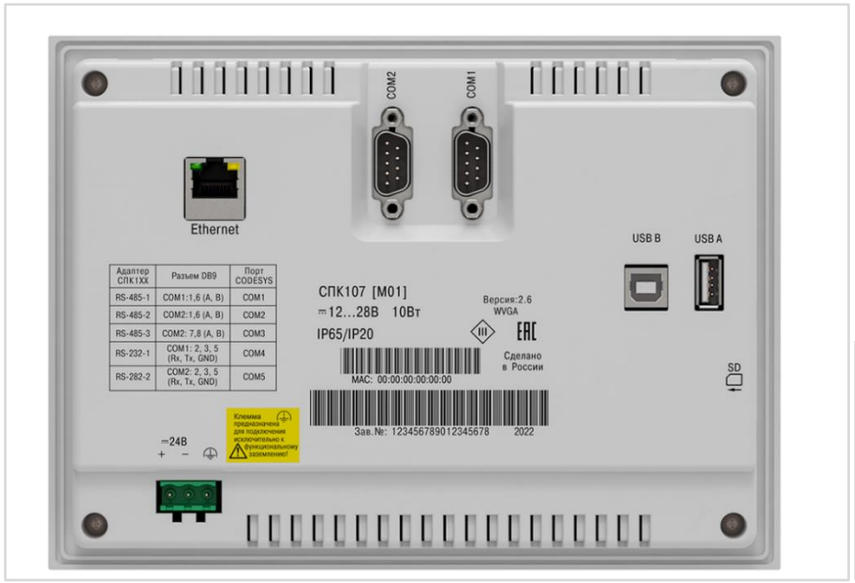
> *COM–порт – это сленговое обозначение, под которым обычно подразумевается последовательный интерфейс устройства.*

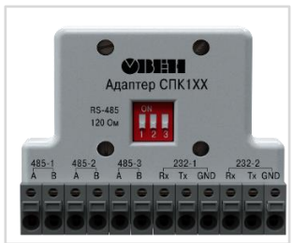
Для RS–485 разъём COM–порта часто представлен в виде клеммной колодки:

Источник: https://wirenboard.com/wiki/RS–485:_Wiring_and_Connection

Но могут использоваться и другие типы разъёмов – DB9, RJ–45 и т. д.

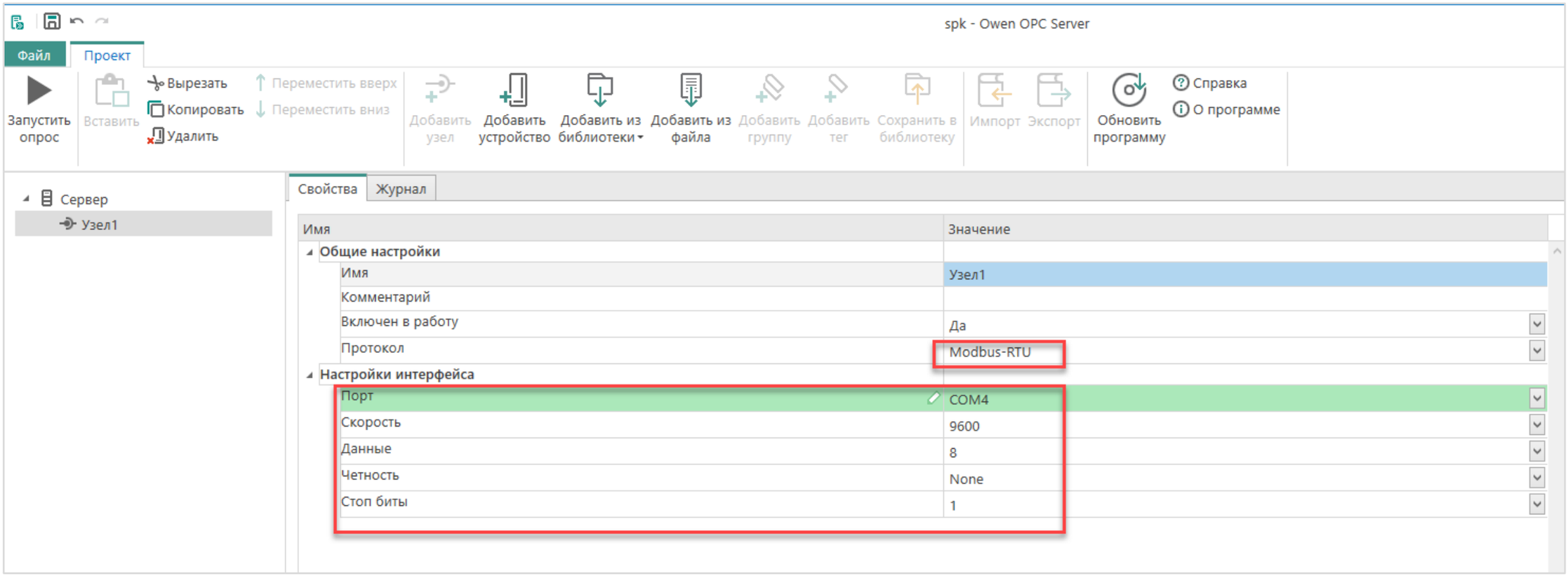
Например, 5 последовательных интерфейсов (3 RS–485, 2 RS–232) сенсорного панельного контроллера СПК1xx [M01] выведены на два разъёма типа DB9. Для удобного подключения к ним проводов используется специальный адаптер.





Терминология UART. Понятие «символа»

Помните, в уроке 2.1 мы начали настройку опроса в OPC–сервере с установки значений параметров COM–порта?



Теперь давайте разберёмся, за что отвечает каждая из них.

Но для этого нам сначала придётся дать определение понятию «**символ**».

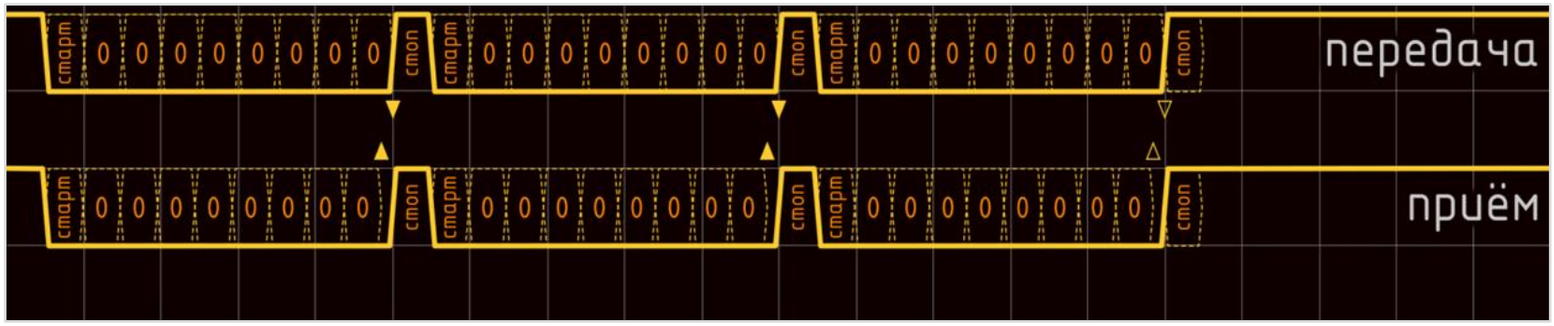
> *Символ (char) – это наименьшая порция данных, которую можно передать через последовательный интерфейс.*

Несмотря на название (появившиеся во времена телетайпа), символ позволяет закодировать не только текстовую, а любую информацию.

Символ включает в себя:

- старт–бит, позволяющий определить начало символа в потоке бит;
- стоп–бит, позволяющий разграничить символы в потоке бит;
- биты данных (обычно от 5 до 8);
- и, опционально, бит чётности, позволяющий проверить целостность символа.

> *См. более подробное изложение в статье Неизвестный UART: теория на портале Habr (ссылка на перепост).*



Источник: https://habr.com/ru/articles/708902/

Настройки COM–порта: скорость обмена

> *Скорость обмена (baudrate) – это полное число бит, которое передаётся по последовательному интерфейсу за единицу времени.*

Под «полнотой» подразумевается, что при расчёте скорости учитываются не только биты данных, но и служебные биты (старт–бит, стоп–бит, бит чётности).

В качестве единицы измерения скорости обычно используется бит/с. Иногда используется термин «бод», который для рассматриваемых в курсе ситуацией эквивалентен бит/с.

Типовой для области АСУ (автоматизированных систем управления) набор скоростей:

> *4800, 9600, 19200, 38400, 57600, 115200*

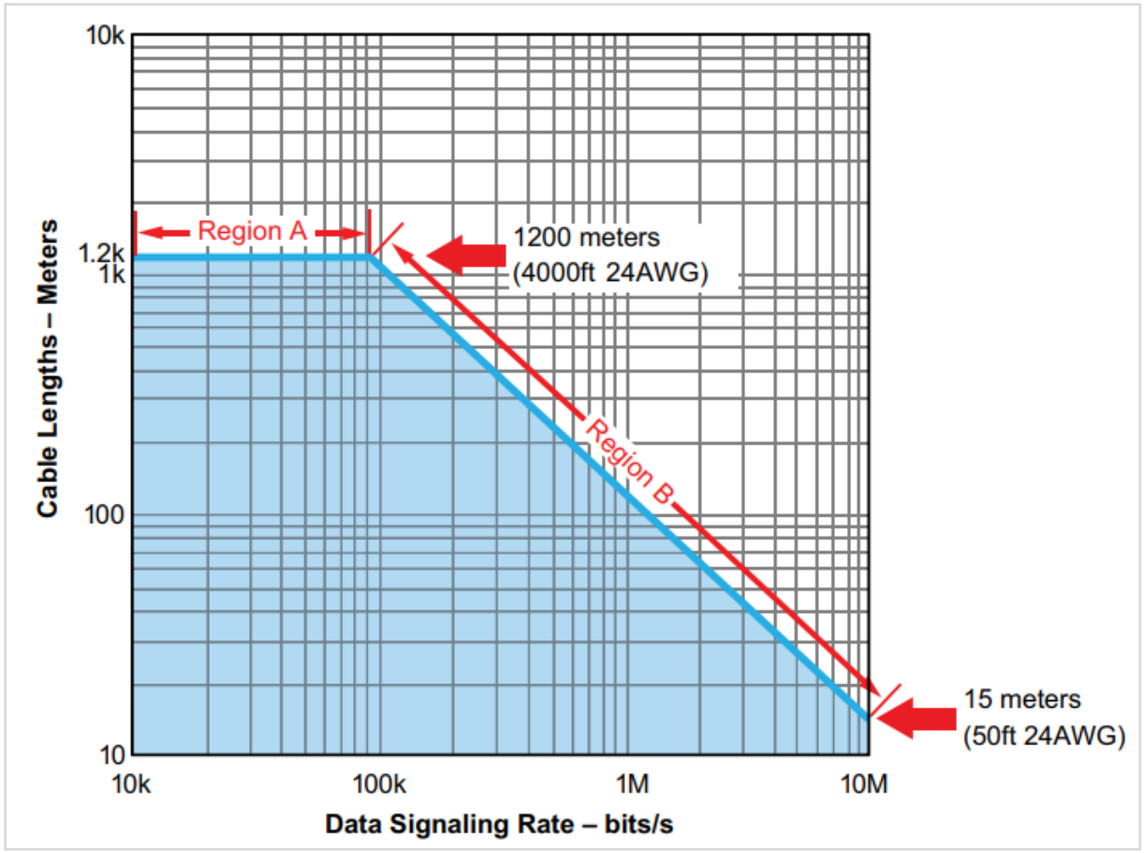
Некоторые устройства могут поддерживать и другие, специфичные скорости – например, 14400.

Спецификация Modbus определяет [1, п. 3.2], что устройство должно поддерживать как минимум две скорости – **9600** и **19200**, при этом **19200** должно использоваться как значение по умолчанию. На практике производители оборудования часто отступают от этого требования, устанавливая для своих устройств по умолчанию скорость **9600** (это характерно для большинства приборов компании ОВЕН) или **115200**.

Для обеспечения корректной передачи и приёма данных все устройства конкретной последовательной линии связи должны использовать одну и ту же скорость.

Выбранная скорость влияет на максимально возможную длину линии связи; в даташитах (datasheet – техническая спецификация) производителей UART–микросхем обычно приводятся соответствующие графики. В существенном количестве случаев для типового в области АСУ набора скоростей (4800...115200) длина линии связи может достигать 1200 метров.

> *Скорость обмена также влияет на степень помехоустойчивости линии связи – чем выше скорость, тем ниже помехоустойчивость. Это связано с тем, что на высоких скоростях обмена длительность электрических импульсов, кодирующих биты, становится очень короткой и при воздействии помехи (которая обычно бывает кратковременной) она полностью «испортит» очередной передаваемый бит. На низких скоростях длительность импульсов больше, и время действия помехи может быть меньше длительности передачи бита; так как в микросхемах UART измерение параметров электрического сигнала, соответствующего передаваемому биту, обычно производится несколько раз (oversampling), то в этом случае есть вероятность сохранить его исходное значение.*



Источник: https://www.maxlinear.com/appnote/an–292.pdf

Настройки COM–порта: количество бит данных

Согласно спецификации Modbus Serial [1, п. 2.5]:

- для протокола **Modbus RTU** в состав символа входит **8 бит данных**;
- для протокола **Modbus ASCII** в состав символа входит **7 бит данных**.

Мы обсудим протокол Modbus ASCII далее в ходе данного урока.

В редких случаях вы можете встретить устройство, которое будет отступать от второго требования (но первое требование выполняется всегда).

Например, в индикаторе СМИ2–М для протокола Modbus ASCII всегда используется 8 бит данных; старший бит является незначимым. Это связано с ограничением аппаратной платформы прибора.

Настройки COM–порта: режим контроля чётности (паритета)

Контроль чётности позволяет определить целостность символа. Под «целостностью» подразумевается, что принятый символ соответствует отправленному. Нарушить целостность, например, может помеха, действующая на линию связи.

Для обеспечения возможности проверки на целостность используется бит чётности.

Существуют три основных режима контроля чётности:

- **NONE** – контроль чётности не производится; бит чётности не проверяется;
- **EVEN** – проверка на чётность; бит чётности при отправке символа выставляется в такое значение, чтобы общее число «1» в битах данных и бите чётности было чётным;
- **ODD** – проверка на нечётность; бит чётности при отправке символа выставляется в такое значение, чтобы общее число «1» в битах данных и бите чётности было нечётным.

Проверка на чётность позволяет детектировать лишь единичные ошибки, которые меняют значение чётного количества битов символа. Эта проверка является вырожденным вариантом циклической контрольной суммы (CRC); если контрольная сумма Modbus позволяет обнаружить ошибку на уровне пакета, то контроль чётности позволяет обнаружить ошибку на уровне отдельного символа. В большинстве реализаций такой повреждённый символ отбрасывается приёмником; это приводит к тому, что при анализе пакета slave–устройство определит, что этот пакет является некорректным и не будет его обрабатывать.

Существуют ещё два специфических варианта воздействия на бит чётности:

- **MARK** – бит чётности всегда устанавливается в «1»;
- **SPACE** – бит чётности всегда устанавливается в «0».

Эти варианты используются в некоторых специфических протоколах обмена для передачи дополнительной информации. См. более подробную информацию, например, здесь: https://stackoverflow.com/questions/13953095/what–is–the–difference–between–using–mark–space–parity–and–parity–none

Настройки COM–порта: количество стоп–бит

Параметр **Количество стоп–бит** определяет интервал времени, разделяющий символы в потоке бит.

Параметр может иметь значение **1** или **2.**

Иногда можно встретить вариант **1.5 стоп–бита**, который может показаться странным с «программной» точки зрения – ведь бит по определению является неделимой величиной. Но с «электрической» точки зрения это значение вполне корректно – речь идёт об интервале времени, которое требуется на передачу 1.5 бит на заданной скорости обмена.

Этот вариант крайне редко встречается в современном оборудовании.

Сокращённая форма записи настроек COM–порта

Часто настройки COM–порта записываются в сокращённой форме, например:

- 9600–8E1;
- 115200–8N2;
- и т. д.

Первой величиной является скорость, дальше записывается количество бит данных, первая буква режима контроля чётности (N, E или O) и количество стоп–бит.

С точки зрения спецификации Modbus Serial [1, п. 2.5]:

- для протокола Modbus RTU размер символа = 11 бит (1 старт–бит, 8 бит данных, 1 бит контроля чётности, 1 стоп–бит);
  Возможные сочетания настроек: 8E1, 8O1, 8N2.
  Для сочетания 8N2 бит контроля чётности заменяется вторым стоп–битом.
  Спецификация требует поддержки режима контроля чётности EVEN и его использования в качестве значения по умолчанию; поддержка других режимов является опциональной (рекомендуется поддержать NONE).
  Это требование выполняется не всегда. Например, для большинства приборов компании ОВЕН по умолчанию используется сочетание настроек 8N1. У некоторых приборов вы можете встретить доступные сочетания 8E2 и 8O2; как именно они реализованы в конкретном приборе – вопрос к его производителю.

> **The format (11 bits) for each byte in RTU mode is :**
>
> | | |
> |---|---|
> | **Coding System:** | 8–bit binary |
> | **Bits per Byte:** | 1 start bit |
> | | 8 data bits, least significant bit sent first |
> | | 1 bit for parity completion |
> | | 1 stop bit |
>
> **Even parity is required,** other modes (odd parity, no parity) may also be used. In order to ensure a maximum compatibility with other products, it is recommended to support also No parity mode. The default parity mode must be even parity.
>
> Remark : the use of no parity requires 2 stop bits.

- для протокола Modbus ASCII размер символа = 10 бит (1 старт–бит, 7 бит данных, 1 бит контроля чётности, 1 стоп–бит);
  Возможные сочетания настроек: 7E1, 7O1, 7N2.
  Для сочетания 7N2 бит контроля чётности заменяется вторым стоп–битом. Спецификация требует поддержки режима контроля чётности EVEN и его использования в качестве значения по умолчанию; поддержка других режимов является опциональной (рекомендуется поддержать NONE). У некоторых приборов вы можете встретить доступные сочетания 7E2 и 7O2; как именно они реализованы в конкретном приборе – вопрос к его производителю.

The format (10 bits) for each byte in ASCII mode is :

Coding System:	Hexadecimal, ASCII characters 0–9, A–F
	One hexadecimal character contains 4-bits of data within each ASCII character of the message
Bits per Byte:	1 start bit
	7 data bits, least significant bit sent first
	1 bit for parity completion;
	1 stop bit

Even parity is required, other modes (odd parity, no parity) may also be used. In order to ensure a maximum compatibility with other products, it is recommended to support also No parity mode. The default parity mode must be Even parity.

Remark : the use of no parity requires 2 stop bits.

***

Для обеспечения возможности обмена данными между устройствами, подключенными к одной и той же последовательной линии связи, настройки их COM–портов должны совпадать.

Тем не менее, возможны частные случаи, в которых обмен будет работать даже при некоторых отличиях в настройках.

Например, некоторое современное ПО игнорирует настройку **Количество стоп–бит**. Это легко проверить в рамках проводимых ранее практических опытов с OPC–серверами – если вы установите в их настройках не то количество стоп–бит, которое фактически выбрано в настройках прибора, то обмен всё равно будет работать корректно.

Но скорость и количество бит данных всегда должны совпадать.

Пакет Modbus RTU

Используя изученную в прошлых шагах информацию – мы можем сформулировать следующее определение:

> *Пакет (фрейм, frame) – это осмысленный (с точки зрения спецификации Modbus) набор символов.*

При приёме данных устройство должно определять:

- признак конца символа, чтобы отличить его от других символов пакета;
- признак конца пакета, чтобы определить момент, когда следует приступать к его анализу.

В спецификации Modbus Serial [1, п. 2.5.1.1] этим признакам соответствуют таймауты (интервалы тишины) на линии связи, обозначаемые как **t(1.5)** и **t(3.5)**.

> ***t(1.5)** – это признак конца символа (inter–char delay).*

Он равен интервалу времени, которое требуется на передачу 1.5 символов на заданной скорости.

Но для скоростей > 19200 бит/с его значение принимается постоянным и составляет 0.750 мс.

Если в течение этого времени приёмник не получает никаких бит – то считается, что приём текущего символа завершён.

> ***t(3.5)** – это признак конца пакета (inter–frame delay).*

Он равен интервалу времени, которое требуется на передачу 3.5 символов на заданной скорости.

Но для скоростей > 19200 бит/с его значение принимается постоянным и составляет 1.750 мс.

Если в течение этого времени приёмник не получает никаких символов – то считается, что приём текущего пакета завершён.

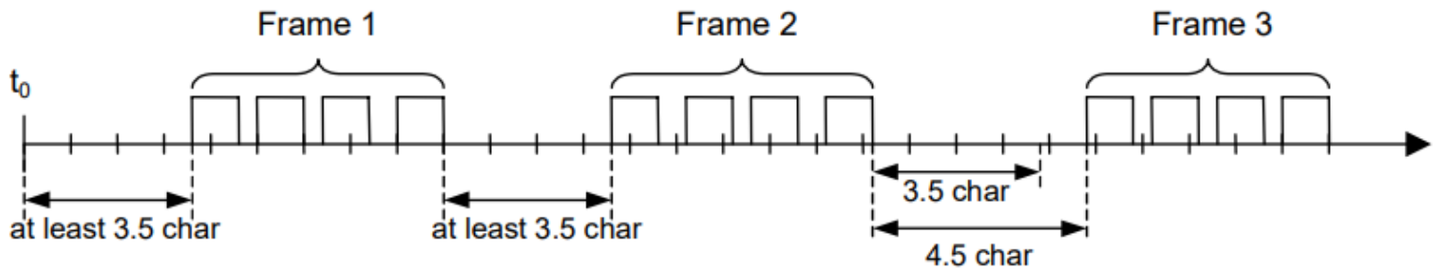
Для отсчёта этих интервалов – устройство должно иметь на борту достаточно точные аппаратные таймеры.

В целом, механизм t(3.5) в настоящее время часто считается устаревшим, и для определения конца пакета могут использоваться другие принципы. Подробности рассмотрены в п. 2.1 статьи Заметки о Modbus.

2.5.1.1 MODBUS Message RTU Framing

A MODBUS message is placed by the transmitting device into a frame that has a known beginning and ending point. This allows devices that receive a new frame to begin at the start of the message, and to know when the message is completed. Partial messages must be detected and errors must be set as a result.

In RTU mode, message frames are separated by a silent interval of at least 3.5 character times. In the following sections, this time interval is called t3,5.



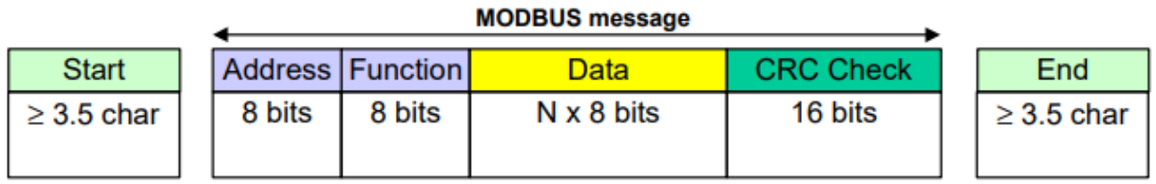
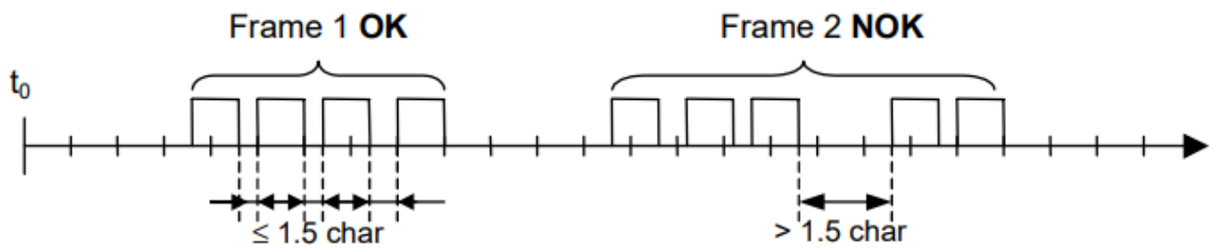


Figure 13: RTU Message Frame

The entire message frame must be transmitted as a continuous stream of characters.

If a silent interval of more than 1.5 character times occurs between two characters, the message frame is declared incomplete and should be discarded by the receiver.



Remark :

The implementation of RTU reception driver may imply the management of a lot of interruptions due to the $t_{1.5}$ and $t_{3.5}$ timers. With high communication baud rates, this leads to a heavy CPU load. Consequently these two timers must be strictly respected when the baud rate is equal or lower than 19200 Bps. For baud rates greater than 19200 Bps, fixed values for the 2 timers should be used: it is recommended to use a value of 750µs for the inter-character time-out ($t_{1.5}$) and a value of 1.750ms for inter-frame delay ($t_{3.5}$).

Протокол Modbus ASCII (часть 1)

В прошлом шаге мы упомянули, что признаком конца пакета протокола Modbus RTU является интервал тишины **t(3.5)**, для возможности определения которого требуется наличие у устройства достаточно точного аппаратного таймера; в целом, процедура определения конца пакета является не самой простой (*если только микроконтроллер не предоставляет соответствующее прерывание*).

Этой сложности лишён протокол Modbus ASCII – второй (и последний) протокол, описанный в спецификации Modbus Serial.

Его основным отличием от Modbus RTU является принцип кодирования данных: если в Modbus RTU по линии связи передаются бинарные данные, то в Modbus ASCII передаются коды символов кодировки ASCII. Эта кодировка является 7–битной – и именно поэтому для протокола Modbus ASCII достаточно использовать 7 бит данных, как мы упоминали пару шагов назад.

Структура пакета Modbus ASCII в целом совпадает со структурой пакета Modbus RTU и имеет следующие отличия [1, п. 2.5.2.1]:

- пакет начинается со старт–символа (:) и заканчивается двумя стоп–символами <CR><LF>;
- контрольная сумма: вместо 2–байтовой CRC16 используется 1–байтовая LRC.

Давайте рассмотрим следующий запрос Modbus RTU (вы уже должны понимать, за что отвечает каждый из его байтов):

> ***10 03 10 09 00 02*** *13 88*

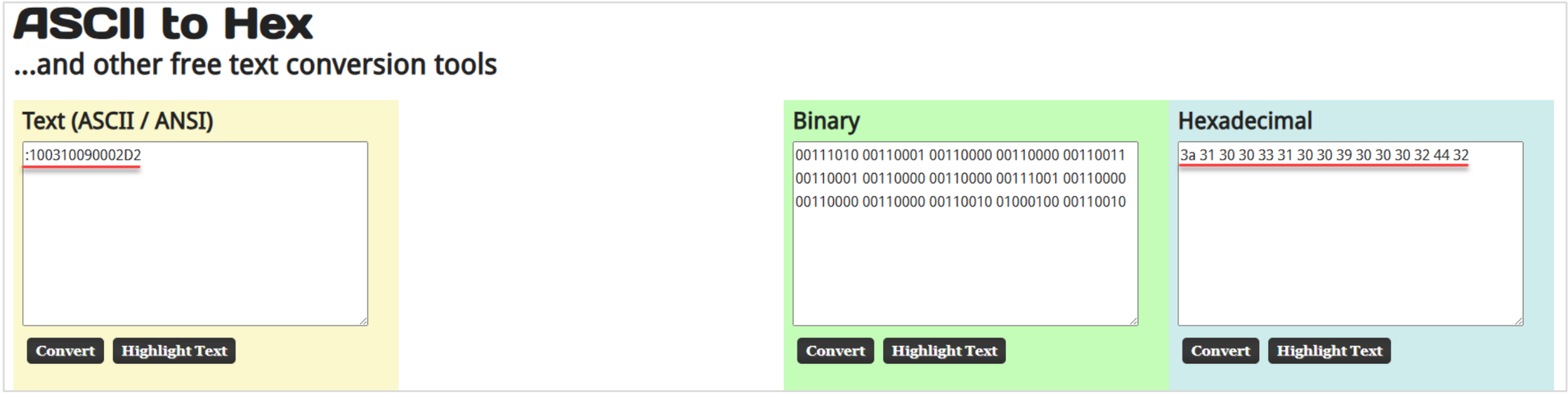
Как и всегда, мы используем бинарную форму записи с шестнадцатеричной системой счисления (HEX). Это именно тот набор байт (и бит, из которых они состоят), который будет передан по линии связи.

Для идентичного пакета Modbus ASCII **текстовая** форма записи будет иметь следующий вид:

> *:**100310090002**D2<CR><LF>*

А по линии связи, как уже упоминалось, будут переданы ASCII–коды этой строки. Поэтому в бинарной форме записи пакет будет выглядеть так:

> *3A 31 30 30 33 31 30 30 39 30 30 30 32 44 32 0D 0A*



Конвертер: https://www.asciitohex.com/

<CR><LF> – это символы переноса строки и возврата каретки телетайпа, по историческим причинам использующиеся и в современно ПО как спецсимволы переноса строк в тексте. Им соответствуют ASCII–коды 0x0D и 0x0A.

> *Подробнее о контрольной сумме (LRC) протокола Modbus ASCII см. в п. 3.3 статьи Заметки о Modbus.*

Протокол Modbus ASCII (часть 2)

С точки зрения спецификации Modbus Serial – поддержка Modbus–совместимым устройством протокола Modbus ASCII является опциональной, в то время как поддержка Modbus RTU является обязательной [1, п. 2.5].

Тем не менее, в некоторых случаях вы можете встретить приборы, которые поддерживают только Modbus ASCII.

Примером такого устройства является тахометр Мерадат–М12ТХГ1 от компании Термодат:

https://www.termodat.ru/catalog/reka_more/m12thg1/

***

У протокола Modbus ASCII есть два преимущества перед Modbus RTU:

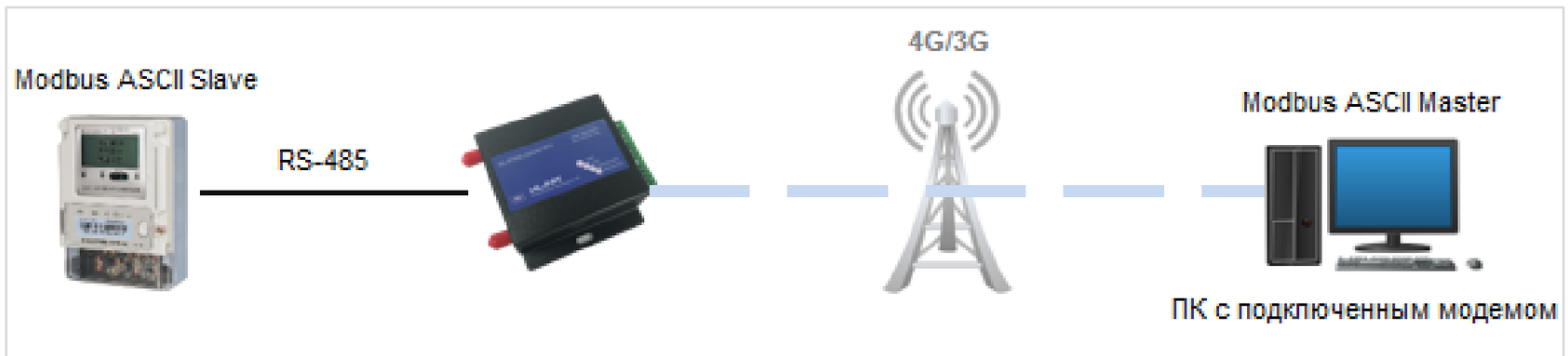
1. Начало и конец пакета определяются по старт- и стоп-символам.

Не требуется измерять интервал тишины t(3.5) – соответственно, не требуется наличие у устройства аппаратных таймеров.

2. Из п. 1 следует, что пакет Modbus ASCII может передаваться с паузами практически произвольной длительности.

В протоколе Modbus RTU задержка в процессе передачи пакета в нескольких миллисекунд приведёт к срабатыванию t(3.5) – и, соответственно, определению признака конца пакета и переходу к его анализу, после которого пакет будет отброшен как некорректный – ведь из–за задержки получены не все его байты.

Но в процессе передачи пакетов по некоторым каналам связи (2G, радиоканал и т. д.) могут возникать задержки, исчисляемые секундами. Соответственно, применить протокол Modbus RTU в данном случае просто не получится – потребуется использовать Modbus ASCII.



Источник (используется отредактированная версия изображения): https://www.wlink–tech.com/art/modbus–in–router

***

Надо отметить, что размер пакета Modbus ASCII будет приблизительно в 2 раза больше по сравнению с идентичным пакетом Modbus RTU. Это связано с тем, что для кодирования одного байта требуется использовать два ASCII–кода (например, для байта 0xFE нам надо передать по линии связи ASCII–код символа «F» и ASCII–код символа «E»).

В некоторых неприятных случаях вы можете встретиться с устройствами, которые «нестабильно» опрашиваются по протоколу Modbus RTU, но без проблем – по протоколу Modbus ASCII. Одним из представителей подобных устройств является линейка терморегуляторов ТРМ2xx, в настоящий момент снятая с продажи; мы упомянули это в уроке 2.6.

Подведение итогов

Мы рассмотрели назначение каждой из настроек COM–порта.

Основное, что в связи с этим следует запомнить – для обеспечения возможности обмена данными между устройствами, подключенными к одной и той же линии связи, настройки их COM–портов должны совпадать.

Мы также обсудили, как происходит определение признака конца символа **t(1.5)** и конца пакета **t(3.5)** в протоколе Modbus RTU, и рассмотрели протокол Modbus ASCII. Его основные отличия от Modbus RTU:

- используется другой принцип кодирования данных – передаются ASCII–коды символов, а не их «байтовые» значения;
- размер пакета Modbus ASCII приблизительно в 2 раза больше, чем у аналогичного пакета Modbus RTU;
- структура пакета Modbus ASCII в целом совпадает с Modbus RTU, но дополнительно используются стартовый и два стоповых символа. Отличается принцип расчёта контрольной суммы;
- конец пакета Modbus ASCII определяется по его стоповым символам, что позволяет использовать этот протокол совместно с каналами связи, подверженными задержкам (2G, радиоканал и т. д.).

В следующих уроках мы рассмотрим другие интервалы времени, важные для протокола Modbus, после чего углубимся в изучение интерфейса RS–485.

2.12 Основы Modbus, часть 3

В прошлом уроке мы узнали про два важных для протокола Modbus RTU параметра – **t(1.5)** и **t(3.5)**.

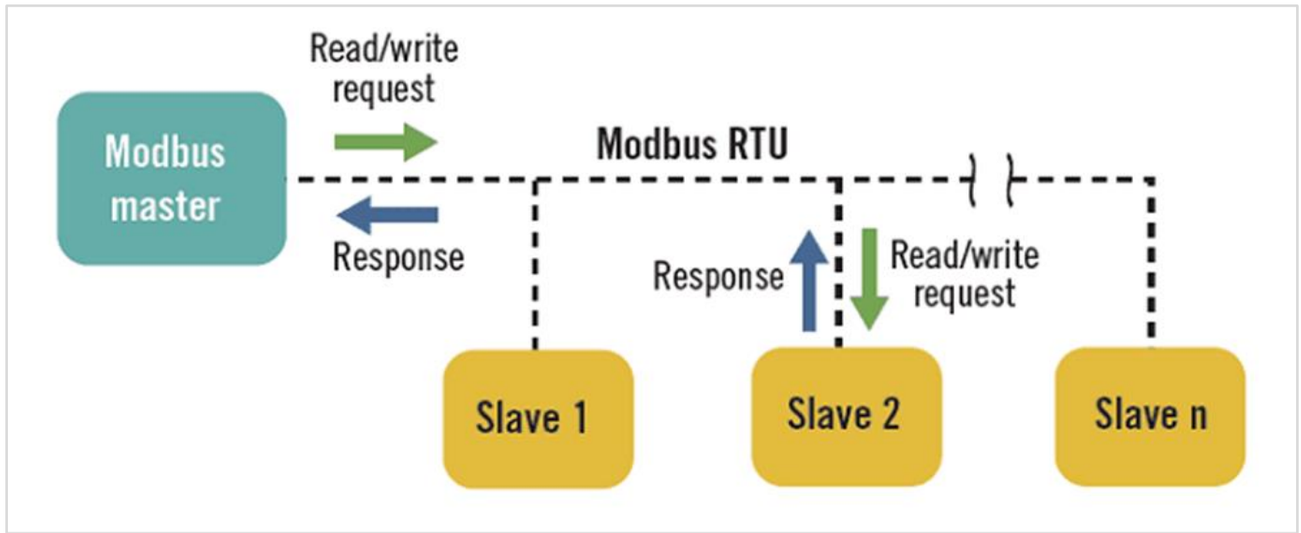
Давайте теперь обсудим другие «параметры времени», которые важны для как для протокола Modbus RTU, так и для Modbus ASCII.

Но для начала мысленно вернёмся в начало урока 2.5 и вспомним, что протокол Modbus основан на архитектуре **Master/Slave**.

В рамках этой архитектуры существуют две роли (режима работы):

- master – инициатор опроса. Он отправляет запрос на чтение или запись данных в slave–устройство;
- slave – обработчик запросов. Он ожидает запроса от мастера и, при его получении, выполняет заданную операцию и отправляет ответ.

Принцип работы устройств с протоколом Modbus Serial можно продемонстрировать следующей схемой:



Источник: https://www.controlglobal.com/network/industrial–networks/article/11304326/introduction–to–modbus

- master–устройство отправляет запрос (request) одному из slave–устройств;
- slave–устройство обрабатывает этот запрос и отправляет ответ (response);
- master–устройство получает этот ответ и отправляет следующий запрос (либо этому же slave–устройству, либо другому);
- slave–устройство обрабатывает этот запрос и отправляет ответ;
- ...
- и так «по кругу».

Для Modbus Serial существует **важное ограничение** [1, п. 1.2]:

- в пределах одной конкретной последовательной линии связи (её часто называют «шиной») может функционировать только одно устройство, выполняющее роль Master.

Параметры master–устройства

Период опроса (polling time)

- Период опроса – это интервал времени, с которым происходит отправка заданного Modbus–запроса. Обычно в настройках master–устройства можно задать произвольный период опроса, но он может быть недостижим в реальности – например, если настроено 100 запросов и для каждого из них указан период опроса = 100 мс; одной миллисекунды не хватит на отправку запроса и получение ответа, поэтому реальный период опроса будет больше, чем заданный. Поэтому при настройке периода опроса следует принимать во внимание всю конфигурацию обмена и особенности опрашиваемых slave–устройств. Мы обсудим связанные с этим вопросы в следующих шагах.

- Некоторые устройства позволяют отправлять запросы не только периодически, но и по команде (по событию, формируемому в логике устройства). Частным случаем команды является «запись по изменению», при которой отправка нового запроса записи происходит в момент изменения значения; например, именно так реализована запись параметров в OPC–серверах.

Таймаут ответа (response timeout)

- После отправки запроса master–устройство ждёт ответа от slave–устройства. Но нет гарантий, что ответ придёт – например, slave–устройство может быть выключено, неисправно или находиться в зависшем состоянии; линия связи с ним может быть повреждена.

- Поэтому master–устройство ожидает ответ не бесконечно, а в течение заданного интервала времени, называемого **таймаутом ответа**. Его значение должно устанавливаться в зависимости от скорости обмена и особенностей slave–устройства. Некоторые slave–устройства отвечают практически мгновенно, и для них будет адекватным значение таймаута из диапазона 20...50 мс. Другие slave–устройства отвечают с задержками, которые могут измеряться сотнями миллисекунд или даже единицами секунд. Задержка ответа не всегда является постоянной величиной; например, обычно slave–устройство может отвечать «быстро», но в конкретные моменты времени в нём могут выполняться какие–то фоновые операции (например, сброс буферов во flash–память), и если запрос от master–устройства поступит в этот момент – то ответ может быть отправлен спустя несколько секунд.

- Спецификация Modbus Serial [1, п. 2.4.1] указывает, что для скорости **9600** типичным значением таймаута является **1 секунда**.

- Обычно в master–устройстве присутствует параметр **Количество повторных запросов в случае отсутствия ответа**. В этом случае после срабатывания таймаута master–устройство повторно отправляет тот же самый запрос, и делает это N раз (часто в качестве N по умолчанию используется значение **3**); если на один из этих повторных запросов не было получено ответа – то происходит отправка следующего запроса. Повторные запросы позволяют справляться с единичными отсутствиями ответов, связанных с воздействием кратковременной помехи на линию связи.

Пауза между запросами (turnaround delay, framing time)

- Если master–устройство получило ответ от slave–устройства – то оно должно отправить следующий запрос. Но если сделать это «мгновенно», то существует вероятность, что COM–порт slave–устройства не успеет переключиться из режима передачи в режим приёма. Чтобы избежать этого – используется параметр **Пауза между запросами**, который в спецификации Modbus называется **turnaround delay** [1, п. 2.4.1].

- Turnaround delay – это пауза, которую master–устройство выдерживает между получением ответа от slave–устройства и отправкой следующего запроса.

- Спецификация Modbus указывает, что типичный диапазон этой задержки – 100...200 мс.

- Для большинства современных устройств будет достаточным значение в диапазоне 10...20 мс.

- Но существуют исключения; например индикатор СМИ2 (в настоящий момент снятый с продажи) после отправки ответа master–устройству «удерживал» линию связи ещё в течение приблизительно 40 мс. Соответственно, для корректного обмена с ним в настройках master–устройства требовалось установить паузу между запросами = 50 мс (с некоторым запасом).

- Ниже приведен скриншот обсуждаемых нами настроек в интерфейсе Owen OPC Server:

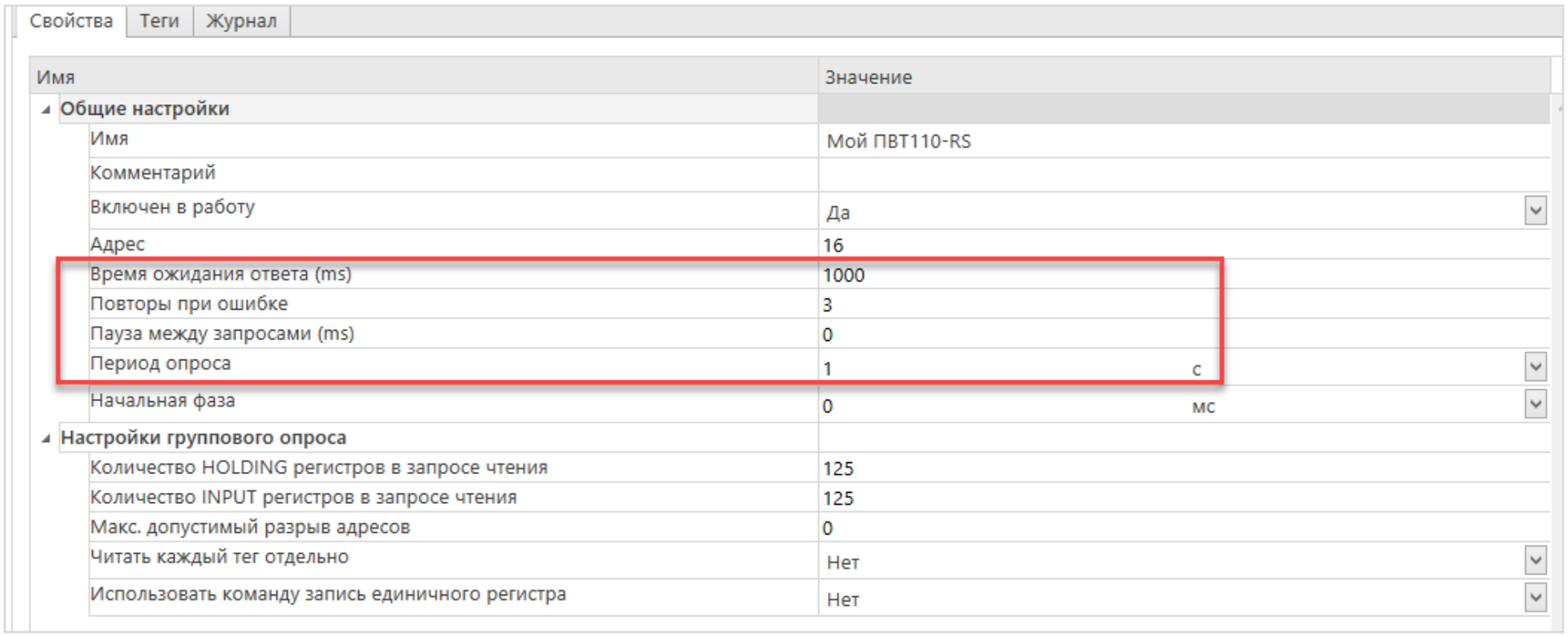
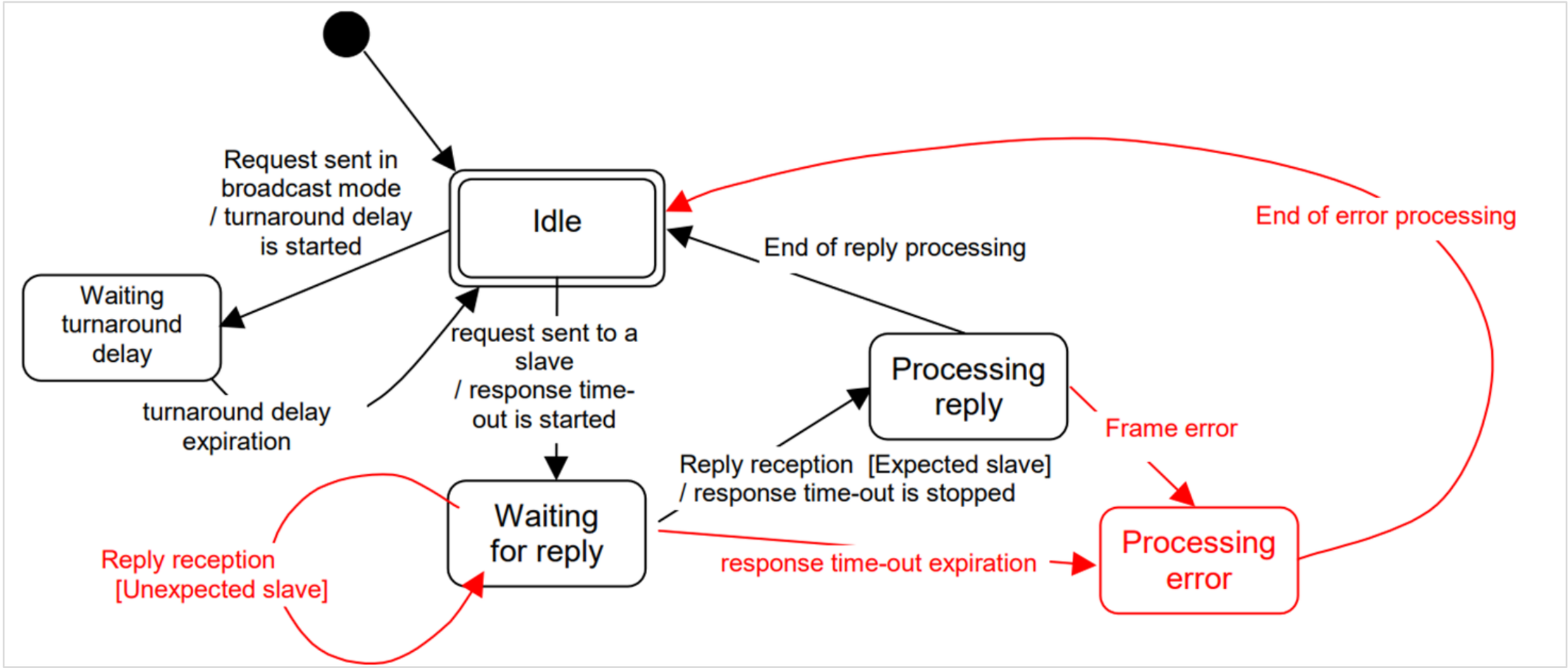


Диаграмма состояний master–устройства

В спецификации Modbus Serial приводится диаграмма состояний master–устройства, на которой отражено почти всё, что мы изучили к текущему моменту [1, п. 2.4.1]:



Изначально master–устройство находится в состоянии **Idle** (ожидание). В нужный момент (определяемый периодом опроса) происходит отправка очередного запроса и переход в состояние **Waiting for reply** (ожидание ответа). Одновременно с этим запускается таймер таймаута.

В состоянии ожидания ответа возможны 3 ситуации:

1. Master–устройство получило ответ, в котором адрес slave–устройства не соответствует тому, который был указан в запросе.

Такое, например, может произойти, если slave–устройство долго не отвечало, сработал таймаут, master–устройство отправило запрос следующему slave–устройству, и тут первое устройство ответило.

Такой ответ игнорируется. Master–устройство остаётся в состоянии **Waiting for reply**.

2. За время таймаута ответ от slave–устройства так и не поступил. Происходит переход в состояние **Processing error** (обработка ошибки).

3. Master–устройство получило ответ, в котором адрес slave–устройства соответствует тому, который был указан в запросе. Таймер таймаута останавливается, и происходит переход в состояние **Processing reply** (обработка ответа).

В процессе обработки ответа могут быть обнаружены ошибки – например, некорректная структура пакета, несоответствие выделенной из ответа и рассчитанной контрольной суммы и т. д. В этом случае ответ считается некорректным, и происходит переход в состояние **Processing error** (обработка ошибки).

Если же ответ корректен – то выполняется соответствующая операция, например:

- если это ответ на запрос чтения – то выделенные из запроса данные копируются в нужную область памяти (например, в случае ПЛК – в область переменных приложения);
- если это ответ с кодом ошибки Modbus (например, **0x01 ILLEGAL FUNCTION**) – то формируется диагностическое сообщение.

После этого происходит возврат в состояние **Idle**.

Из состояния **Processing error** тоже происходит переход в состояние **Idle** – просто в этом случае не произойдет никаких дополнительных действий (хотя, возможно, факт получения «битого» пакета тоже будет записан в сообщения диагностики).

В состоянии **Idle** перед отправкой следующего запроса выдерживается пауза, определяемая значением параметра **turnaround delay**. Момент отправки следующего запроса зависит от настроек периода опроса; кроме того, если на текущий запрос не был получен ответ, то спустя turnaround delay этот запрос может быть отправлен повторно – это зависит от значения параметра **Количество повторных запросов**.

Параметры slave–устройства

Время на обработку запроса

Это время между получением запроса от master–устройства и отправкой ему ответа.

Оно является специфичным для конкретного slave–устройства; обычно повлиять на него крайне затруднительно или вовсе невозможно.

Задержка ответа (response delay)

Задержка ответа – это дополнительная задержка, которую slave–устройство выдерживает перед отправкой ответа master–устройству. Она не описана в спецификации Modbus, но присутствует у разумно спроектированных устройств. У всех рассмотренных нами ранее приборов (ПВТ110–RS, ТРМ201, ТРМ1) такой параметр присутствует.

Задержка ответа по своему смыслу соответствует паузе между запросами (turnaround delay) master–устройства; за этот интервал времени COM–порт master–устройства должен успеть переключиться из режима передачи в режим приёма.

В большинстве приборов ОВЕН для этого параметра по умолчанию установлено значение **2 мс**; этого хватает для большинства современных устройств. Для устройств с «заторможенными» COM–портами может потребоваться увеличить значение задержки до 10 мс и даже более высоких значений.

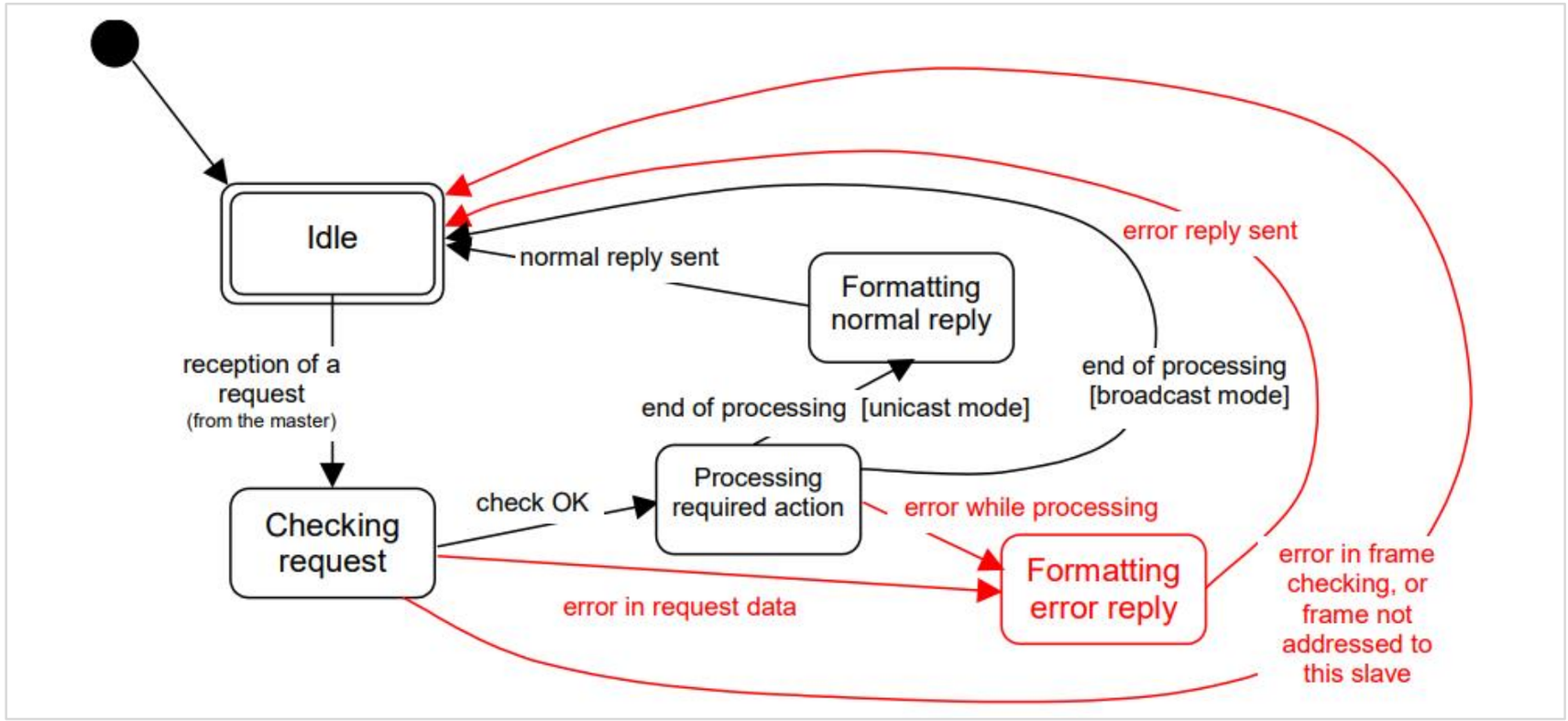
Таймаут безопасного состояния

Master–устройство достаточно легко может определить отсутствие связи с slave–устройством – по отсутствию от него ответа в течение интервала времени, определяемого таймаутом.

Но для некоторых slave–устройств тоже полезно определить отсутствия связи с master'ом – например, чтобы переключиться из автоматического режима в режим ручного управления или изменить состояние своих выходов. Собственно, наибольшее значение это имеет как раз для устройств с дискретными или аналоговыми выходами, к которым подключены исполнительные механизмы – в случае обрыва линии связи с master–устройством (которое обычно выполняет алгоритмы управления системой) или его выхода из строя требуется перевести их в «безопасное» (с точки зрения конкретной системы) состояние (например, отключить).

Этот параметр не описан в спецификации Modbus, но присутствует у разумно спроектированных устройств.

Диаграмма состояний slave–устройства



В начальный момент времени устройство находится в состоянии **Idle** (ожидание).

После получения запроса от master–устройства происходит переход в состояние **Checking request** (проверка запроса).

В результате проверки возможны 3 ситуации:

1. Запрос является корректным и может быть обработан. Происходит переход в состояние **Processing required action** для его выполнения.

2. Запрос является корректным, но не может быть обработан – например, используемая в нём функция не поддерживается slave–устройством или происходит доступ к несуществующему регистру. Происходит переход в состояние **Formatting error reply** для подготовки ответа с кодом ошибки Modbus.

3. Запрос не предназначен для данного slave–устройства (в нём используется другой адрес устройства) или является некорректным – например, в нём присутствует ошибка контрольной суммы или структура запроса в целом не соответствует спецификации Modbus. Происходит возврат в состояние **Idle**; никакого ответа master–устройству в данном случае не отправляется.

Если процесс выполнения запроса завершается без ошибок – то происходит переход в состояние **Formatting normal reply** для подготовки ответа master–устройству, его отправка и возврат в состояние **Idle**.

Если в процессе выполнения корректного запроса (см. п. 1) возникает какая–то ошибка (например, при подготовке значений запрошенных регистров происходит ошибка доступа к памяти) – то происходит переход в состояние **Formatting error reply** для подготовки ответа с кодом ошибки Modbus (обычно используется код **0x04 SLAVE DEVICE FAILURE**). Этот ответ отправляется master–устройству, после чего происходит возврат в состояние **Idle**.

Обобщенная диаграмма состояний (Modbus RTU, Modbus ASCII)

В спецификации Modbus Serial [1, п. 2.5.1.1, п. 2.5.2.1] приведены и более детализированные диаграммы состояний, на которых отражены особенности протоколов Modbus RTU и Modbus ASCII. Эти диаграммы являются универсальными для режимов master и slave.

Например, для Modbus RTU на этой диаграмме отражены интервалы тишины **t(1.5)** и **t(3.5)**, используемые для определения конца символа и конца пакета соответственно, а для Modbus ASCII – процесс анализа стартового и стопового символов.

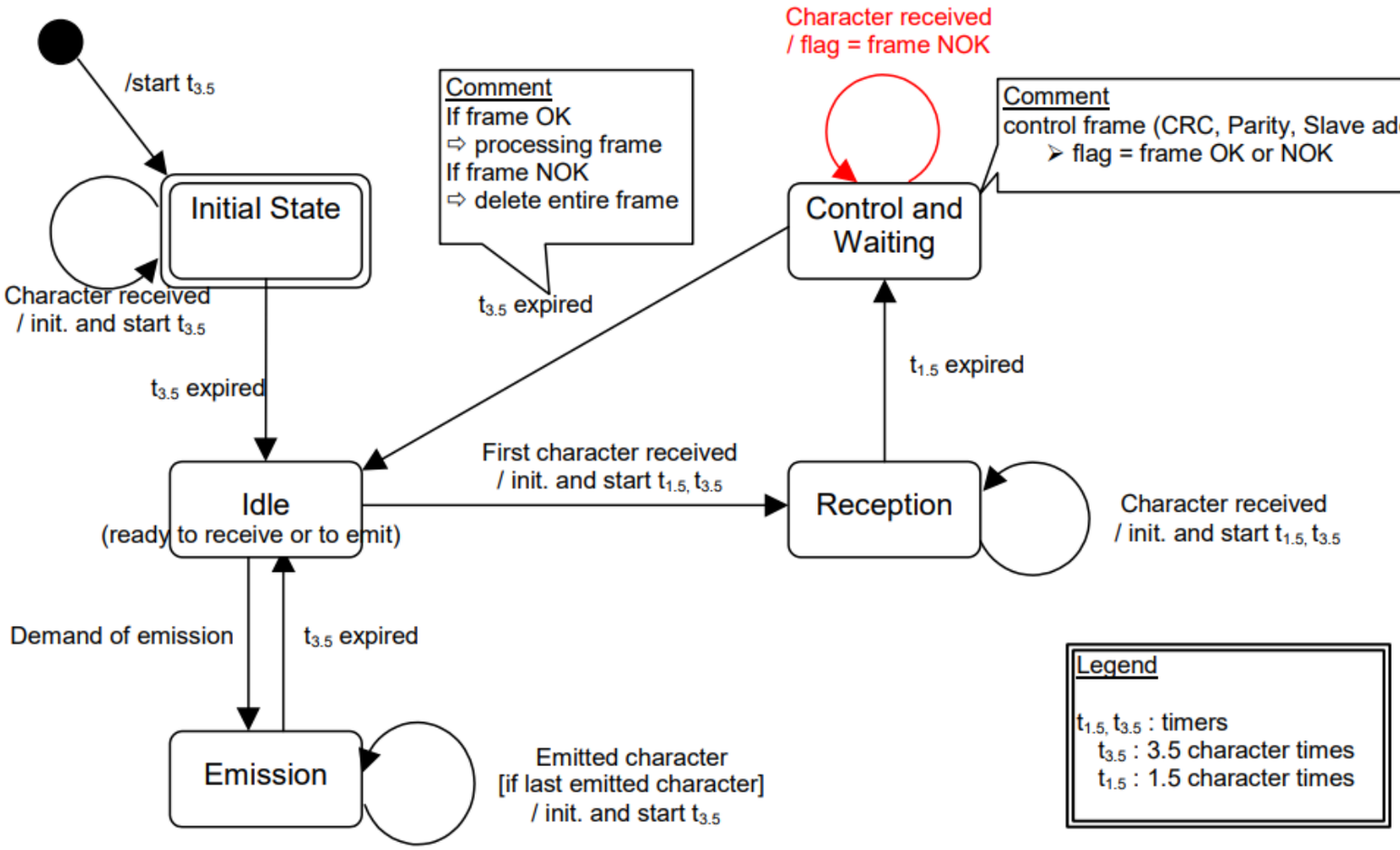


Figure 14: RTU transmission mode state diagram

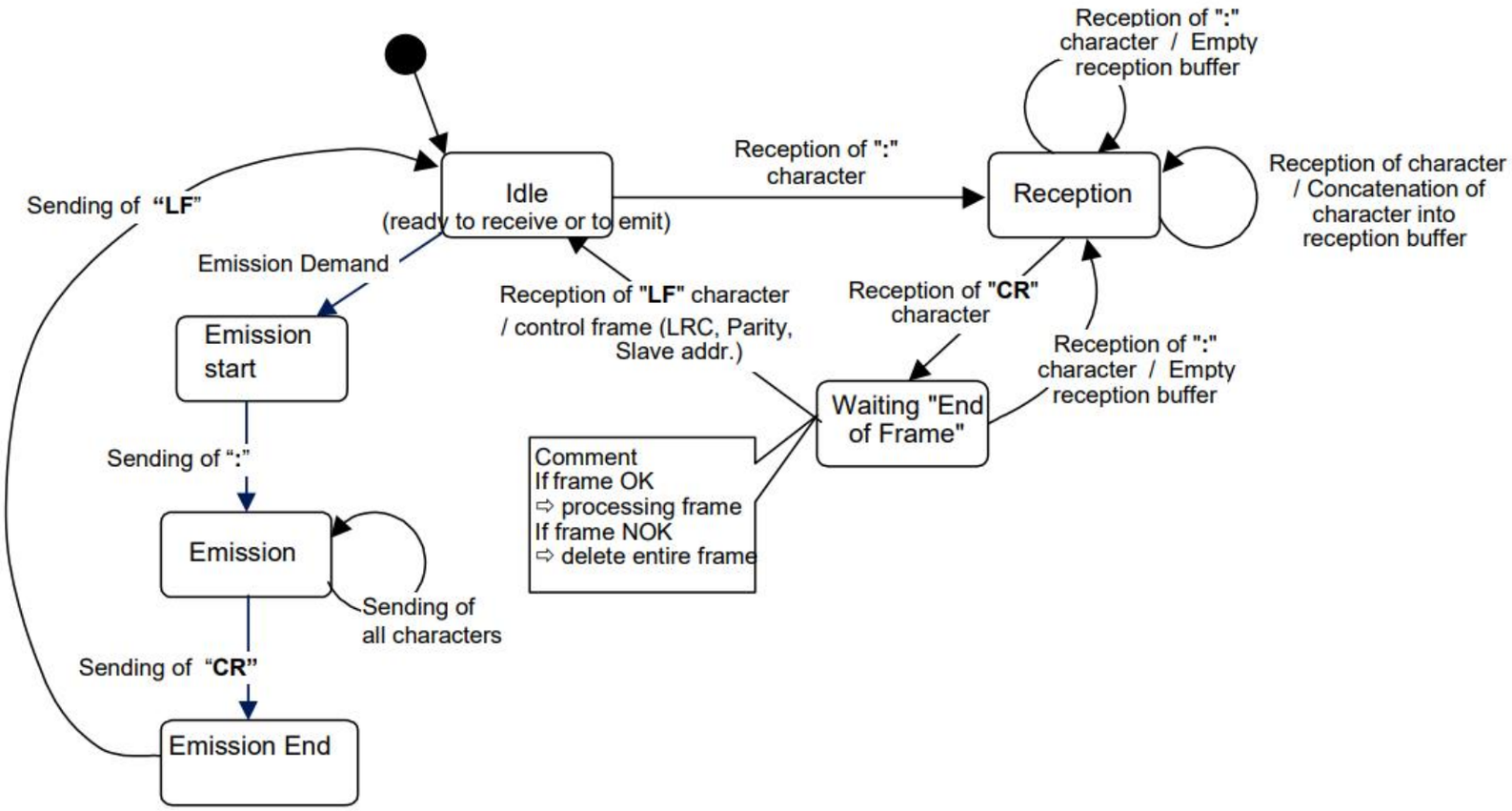


Figure 18: ASCII Transmission mode State diagram

Арифметика таймингов Modbus

Давайте в очередной раз рассмотрим вот такой запрос:

> *10 03 10 09 00 02 13 88*

и ответ на него:

> *10 03 04 41 CC 00 00 2F 31*

Запрос состоит из 8 байт, ответ из 9.

Предположим, что скорость обмена = 115200 бит/с.

Тогда время на отправку запроса составит:

- $\frac{8*11}{115200} = 0.76$мс; округлим до **1 мс**

8 – это число байт в запросе, а 11 – размер «символа» (в который входит байт данных, а также старт–бит, стоп–бит и бит чётности).

- время на отправку ответа, соответственно, будет приблизительно таким же (**1 мс**).

По спецификации Modbus:

- t(3.5) = 1.750 мс (округлим до **2 мс**)

Предположим, что:

- slave–устройству требуется **2 мс** на обработку запроса мастера;
- время задержки (response delay) = **2 мс**;
- пауза перед следующим запросом master–устройства (turnaround delay) = **10 мс**.

В сумме: 1 мс + 1 мс + 2 мс + 2 мс + 2 мс + 10 мс = **18 мс**

Мы вполне можем округлить это значение до **20 мс** для удобства расчётов.

Таким образом, в данных условиях за 1 секунду может быть произведено 50 операций чтения, в рамках каждой из которых происходит получение значений двух регистров – всего мы получим 100 регистров.

Изложенные выше предположения совершенно реалистичны; в существенном числе случаев реальные slave–устройства будут работать ещё медленнее, чем в нашем примере.

Обратите внимание, что из полученных нами 18 мс всего 2 мс уходят непосредственно на передачу данных. Всё остальное – различного рода задержки.

Если бы была возможность сделать это одним групповым запросом – то в приведённой выше сумме изменилось бы значение только одного слагаемого – времени, затраченного на отправку ответа.

Размера ответа составил бы 205 байт и, соответственно, на его передачу потребовалось бы:

- $\frac{205*11}{115200} = 20$мс

То есть время, затрачиваемое на считывание 100 регистров одним групповым запросом сопоставимо с временем считывания двух регистров одним запросом. Переход на групповой запрос в приведённом выше примере повысил бы эффективность использования канала связи в 50 раз.

Это наглядный пример того, что поддержка групповых запросов является одним из наиболее эффективных способов оптимизации обмена по протоколу Modbus.

Широковещательный запрос (broadcast). Часть 1

На диаграммах из предыдущих шагов можно увидеть обозначение «**broadcast mode**». Оно связано с широковещательным запросом, который мы уже несколько раз кратко упоминали в ходе нашего курса.

Широковещательный запрос – это особый вид запроса, который получат все slave–устройства в пределах данной линии связи. Slave–устройства не должны отвечать на такой запрос, потому что их одновременные ответы (которые представлены электрическими сигналами) «наложатся» друг на друга и не смогут быть восприняты master–устройством.

Поэтому широковещательный запрос используется исключительно с функциями записи (0x05, 0x0F, 0x06, 0x10).

Для отправки широковещательного запроса следует в пакете для поля адреса slave–устройства использовать значение **0**. Этот адрес специально зарезервирован для широковещательного запроса и, согласно спецификации Modbus Serial, не может быть назначен slave–устройству [1, п. 2.1].

Пример широковещательного запроса:

> *00 10 00 32 00 01 02 00 00 AF D2*

После отправки этого запроса – для всех slave–устройств в пределах данной линии связи в регистр 50 (0x0032) будет записано значение 0.

Rapid SCADA Modbus Parser

Protocol:
- (•) Modbus RTU
- ( ) Modbus TCP

Data Direction:
- (•) Request
- ( ) Response

Data Package (Application Data Unit):

```
00 10 00 32 00 01 02 00 00 AF D2
```

For example: 10 06 02 02 00 03 6A F2

Parse

Part of Data Package	Description	Value
00	Slave address	0x00 (0)
10	Function code	0x10 (16) - Write Multiple Registers
00 32	Starting address	Physical: 0x0032 (50) Logical: 0x0033 (51)
00 01	Quantity	0x0001 (1)
02	Byte count	0x02 (2)
00 00	Registers value	0x0000 (0)
AF D2	CRC	0xAFD2 (45010)

Copyright © 2025 Rapid SCADA

Широковещательный запрос (broadcast). Часть 2

Широковещательный запрос удобен, если требуется одновременно записать одни и те же значения в одни и те же регистры нескольких slave–устройств.

На линии связи могут присутствовать разные устройства; во многих ситуациях хотелось бы использовать широковещательный запрос для записи данных только в **некоторые** из них, но такой возможности нет. Широковещательный запрос получают все slave–устройства, подключенные к данной линии связи.

Если у slave–устройства нет регистра с адресом, указанным в широковещательном запросе, то оно просто никак на него не отреагирует (напомним, что ответ на широковещательный запрос не посылается).

Но если адреса регистров разных slave–устройств «пересекаются» и соответствуют различным параметрам, то отправка широковещательного запроса может привести к нежелательным последствиям – в разных устройствах в разные параметры будут записаны одни и те же значения; сложно представить ситуацию, в которой потребуется именно такое поведение.

Конкретное устройство может не поддерживать широковещательные запросы.

Но даже если такие запросы поддерживаются – их реализация может быть неудачной, например:

- slave–устройство позволяет назначить себе адрес 0;
- master–устройство позволяет указать для широковещательного запроса функцию чтения;
- master–устройство ожидает ответ на широковещательный запрос;
- master–устройство отправляет после любого (в т. ч. широковещательного) запроса записи запрос на чтение этих же регистров.

24.09.2012, 10:46

Arsie
Сотрудник Segnetics

Регистрация: Jan 2006
Адрес: Russia, SPb
Сообщения: 18 844
Благодарил(а): 15 раз(а)
Поблагодарили: 749 раз(а) в 676 сообщениях

Ответ: СМИ2

Цитата:

> Сообщение от **LordN**
> *не могу понять как на смлоджике организовать такую широковещательную посылку.*
> *это вообще возможно впринципе?*

Для наших контроллеров адрес 0 является таким же обычным адресом как и любой другой. Т.е. контроллер отправляет запрос и ждёт ответ.

В принципе, вы можете создать слейва с нулевым адресом и поставить тайм-аут 1 мсек, этим вы обеспечите минимально возможные задержки в работе мастера.

_____
Программа делает то что **написал** программист, а не то что он **хотел**.

Добро всегда побеждает зло. Кто победил - тот и добрый.

Segnetics в Telegram!

Источник: https://forum.segnetics.com/showpost.php?p=12405&postcount=7

Широковещательный запрос (broadcast). Реализация в СМИ2 и СМИ2–М

При разработке modbus–индикатора СМИ2 среди возможных пользовательских сценариев рассматривался следующий:

- что, если пользователю нужно как можно чаще обновлять данные на различных индикаторах, которые отображают разные параметры?

Логичным решением стала поддержка широковещательного запроса, но для этого потребовалось «научить» СМИ2 выделять из передаваемых в нём данных значение «своего» параметра.

Для этого в настройки индикатора был добавлен параметр **AD.AD** (additional address).

СМИ2 рассматривает сумму параметров **ADDR** (адрес индикатора) и **AD.AD** как «номер параметра в широковещательном запросе», и на основании него выделяет из данных этого запроса значение «своего» параметра. Каждый параметр в широковещательном запросе занимает 4 регистра (неиспользуемые регистры заполняются нулями); это позволяет передать в его рамках значение любого из типов данных, поддерживаемых СМИ2, и легко определить данные параметры по его номеру (параметр 1 занимает первые 4 регистра поля данных, параметр 2 следующие 4 регистра и т. д.).

Таким образом, в рамках одного широковещательного запроса можно записать не более 30 параметров (вспомним, что с помощью функции 0x10 можно записать не более 123 регистров).

Подробнее этот механизм рассмотрен в примере: https://owen.ru/uploads/40/example_smi2_broadcast.zip

***

В индикаторе СМИ2–М, являющимся развитием СМИ2, обработка широковещательного запроса реализована другим образом. В нём она является частным случаем режима **Modbus Spy,** который поддерживает этот индикатор. В этом режиме индикатор работает как slave–устройство, но «прослушивает» все запросы и ответы,

проходящие по линии связи, не отвечая ни на один из них и дожидаясь запроса с заданными характеристиками (адресом slave–устройства, кодом функции, адресом регистра).

> *При прослушивании линии связи возникает задача идентификации типа ответа – является ли он запросом или ответом. Для этого можно использовать алгоритм, рассмотренный нами в уроке 2.8.*

Это полезно для интеграции индикатора в те системы, где уже присутствуют master–устройства и slave–устройства, которые нельзя перенастроить, но при этом требуется реализовать отображение передаваемых между устройствами значений.

Для обработки широковещательного запроса – требуется в настройках индикатора задать режим **Modbus Spy** и задать для него следующие настройки:

- Адрес устройства = 0;
- Номер функции = 0x10 (Write Multiple Registers);
- Адрес регистра – адрес регистра, «входящего в состав» широковещательного запроса.

СМИ2–М будет ждать на линии связи запрос с данным адресом и номером функции, и при его обнаружении выделит из него нужное количество регистров (зависящее от выбранных в настройках индикатора типа данных), начиная с заданного, и отобразит на индикаторе.

Имя	Значение	Значение по умолчанию
Настройки порта RS-485		
Скорость COM-порта	115200	9600
Размер данных	8 бит	8 бит
Кол. стоп-битов	1 стоп-бит	1 стоп-бит
Контроль чётности	Нет	Нет
Признак конца кадра	IDLE frame	IDLE frame
Индикатор		
Настройки Modbus Master		
Настройки Modbus Spy		
Адрес устройства	0	1
Номер функции	(0x10) Write Multiple registers	(0x04) Read Input Registers
Адрес регистра	1000	1
Общие настройки Modbus		
Настройки индикатора		
Оперативные значения		
Встроенная логика		
Режим работы устройства	SPY	SLAVE
Сохранение параметров на flash по RS	0	0
Статус прибора		

Подробнее этот механизм рассмотрен в примере: https://owen.ru/uploads/250/ispolzovanie_shirokoveshhatelnoj_komandy.pdf

и связанным с ним учебном видео: https://owen.ru/media/video/ispolzovaniye_shirikoveshatelnoy_komandy

> *Работа СМИ2–М в режиме Modbus Spy рассмотрена в отдельном примере:*
>
> *https://owen.ru/uploads/250/nastrojka_obmena_v_rezhime_spy.pdf*
>
> *https://owen.ru/media/video/nastroika_smi2m_spy_mode*

Подведение итогов

Мы обсудили параметры обмена, используемые в Modbus Serial, и рассмотрели диаграммы состояний master– и slave–устройств.

Познакомились с широковещательным запросом (broadcast) и изучили особенности его реализации в индикаторах СМИ2 и СМИ2–М.

В следующем уроке мы немного отвлечёмся от протокола Modbus и поговорим о некоторых аспектах интерфейса RS–485, поверх которого он обычно используется.

2.13 Тестовые задания

Ответы приведены в п. 9.

1. Укажите размер символа (в битах) для протокола Modbus RTU.

2. Укажите формулировки, справедливые для протокола Modbus ASCII.
   - Бинарный протокол
   - Позволяет передавать данные быстрее, чем Modbus RTU
   - Контрольная сумма является болеё надежной, чем у протокола Modbus RTU
   - Позволяет передать данные даже при возникновении существенных задержек между передаваемыми символами
   - Текстовый протокол
   - Размер пакета больше, чем у аналогичного пакета Modbus RTU

3. Для какой цели в спецификации Modbus введён интервал времени t(3.5)?
   - Для того, чтобы slave-устройство успело переключить свой COM-порт из режима передачи в режим приёма
   - Для того, чтобы master-устройство успело переключить свой COM-порт из режима передачи в режим приёма
   - Для определения конца передаваемого пакета
   - Для определения момента отправки очередного запроса к slave-устройству
   - Для определения отсутствия связи между slave-устройством и master-устройством

4. Какой адрес slave–устройства используется при отправке широковещательного (broadcast) запроса?
   - 0
   - 255
   - 248
   - 247

5. Что такое таймаут ответа?
   - Время, в течение которого slave-устройство ожидает ответ от master-устройства
   - Пауза перед ответом на некорректный запрос для формирования правильного кода ошибки
   - Пауза, которая позволяет slave-устройству успеть переключить свой COM-порт из режима передачи в режим приёма
   - Время, в течение которого master-устройство ожидает ответ от slave-устройства

6. Интервалы времени t(1.5) и t(3.5) являются параметрами протокола...
   - Modbus TCP
   - Modbus ASCII
   - Modbus RTU

7. Расположите в правильном порядке состояния master-устройства после отправки запроса slave-устройству в случае получения некорректного ответа, например: несоответствия выделенной из ответа и рассчитанной контрольной суммы, некорректной структуры пакета и т.д.

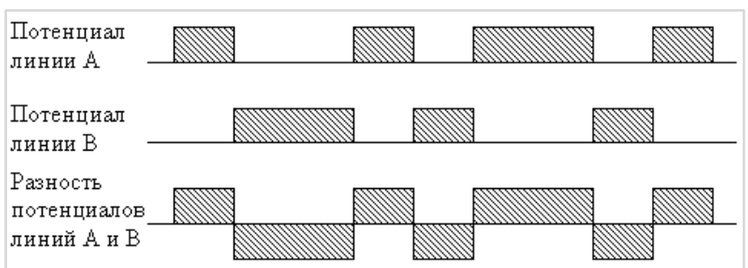
Waiting for reply (ожидание ответа)
Idle (ожидание)
Processing error (обработка ошибки)
Processing reply (обработка ответа)
Idle (ожидание отправки нового запроса)

2.14 Основы RS–485

> *RS–485 (также называемый EIA/TIA–485) – это стандарт физического уровня, определяющий электрические характеристики передатчиков и приёмников, подключаемых к последовательной линии связи; первая версия стандарта появилась в 1983 году. Аббревиатура RS означает Recommended Standard (рекомендуемый стандарт).*
>
> *В уроке использованы материалы следующих статей (мы выражаем признательность их авторам):*
>
> *https://wirenboard.com/wiki/RS–485:_Wiring_and_Connection*
>
> *https://www.reallab.ru/bookasutp/2–promishlennie–seti–i–interfeisi/2–3–interfeisi–rs–485–rs–422–i–rs–232*
>
> *https://www.ivtechno.ru/articles–one?id=19*
>
> *Если у вас есть рекомендации по улучшению этого раздела – то, пожалуйста, опубликуйте их в комментариях.*
>
> *Далее рассматривается только двухпроводной (2–wire) вариант RS–485. Информация о 4–проводном (4–wire) варианте приводится в спецификации Modbus и другой технической литературе.*

В основе интерфейса RS–485 лежит **дифференциальный способ передачи** сигнала, при котором напряжение, соответствующее уровню логической единицы или нуля, отсчитывается не от «земли», а измеряется как разность потенциалов между двумя передающими линиями: **Data+** и **Data–**. Соответственно, на одной линии устанавливается «исходное» значение сигнала, а на второй – его инвертированное значение.

Если разность потенциалов между ними превышает 200 мВ – то по линии связи передаётся логическая «единица» (бит со значением TRUE); если она меньше –200 мВ – то передаётся логический «0» (бит со значением FALSE).



Источник: https://www.massflow.ru/info/cifrovye–kommunikaci/peredacha–dannyh–v–promyshlennyh–setyah–na–osnove–rs485/

При этом напряжение каждой линии относительно «земли» может быть произвольным, но не должно выходить за диапазон –7...+12 В, определяемый спецификацией RS–485.

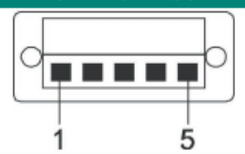
> *Смотрите: пакеты Modbus состоят из «символов», которые представляют собой наборы бит, а каждый бит – это разность потенциалов, установленная в течение некоторого времени (определяемого скоростью обмена) на линии связи – то есть электрический импульс.*

Соответственно, для подключения устройств между собой должны использоваться два провода. У разных производителей приняты различные обозначения для соответствующих клеммников; например, у приборов ОВЕН используются обозначения **A** (соответствующее **Data+**) и **B** (**Data–**). Соответственно, при подключении устройств нужно соединить их клеммы следующим образом: A – A, B – B.

В приборах компании Моха используется противоположное обозначение – клемма **A** соответствует **Data–**, а B соответствует **Data+**. Поэтому при подключении такого устройства к прибору компании ОВЕН потребуется соединение: A – B, B – A.

Terminal Block Pin Assignments

The UPort 1130/1130I/1150/1150I comes with a DB9 to terminal block converter, with pin assignments as shown below:

Terminal Block	Pin	RS-422 4-wire RS-485	2-wire RS-485
	1	TxD+(B)	–
	2	TxD-(A)	–
	3	RxD+(B)	Data+(B)
	4	RxD-(A)	Data-(A)
	5	GND	GND

Из–за этого расхождения в обозначениях – достаточно часто приходится определить полярность проводов линии связи опытным путём.

«Физическая» природа передачи данных по RS–485 обуславливает уже не раз упомянутые нами ограничения:

- на линии связи должно быть только одно master–устройство. Если их будет несколько – то они будут одновременно устанавливать на шине разные значения потенциалов, в результате чего с точки зрения slave–устройств будет передаваться какой–то «мусор»;
- на широковещательный запрос (broadcast) ни при каких обстоятельствах не должен отправляться ответ. Если несколько slave–устройств начнут отвечать одновременно – то получится точно такая же ситуация, что описана в предыдущем пункте;
- если slave–устройство сразу после получения запроса попробует отправить ответ – то, возможно, в этот момент master–устройство всё ещё будет «удерживать» на линии связи разность потенциалов, соответствующую последнему переданному биту. Для предотвращения этой ситуации в настройках некоторых slave–устройств присутствует параметр «Задержка ответа»;
- если master–устройство сразу после получения ответа попробует отправить новый запрос – то может зеркально повториться ситуация из предыдущего пункта. Для её предотвращения в настройках master'а обычно присутствует параметр «Пауза между запросами» (turnaround delay).

Подытожить всё вышесказанное можно так – на **последовательной** линии связи обмен происходит **последовательно** (запрос – ответ или таймаут, запрос – ответ или таймаут и т. д.); в каждый момент времени только одно устройство устанавливает на линии связи разность потенциалов. Кроме того, на последовательной линии связи устройства подключаются друг к другу последовательно – мы обсудим это спустя пару шагов.

Поэтому интерфейс RS–485 называют **полудуплексным** – он поддерживает «двустороннюю» связь между устройствами (любое из них может осуществлять передачу), но в каждый момент времени передача данных происходит только в одном направлении.

Витая пара. Экранированный кабель

Оба провода (A и B) должны быть проложены витой парой – так называется кабель, в котором оба провода на этапе изготовления свиты между собой с равным шагом. Это увеличивает помехозащищённость – токи, наводимые в соседних витках, вследствие явления электромагнитной индукции по «правилу буравчика» оказываются направленными навстречу друг другу и взаимно компенсируются. Степень компенсации определяется качеством изготовления кабеля и количеством витков на единицу длины.

Рекомендуется использовать специализированный промышленный кабель. Информация о подобных кабелях приведена в данной статье: https://cs–cs.net/cables–interface–rs–485–dmx

> *Автор статьи (Электрошаман) присутствовал на первоначальной очной версии курса и передал множество ценных замечаний и уточнений. Мы благодарим его за этот вклад.*

Если используется экранированный кабель, то экран должен быть заземлён в одной конкретной точке (в какой именно – не очень важно, но обычно это происходит на стороне интерфейса master–устройства) – это позволяет избежать возникновения «земляных петель». Экран обеспечивает защиту от паразитных емкостных связей и внешних магнитных полей. Спецификация Modbus Serial требует наличие экрана [1, п. 3.6], но в реальной жизни он используется далеко не всегда.

«Общий» провод

Спецификация RS–485 описывает возможность использования третьего провода, который называется «общим»; в англоязычной литературе он обозначается как signal ground, common ground и reference ground. С точки зрения спецификации Modbus Serial использование этого провода является обязательным [1, п. 3.3.2], но на практике – это требование выполняется далеко не всегда; многие устройства даже не имеют соответствующей клеммы.

Этот провод позволяет организовать выравнивание потенциалов устройств, что может быть важным в тех ситуациях, когда блоки питания устройств подключены к разным «землям», имеющим различные потенциалы. В этом случае дифференциальный сигнал, формируемый одним из устройств, может не восприниматься другим, потому что выйдет за диапазон, указанный в спецификации RS–485.

Иными словами: потенциал между источниками питания микросхем UART различных устройств может быть не равен 0 – а хотелось бы, чтобы был. Для этого и используется общий провод, который фактически соединяет «минусы» питания UART различных устройств. Если UART имеет изоляцию от источника питания – то можно обойтись и без этого провода.

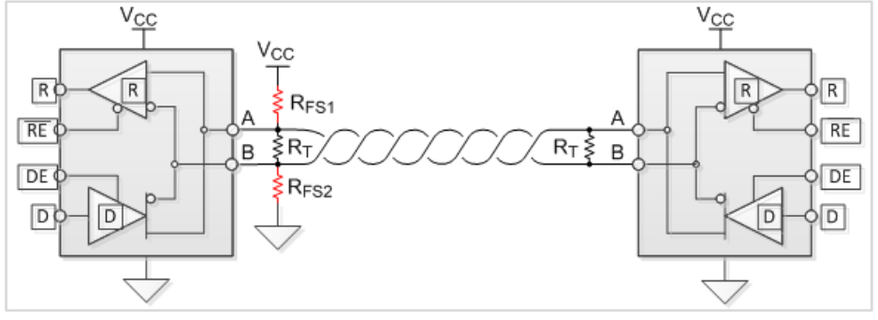
Подтягивающие резисторы

Как мы уже упоминали: если разность потенциалов между проводами A и B превышает 200 мВ – то по линии связи передаётся логическая «единица» (бит со значением TRUE); если она меньше –200 мВ – то передаётся логический «0» (бит со значением FALSE).

Диапазон –200...200 мВ соответствует «неопределённому» состоянию. Оно может наблюдаться на линии связи в момент отсутствия передачи данных. Если в этот момент на линию связи подействует помеха (о возможных источниках помех мы поговорим в конце урока) – то она может быть воспринята устройствами как последовательность бит, характеризующих начало передачи нового пакета (который, очевидно, после анализа будет оценен как некорректный).

Для предотвращения этой ситуации используются подтягивающие резисторы (также называется резисторами подтяжки, «подтяжкой», pull–up/pull–down, fail–safe biasing или line polarization). Спецификация Modbus Serial требует их наличия [1, п. 3.4.6], если они необходимы для работы конкретного устройства (к сожалению, в документации устройств редко указывается этот факт).

Резисторы подтяжки всегда устанавливаются парой в одном месте линии связи (обычно – на интерфейсе master–устройства). Один из них «подтягивает» провод A к +5 В, а второй – провод B к «земле». На рисунке ниже эти резисторы обозначены как R(FS1) и R(FS2). Создаваемое ими смещение делает состояние линии связи в момент отсутствия передачи данных определённым: на ней будет логическая «единица», которая будет восприниматься всеми устройствами как стоповый бит. В существенном количестве случаев даже воздействие помехи не повлияет на это смещение.



Источник: https://www.ti.com/document–viewer/lit/html/SSZTAK0

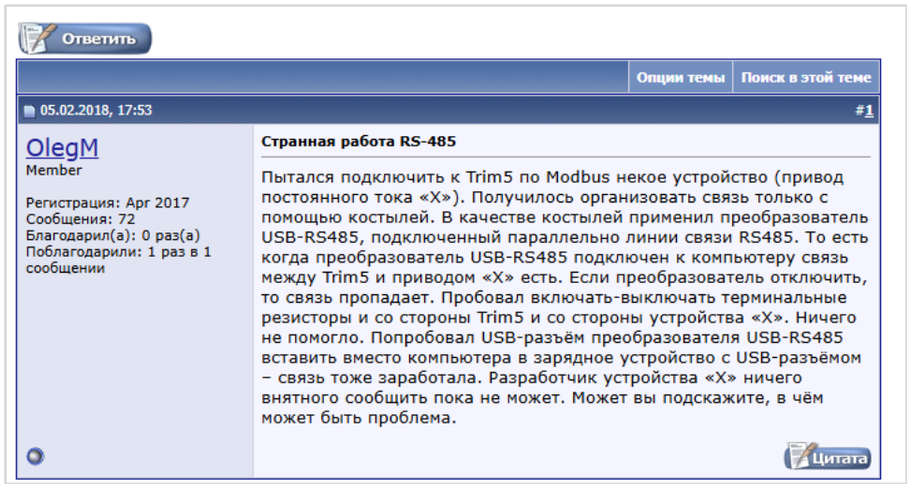
Согласно спецификации Modbus Serial, сопротивление этих резисторов должно выбираться из диапазона 450…650 Ом; чем больше устройств на линии связи – тем выше должно быть сопротивление. В реальности для резисторов подтяжки иногда используются и другие значения сопротивления.

Некоторые устройства имеют встроенные резисторы подтяжки, подключаемые аппаратно (например, с помощью DIP–переключателя) или программно; в противном случае резисторы (в случае необходимости) придётся подключать к клеммам прибора вручную.

Как понять, что в рамках конкретной системы требуется установка подтягивающих резисторов?

Вот описание характерной ситуации, которая наводит на эту мысль:

> *Пояснение: промышленные конвертеры RS–485/USB часто имеют встроенный резистор подтяжки.*



Источник: https://forum.segnetics.com/showthread.php?t=3485

Согласующие резисторы

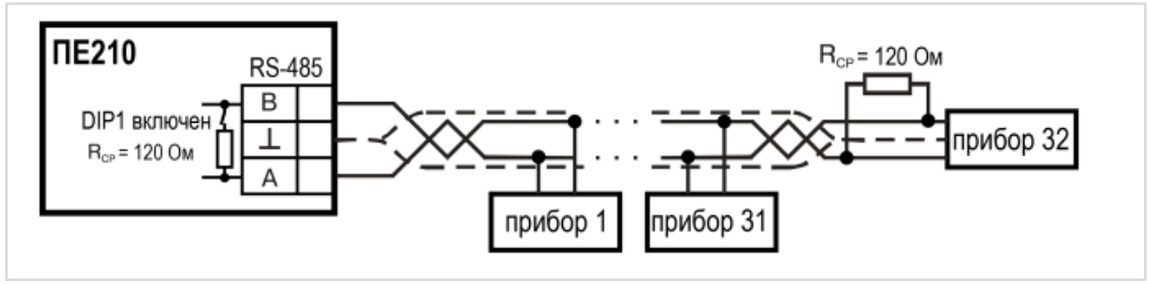
Передаваемые по интерфейсу RS–485 данные физически представляют собой электромагнитные волны, которые распространяются по проводнику (кабелю), постепенно теряя свою энергию. Достигая конца линии связи – волна отражается, что может привести к искажению следующих импульсов (т. е. – к повреждению пакета данных). Это особенно характерно в случае высокой скорости обмена (38400 и выше); на более низких скоростях затухание обычно происходит до начала передачи следующего бита.

Для устранения этого эффекта используются согласующие резисторы; они также называются терминирующими резисторами или «терминаторами». Сопротивление согласующих резисторов должно совпадать с волновым сопротивлением кабеля связи; типовым значением является 120 Ом. Терминирующие резисторы представляют собой согласованную нагрузку, используемую для «поглощения» электромагнитной волны в конце линии связи.

Спецификация Modbus Serial указывает на необходимость их использования [1, п. 3.4.5], но в реальной жизни это требование выполняется не всегда.

Согласующие резисторы обязательно должны устанавливаться парой – между клеммами A и B первого и последнего устройства линии связи. Ниже приведена схема подключения slave–устройств к облачному шлюзу ПЕ210, на которой можно увидеть многие моменты, которые мы обсудили:

- клеммы A и B и соединяющая их витая пара (её «перевитость» выражена даже графически);
- клемма «общего» провода, обозначенная пиктограммой «земли»;
- согласующие резисторы;
- интересным является ограничение в 32 прибора – мы обсудим его пару шагов спустя.



Некоторые устройства имеют встроенный согласующий резистор, подключаемый аппаратно (например, с помощью DIP–переключателя) или программно; в противном случае резистор (в случае необходимости) придётся подключать к клеммам прибора вручную.

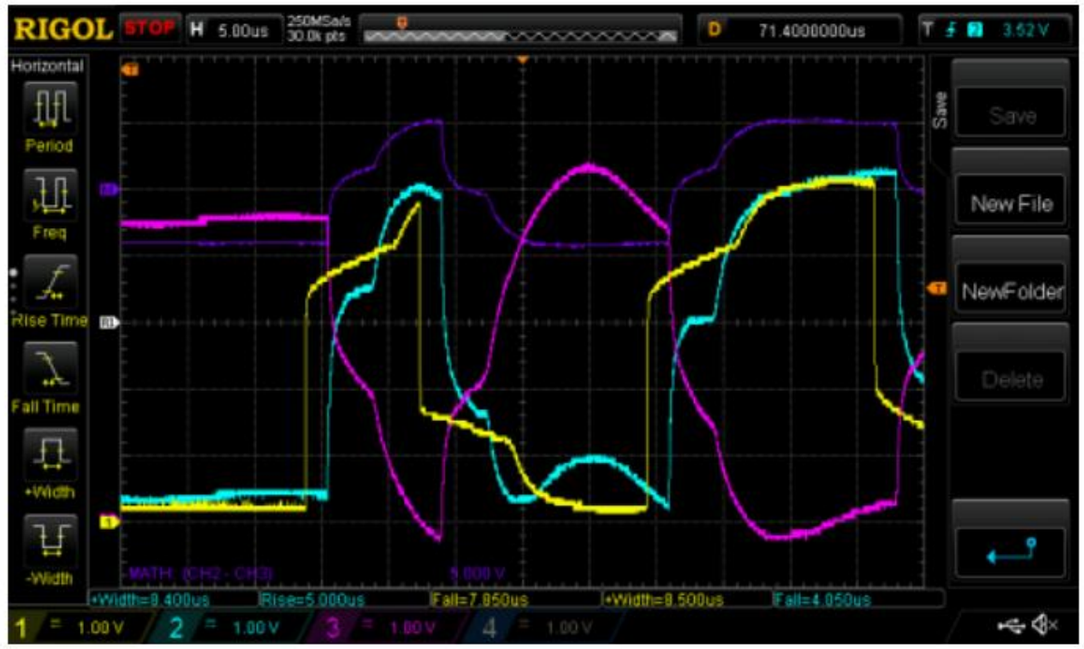
***

Эффект отражения характерен для длинных (20–30 метров и более) линий связи – в этом случае электромагнитной волне требуется определённое время на распространение по всей линии, а за это время может настать момент передачи очередного бита (что особенно вероятно в случае высокой скорости обмена); на коротких линиях связи волна практически мгновенно достигает конца линии связи и даже в случае отражения – затухает до начала передачи следующего бита.

Но в некоторых случаях согласующие резисторы могут потребоваться и на коротких линиях для устранения «звона»: так как провод является колебательным контуром, сигнал с крутым фронтом может приводить к остаточным колебаниям в этом контуре.

В реальной жизни необходимость установки согласующих резисторов обычно определяется опытным путём.

Сигнал на линии с терминаторами. Желтая - линия А передатчика, голубая - А приемника, розовая - В приемника, синяя - разность А-В - т.е. сигнал. Длина кабеля 600 метров, частота 115200.



Сигнал без терминаторов в середине линии — ситуация плохая, однако ж работает.

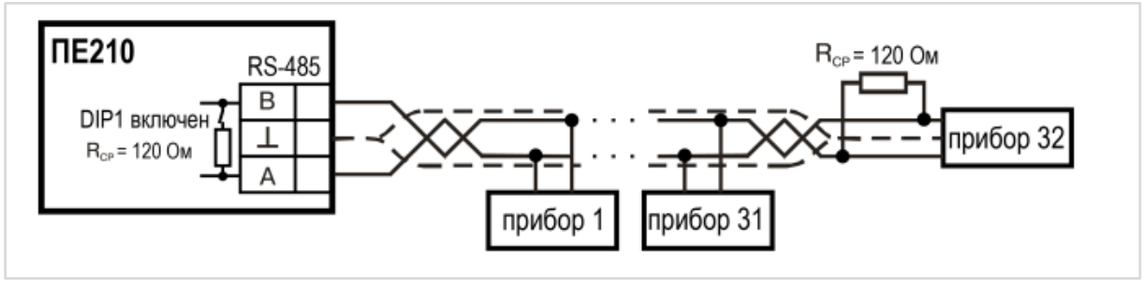
Источник: https://wirenboard.com/wiki/RS–485:_Wiring_and_Connection

Топология «шина»

> *В рамках нашего курса под «топологией» мы будем подразумевать принцип подключения участвующих в обмене устройств друг к другу.*

Описанный в прошлом шаге эффект отражения сигнала приводит к ограничению на топологии, которые возможны для интерфейса RS–485: поскольку отражения происходят от любой неоднородности, в том числе ответвлений от линии, то единственно правильной топологией сети будет такая, которая выглядит как единая линия без отводов. Такая топология называется «шиной».

В рамках шины клеммы **Data+** и **Data–** каждого устройства соединены с теми же клеммами «соседних» устройств; соответственно, два «крайних» устройства шины соединены только с одним устройством, а все остальные устройства – с двумя (предыдущим и следующим). Традиционно одним из «крайних» устройств является master–устройство, хотя это не является обязательным требованием.



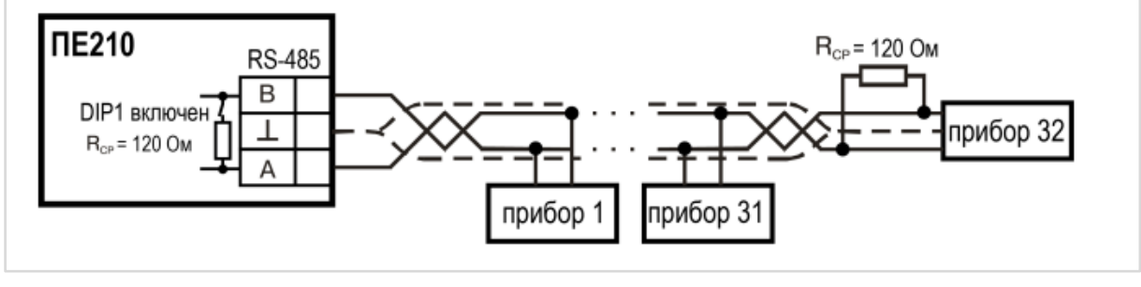
Спецификация Modbus Serial описывает потенциальную возможность наличия у шины коротких «ответвлений» [1, п. 3.4.3]. Длина одиночного ответвления не должна превышать 20 метров. В случае наличия нескольких ответвлений – длина каждого из них не должна превышать 40/N метров, где N – число ответвлений. Наличие ответвлений усугубляет эффект отражения (и, более того, в случае наличия ответвлений установка согласующих резисторов становится сомнительным решением – непонятно, как выбирать линию, на которой его стоило бы установить), поэтому рекомендуется по возможности избегать их.

Сколько slave–устройств можно подключить к шине Modbus Serial?

Мы помним, что в пакете Modbus под значение адреса slave–устройства выделяется один байт.

Спецификация указывает, что slave–устройству может быть назначен адрес из диапазона **1...247**; адрес **0** зарезервирован под широковещательный запрос, адреса **248...255** – под потенциальные будущие расширения стандарта. Таким образом – с точки зрения протокола на шине может присутствовать **247** устройств (или – при небольшом отступлении от спецификации – **255**).

Откуда же взялось число **32** на этом рисунке? (*его часто можно встретить в документации на совершенно разные приборы*)



Это число используется в спецификации RS–485 – но не в контексте «максимального числа устройств на шине». В спецификации указано, что

> *The electrical parameters specified in the following sections are selected so that a generator can drive a total load having the value of **32 unit loads** <...>*

А в спецификации Modbus Serial указано следующее [1, п. 3.4.1]:

> *A figure of 32 devices is always authorized on any RS485–MODBUS system without repeater. Depending of :*
>
> - *all the possible addresses;*
> - ***the figure of RS485 Unit Load used by the devices**;*
> - *and the line polarization in need be.*
>
> *a RS485 system may implement a larger number of devices. **Some devices allow the implementation of a RS485–MODBUS serial line with more than 32 devices, without repeater**. In this case these MODBUS devices must be documented to say how many of such devices are authorized without repeater.*

Unit load определяется как эквивалент потребления тока силой 1 мА при постоянном напряжении 12 В (или же, соответственно, эквивалент нагрузки в 12 кОм). Спецификация RS–485 требует от драйвера UART поддержки не менее 32 unit load. Но в настоящее время различные устройства имеют разный «вес»; например, устройство может «потреблять» 1/2, 1/4 и даже 1/8 unit load.

Соответственно, если использовать устройства с 1/8 unit load – то можно физически подключить к шине 247 (или даже 255) устройств – и их опрос будет корректно работать. К сожалению, в документации на устройства редко упоминается их unit load.

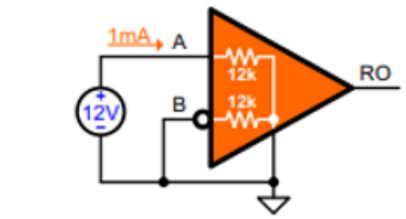


Figure 1. Definition of a Single Unit Load

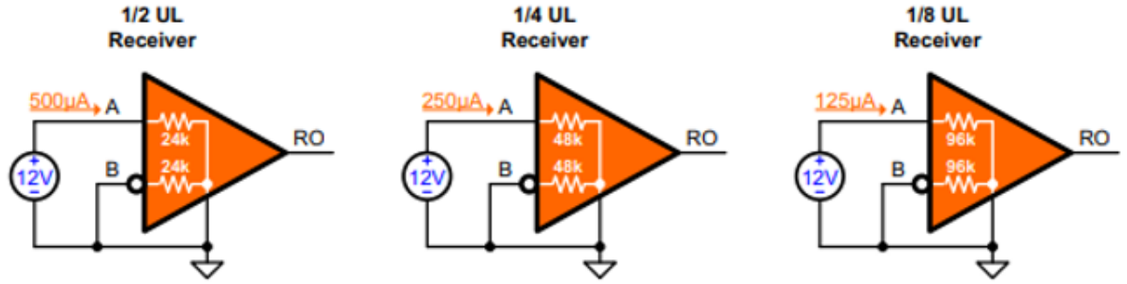


Figure 2. Fractional Unit Load Ratings Increase the Number of Transceivers Allowed on the Bus

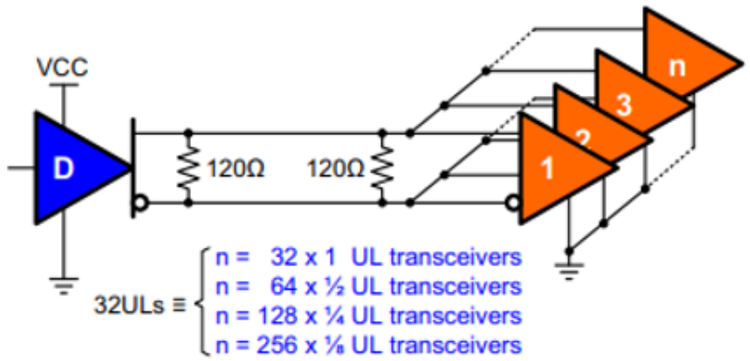


Figure 3. Establishing the Maximum Number of Bus Transceivers

The steady-state load on a single active generator shall be defined in the terms of unit loads (ULs). The load on the system caused by each receiver and passive generator shall be specified by the number or fractions of unit loads each presents. A unit load is defined by the current-voltage characteristics specified in 4.1.1. A load is defined as a passive generator (G), a receiver (R), or both (a Transceiver (T)). The electrical parameters specified in the following sections are selected so that a generator can drive a total load having the value of 32 unit loads and an effective total termination resistance as low as 60 Ω while providing a minimum differential voltage of 1.5 V.

COPYRIGHT 2000 Electronic Industries Alliance
Information Handling Services, 2000

Источник: https://www.renesas.com/us/en/document/oth/tb511–unit–load–concept?srsltid=AfmBOopRMByNCq9Aj6puBk68PASCrZLgO6zZsPsQQyIDMFU_zmsF9kc4

По ссылке ниже упомянут эксперимент компании Wiren Board, в рамках которого был произведён успешный опрос 247 устройств производства этой компании, собранных в одну шину:

https://wirenboard.com/wiki/RS–485


wirenboard

Повторители

Предположим, каждое из ваших slave–устройств потребляет один «полный» unit load, но вам нужно подключить к шине много таких устройств – больше, чем 32. Соответственно, требуется организовать усиление ослабленного электрического сигнала, передаваемого по шине RS–485.

Для этой цели используются специальные устройства – повторители (репитеры; repeater).

Схемотехнически повторитель представляет собой два порта RS–485, которые объединены друг с другом в пределах одного корпуса.

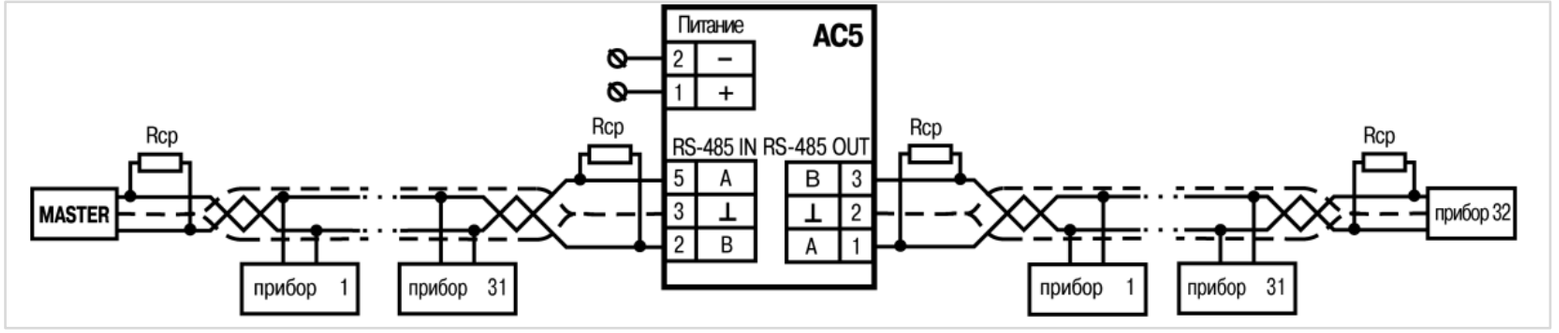
Примером такого устройства является АС5 от компании ОВЕН:

https://owen.ru/product/owen_as5

Он имеет встроенные согласующие резисторы и гальваническую изоляцию между своими интерфейсами. Смысл гальванической изоляции заключается в следующем: если к линии связи, подключенной к одному из портов повторителя, по ошибке будет подведено напряжение 220 В (или какое–то другое высокое напряжение) – то, конечно, интерфейсы всех подключенных к ней устройств выйдут из строя, но зато это никак не повлияет на устройства, подключенные ко второму интерфейсу повторителя; он выступит изолятором.

Наличие гальванической изоляции может являться ценным свойством приборов, эксплуатируемых в сложных условиях.

Схема подключения приборов к АС5:



Разветвители

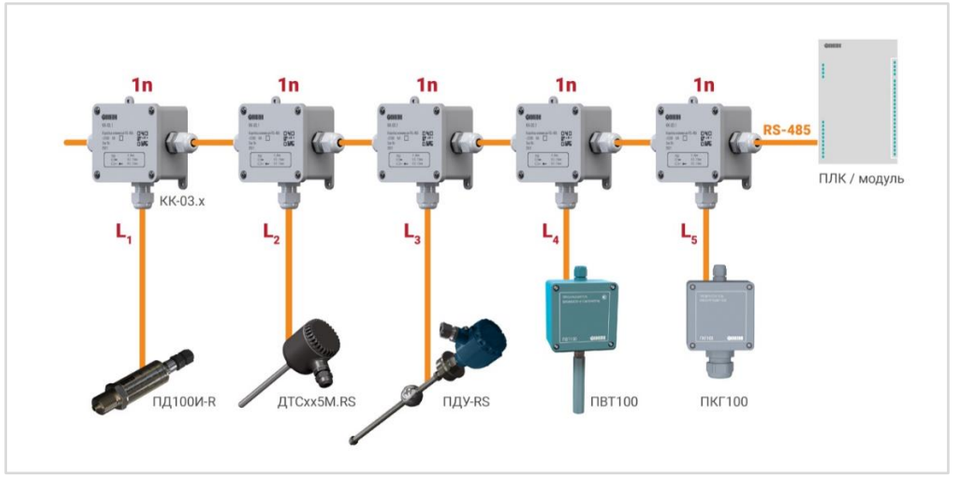
Мы уже упоминали, что для интерфейса RS–485 рекомендуется использовать топологию «шина» без каких–либо ответвлений. Но если по каким–то причинам без них не обойтись – то для удобства монтажа можно использовать специальные разветвители (клеммные коробки).

Примером такого устройства является КК–03 от компании ОВЕН:

https://owen.ru/product/kk_03

На рисунке показана линия связи с 5 ответвлениями. Согласно спецификации Modbus Serial:

- длина одного ответвления не должна превышать 40/5 = 8 метров.



Помехи

На линию связи могут действовать помехи. Их источниками могут быть:

- силовой кабель, заботливо проложенный в одном лотке с кабелем шины RS–485 (так делать не стоит);
- несколько кабелей шины RS–485, расположенных рядом друг с другом, по которым идёт обмен на разных скоростях (пример проявления);
- преобразователи частоты, контакторы и другое силовое оборудование;
- помехи, приходящие через контур заземления устройств;
- и так далее.

В реальной жизни – в роли источника помех в той или иной ситуации может выступать практически любое оборудование и окружающие факторы; см., например, информацию из этой статьи:

https://www.aktivsb.ru/statii/zashchita_ot_pomekh_v_liniyakh_svyazi_praktika_proektirovaniya_i_montazha_isb.html

Одним из типовых источников помех являются преобразователи частоты. Для них существуют специальные способы подавления помех – сетевые и моторные дроссели, ферритовые кольца и т. д. Более подробная информация о способе подавления помех может приведена в руководстве на конкретный преобразователь частоты.

«Плохой обмен»–бинго

Типовые сценарии, в которых наладка обмена между устройствами, вероятно, будет сложной:

- много (например, 32 и более) устройств, подключенных к одной линии связи;
- длина линии связи превышает несколько десятков метров;
- вместо промышленного кабеля для соединения устройств используется «лапша» (телефонный провод) без экрана и общего провода;
- линия связи имеет ответвления;
- подтягивающие и терминирующие резисторы установлены некорректно (например, установлен только один согласующий резистор или подтягивающие резисторы установлены на нескольких устройствах сразу или сопротивление резисторов не соответствует требованиям и т. д.);
- одновременно со всем вышесказанным, скорость обмена – 115200 бит/с, и поменять её по тем или иным причинам нельзя;
- линия связи проложена по подвалам, тоннелям и другим труднодоступным местам;
- в одном лотке вместе с кабелем шины RS–485 проложен силовой кабель;
- рядом установлены преобразователи частоты и другое силовое оборудование.

Часто одновременно с этим можно услышать две сакраментальные фразы от заказчика:

- «нужно, чтобы все несколько сотен параметров опрашивались за одну секунду и без единой ошибки в опросе»;
- «на другом объекте всё спроектировано/собрано/смонтировано/настроено точно также, и отлично работает».

Всех описанных выше ситуаций по возможности стоит избегать. В некоторых случаях вы можете указать на отступления от спецификации Modbus Serial, мешающие добиться стабильного обмена (как на эти замечания отреагирует заказчик – совсем другой вопрос).

Подведение итогов

В данном уроке мы рассмотрели основы интерфейса RS–485 и обсудили:

- как именно организовать подключение устройств;
- с какой целью могут использоваться подтягивающие и терминирующие резисторы;
- какое максимальное количество slave–устройств можно подключить к линии связи **(247** согласно спецификации или **255** при небольшом отступлении от неё);
- зачем используются повторители и разветвители;
- какие могут быть источники помех, влияющие на линию связи.

Типовым источником помех является силовое оборудование – в частности, преобразователи частоты.

Давайте рассмотрим, как настроить обмен с одним из них.

> *Ссылки на статьи:*
>
> *https://wirenboard.com/wiki/RS–485:_Wiring_and_Connection*
>
> *https://www.reallab.ru/bookasutp/2–promishlennie–seti–i–interfeisi/2–3–interfeisi–rs–485–rs–422–i–rs–232*
>
> *https://www.massflow.ru/info/cifrovye–kommunikaci/peredacha–dannyh–v–promyshlennyh–setyah–na–osnove–rs485/*
>
> *https://cs–cs.net/cables–interface–rs–485–dmx*
>
> *https://wirenboard.com/wiki/RS–485*
>
> *https://www.aktivsb.ru/statii/zashchita_ot_pomekh_v_liniyakh_svyazi_praktika_proektirovaniya_i_montazha_isb.html*

2.15 Практика: опрос ПЧВ

Преобразователь частоты (также называемый ПЧВ, ПЧ или VFD – variable–frequency drive) – это устройство, которое преобразует частоту переменного тока. Оно получает из электросети переменный ток с частотой 50 или 60 Гц, и преобразует его в ток с другой частотой; допустимый диапазон этой частоты зависит от конкретного ПЧВ и может, например составлять 0.01...600 Гц.

Часто ПЧВ используется для плавного управления частотой вращения подключенного к нему электродвигателя. Это позволяет реализовать более эффективное и экономичное использование оборудования, управляемого этим двигателем.

Давайте настроим обмен с преобразователем частоты ПЧВ1 [M01] по протоколу Modbus RTU.

Как и в предыдущих случаях – мы подключим его к ПК с помощью конвертера RS–485/USB AC4–M и будем опрашивать с помощью MasterOPC Universal Modbus Server.

Настройка ПЧВ1 [M01]

Настройку ПЧВ можно провести со встроенной локальной панели оператора (ЛПО) или через программу–конфигуратор. Процесс настройки описан в документации на прибор:

https://owen.ru/product/pchv_m01/documentation

Нам потребуется задать следующие настройки:

- адрес устройства (**F12.01**);
- скорость обмена (**F12.02**);
- другие параметры COM–порта (**F12.03**).

9.13 Группа F12: Параметры связи

Обозначение (адрес) и статус	Название	Описание	Значение по умолчанию (диапазон)	Режим управле-ния
F12.00 (0x0C00) STOP	Выбор ведущего- ведомого	0: Ведомый 1: Ведущий	0 (0-1)	U/f, SVC
F12.01 (0x0C01) STOP	Адрес связи по протоколу Modbus		1 (1-247)	U/f, SVC
F12.02 (0x0C02) STOP	Выбор скорости передачи данных	0: 1200 бит/с 1: 2400 бит/с 2: 4800 бит/с 3: 9600 бит/с 4: 19200 бит/с 5: 38400 бит/с 6: 57600 бит/с	3 (0-6)	U/f, SVC

Обозначение (адрес) и статус	Название	Описание	Значение по умолчанию (диапазон)	Режим управле-ния
F12.03 (0x0C03) STOP	Формат данных по протоколу Modbus	0: (N, 8, 1) Без проверки, Биты данных: 8, Стоп-бит: 1 1: (E, 8, 1) Проверка на четности Биты данных: 8, Стоп-бит: 1 2: (O, 8, 1) Проверка на нечетность Биты данных: 8, Стоп-бит: 1 3: (N, 8, 2) Без проверки, Биты данных: 8, Стоп-бит: 2 4: (E, 8, 2) Проверка на четности Биты данных: 8, Стоп-бит: 2 5: (O, 8, 2) Проверка на нечетность, Биты данных: 8, Стоп-бит: 2	0 (0-5)	U/f, SVC

Обратите внимание, что максимальное значение скорости обмена составляет 57600. Это означает, что данный ПЧВ не получится использовать на одной линии связи с устройствами, настроенными на скорость 115200.

Интерес вызывает параметр **F12.00** («Выбор ведущего–ведомого»).

Данный ПЧВ может работать не только в режиме Modbus Slave, но и в режиме Modbus Master. В этом случае он циклически отправляет значения своих выбранных параметров (см. **F12.10** и **F12.19**) с помощью широковещательного запроса. Настройки запроса определяются параметрами F12.11 – F12.18.

Мы будем использовать ПЧВ в режиме Modbus Slave.

Кроме сетевых настроек, нам потребуется задать ещё два параметра:

- параметру **F01.01** («Источник подачи сигнала запуска«) нужно присвоить значение **2** («Запуск по интерфейсу RS–485«);
- параметру **F01.02** («Источник задания частоты«) нужно присвоить значение **6** («Задание частоты по интерфейсу RS–485«).

Карта регистров

Обычно ПЧВ имеет сотни параметров, из которых большая часть является конфигурационными и представляют интерес только на этапе первоначальной настройки прибора.

Оперативные параметры ПЧВ1 [M01] описаны в п. 10 руководства по эксплуатации.

https://owen.ru/product/pchv_m01/documentation

Из этого раздела можно узнать, что:

- ПЧВ поддерживает только одну функцию чтения – **0x03 (Read Holding Registers)**;
- ПЧВ поддерживает только одну функцию записи – **0x06 (Write Single Register)**, то есть записать несколько регистров одним запросом не получится;
- все параметры ПЧВ имеют тип **Uint16**.

Нас будут интересовать три параметра:

- заданная частота. Она задаётся в виде целого значения со смещением десятичной точки – то есть значение **2500** соответствует частоте **25.00** Гц. Помните, мы уже видели этот подход в ТРМ201?;
- задаваемая команда. В столбце **Описание** приведён список возможных значений команд и связанных с ними операций. В каждый момент времени на ПЧВ1 [M01] можно отправить только одну команду. В некоторых других ПЧВ управление осуществляется с помощью битовой маски, называемой **командным словом**. Каждый бит этого слова соответствует определённой команде – соответственно, можно путём отправки одного значения передать сразу несколько команд.
- информация о состоянии ПЧВ (часто называемая **словом состояния**). Представляет собой битовую маску, каждый бит которой кодирует одно из возможных состояний ПЧВ.

Хороший вопрос – почему для каждого параметра указано два адреса регистра (0x2xxx / 0x3xxx)?

Дело в том, что параметры с адресами **0x2xxx** являются энергозависимыми – они теряют свои значения после перезагрузки ПЧВ. Параметры с адресами **0x3xxx** является энергонезависимыми – после перезагрузки они сохранят свои значения.

10 Карта регистров Modbus

Функция чтения **0x03**. Функция записи **0x06**. Тип данных в регистре – **Uint16**.

Адрес (hex)	Название	Тип доступа	Размерность (диапазон)	Описание
0x2000 /0x3000	Заданная частота	R/W*	0.01 Гц (0.00-320.00)	Заданная частота коммуникации
ПРИМЕЧАНИЕ * Тип доступа: • R — только чтение; • W — только запись; • R/W — чтение и запись.				
0x2001 /0x3001	Задаваемая команда	W	0x0000 (0x0-0x0103)	0x0000: Неверная команда; 0x0001: Запуск в прямом направлении; 0x0002: Запуск в обратном направлении; 0x0003: Фиксированная скорость в прямом направлении; 0x0004: Фиксированная скорость в обратном направлении; 0x0005: Останов с замедлением; 0x0006: Останов; 0x0007: Сброс аварии; 0x0008: Запрет запуска; ** 0x0009: Разрешение запуска; 0x0101: Эквивалент F2.07 = 1 (автонастройка с вращением), плюс команда «Пуск»; 0x0102: Эквивалент F2.07 = 2 (автонастройка без вращения), плюс команда «Пуск»; 0x0103: Эквивалент F2.07 = 3 (авто определение сопротивления статора), плюс команда «Пуск»
ПРИМЕЧАНИЕ ** После записи 0008 преобразователь остановится, чтобы снова запустить преобразователь частоты необходимо записать 0009 или перезагрузить преобразователь частоты.				

Адрес (hex)	Название	Тип доступа	Размерность (диапазон)	Описание
0x2002 /0x3002	Информация о состоянии преобразователя частоты	R	Двоичный код	Бит 0: 0 - остановлен, 1- в работе; Бит 1: 0 - нет разгона, 1 - разгон; Бит 2: 0 - нет торможения, 1 - торможение; Бит 3: 0 - вращение в прямом направлении, 1 - вращение в обратном направлении; Бит 4: 0 - преобразователь частоты исправен, 1 - ошибка преобразователя частоты; Бит 5: 0 - преобразователь частоты заблокирован, 1 - преобразователь частоты разблокирован; Бит 6: 0 - нет предупреждений, 1 - есть предупреждения Бит 7: 0 – запуск невозможен, 1 – запуск возможен

Настройка опроса в MasterOPC Universal Modbus Server

Мы уже несколько раз настраивали опрос через MasterOPC Universal Modbus Server, поэтому сейчас напомним только основные моменты.

- в коммуникационном узле выберите номер виртуального COM–порта вашего ПК и задайте его сетевые настройки в соответствии с настройками ПЧВ (см. параметры **F12.02**, **F12.03**);
- в узле устройства укажите адрес (см. параметр **F12.01**).

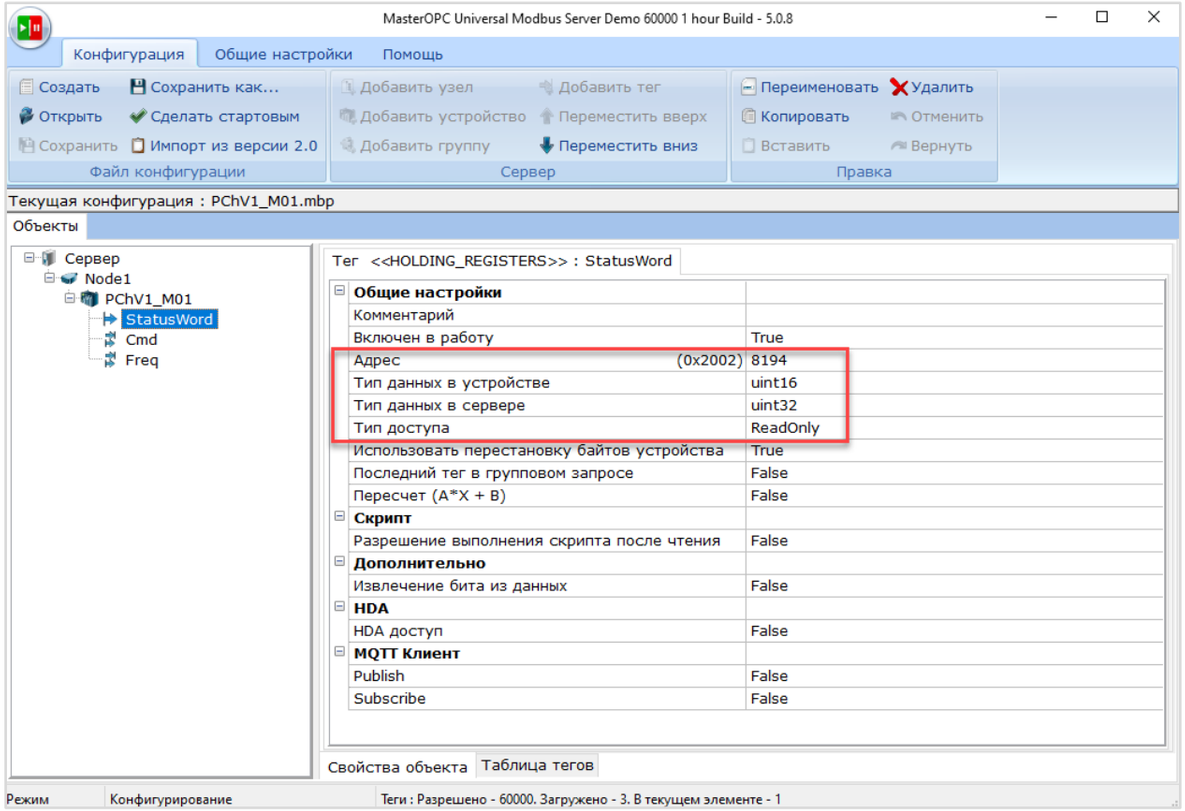
Добавьте в устройство три тега. Настройки тега **StatusWord** (Информация о состоянии ПЧВ) приведены ниже. Настройки остальных тегов отличаются только адресом регистра (см. предыдущий шаг) и типом доступа (для них нужен доступ **ReadWrite**).

Готовая конфигурация OPC–сервера доступна по ссылке:

https://ftp.owen.ru/ModbusCourse/01_OpcTemplates/PChV1_M01.mbp

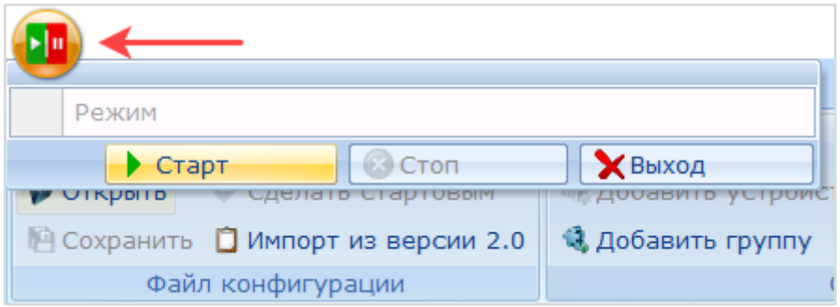
Для её открытия используйте команду Открыть.

> ***Примечание**: ПЧВ1 [M01] поддерживает только функцию записи **0x06 (Write Single Register)**. Если ваш ПЧВ поддерживает только функцию записи **0x10 (Write Multiple Registers)**, то в узле устройства следует для параметра **Не использовать команду WRITE_SINGLE_REGISTER (0x06)** установить значение **TRUE** (см. скриншот в комментариях к шагу).*



Проверка обмена

Запустите OPC–сервер. Для этого нажмите на круглую кнопку, расположенную в верхнем левом углу, и используйте команду **Старт**.



Если всё настроено и подключено корректно, то у всех тегов будет отображаться статус **Good**.

Введите в тег **Freq** значение **2500**, которое будет соответствовать частоте **25.00 Гц**.

В регистр **Cmd** введите значение **1**, которое соответствует команде запуска ПЧВ.

Если к ПЧВ подключен двигатель – вы увидите его вращение.

Тег **StatusWord** примет значение **129** – это означает «запуск возможен» и «в работе».

MasterOPC Universal Modbus Server Demo 60000 1 hour Build - 4.2.28

Стартовая конфигурация : simulator.mbp

Объекты

- Сервер
  - Node1
    - PChV1_M01
      - StatusWord
      - Cmd
      - Freq

Устройство <<PChV1_M01>>

Теги

Имя	Регион	Адрес	Значение	Каче...	Время (UTC)	Тип в се...	Тип в ус...	Дос
Node1.PChV1_M01.StatusWord	HOLDING_REGISTERS	(0x2002) 8194	129	GOOD	2025-04-09 09:17:4...	uint32	uint16	Rea
Node1.PChV1_M01.Cmd	HOLDING_REGISTERS	(0x2001) 8193	0	GOOD	2025-04-09 09:17:4...	uint32	uint16	Rea
Node1.PChV1_M01.Freq	HOLDING_REGISTERS	(0x2000) 8192	2500	GOOD	2025-04-09 09:17:4...	uint32	uint16	Rea

Ввод числа

1

Backspace | Del

7 | 8 | 9

4 | 5 | 6

1 | 2 | 3

0 | - | .

Да | Нет

Сообщения | Запросы | Сообщения скриптов

Режим вывода: Запущен Фильтр: PChV1_M01

09-04-2025 09:17:45.324 Node1::PChV1_M01:(COM4) Rx: [0011] 01 03 06 09 C4 00 00 00 81 10 5D
09-04-2025 09:17:45.278 Node1::PChV1_M01:(COM4) Tx: [0008] 01 03 20 00 00 03 0E 0B
09-04-2025 09:17:44.219 Node1::PChV1_M01:(COM4) Rx: [0011] 01 03 06 09 C4 00 00 00 81 10 5D
09-04-2025 09:17:44.172 Node1::PChV1_M01:(COM4) Tx: [0008] 01 03 20 00 00 03 0E 0B
09-04-2025 09:17:43.112 Node1::PChV1_M01:(COM4) Rx: [0011] 01 03 06 09 C4 00 00 00 81 10 5D
09-04-2025 09:17:43.065 Node1::PChV1_M01:(COM4) Tx: [0008] 01 03 20 00 00 03 0E 0B
09-04-2025 09:17:42.007 Node1::PChV1_M01:(COM4) Rx: [0011] 01 03 06 09 C4 00 00 00 81 10 5D
09-04-2025 09:17:41.960 Node1::PChV1_M01:(COM4) Tx: [0008] 01 03 20 00 00 03 0E 0B
09-04-2025 09:17:40.904 Node1::PChV1_M01:(COM4) Rx: [0011] 01 03 06 09 C4 00 00 00 81 10 5D
09-04-2025 09:17:40.857 Node1::PChV1_M01:(COM4) Tx: [0008] 01 03 20 00 00 03 0E 0B
09-04-2025 09:17:39.803 Node1::PChV1_M01:(COM4) Rx: [0011] 01 03 06 09 C4 00 00 00 81 10 5D
09-04-2025 09:17:39.756 Node1::PChV1_M01:(COM4) Tx: [0008] 01 03 20 00 00 03 0E 0B
09-04-2025 09:17:38.695 Node1::PChV1_M01:(COM4) Rx: [0011] 01 03 06 09 C4 00 00 00 81 10 5D

Адрес (hex)	Название	Тип доступа	Размерность (диапазон)	Описание
0x2002 /0x3002	Информация о состоянии преобразователя частоты	R	Двоичный код	Бит 0: 0 - остановлен, 1- в работе; Бит 1: 0 - нет разгона, 1 - разгон; Бит 2: 0 - нет торможения, 1 - торможение; Бит 3: 0 - вращение в прямом направлении, 1 - вращение в обратном направлении; Бит 4: 0 - преобразователь частоты исправен, 1 - ошибка преобразователя частоты; Бит 5: 0 - преобразователь частоты заблокирован, 1 - преобразователь частоты разблокирован; Бит 6: 0 - нет предупреждений, 1 - есть предупреждения Бит 7: 0 – запуск невозможен, 1 – запуск возможен

Несколько интересных сетевых настроек ПЧВ1 [M01]

В группе коммуникационных параметров (**F12.xx**) у ПЧВ1 [M01] есть несколько настроек, которые стоит упомянуть:

- **F12.04** – позволяет отключить отправку ответов на команды записи. Сложно представить, в каких ситуациях это может быть полезным. С точки зрения master–устройства – раз ПЧВ не ответил на запрос, то, значит, не получил его. Многие master–устройства попробуют в такой ситуации отправить запрос повторно;
- **F12.05** – это задержка, которую ПЧВ выдерживает между получением запроса от master–устройства и отправки ему ответа. Она позволяет master–устройству успеть переключить COM–порт из режима передачи в режим приёма. Мы уже видели подобную настройку у использованных ранее приборов – у ПВТ110–RS она называлась «таймаут ответа» (*откровенно неудачное название)*, у ТРМ210 – «rs.dl», у ТРМ1 – «Idle»;
- **F12.06** – это таймаут безопасного состояния (*названия параметра в документации ПЧВ, опять же, не очень удачное*). Если в течение этого времени ПЧВ не получает ни одного запроса от master–устройства – то выполняет действие, определяемое параметром **F12.07** (например, выполняет принудительную остановку).

F12.04 (0x0C04) RUN	Обработка ответа на передачу по протоколу Modbus	0: Отправлять ответ на команды записи 1: Не отправлять ответ на команды записи	0 (0-1)	U/f, SVC
F12.05 (0x0C05) RUN	Задержка ответа по протоколу Modbus		0 мс (0-500 мс)	U/f, SVC
F12.06 (0x0C06) RUN	Время неисправности тайм-аута связи по протоколу Modbus		1.0 с (0.1-100.0 с)	U/f, SVC
F12.07 (0x0C07) RUN	Обработка отключения связи	0: Отключено 1: Неисправность и свободная остановка 2: Предупреждение и продолжение работы 3: Принудительная остановка	0 (0-3)	U/f, SVC

***

Из руководства на ПЧВ1 [M01] можно также узнать, что он имеет встроенный согласующий резистор, который можно включить/отключить с помощью DIP–переключателя 3, расположенного под съемной панелью на лицевой стороне прибора:

6.8 Назначение переключателей

Блок переключателей располагается под съемной панелью на лицевой стороне прибора (см. рисунок ниже).

Рисунок 6.4 – Вид на блок DIP-переключателей

Таблица 6.6 – Назначение переключателей

Пере-ключа-тель	Положе-ние	Назначение
ПЧВ 0,75–5,5 кВт		
1	Вкл.	Аналоговый выход в режиме «напряжение». Диапазон выходного сигнала 0...10 В
2	Вкл.	Аналоговый выход в режиме «ток». Диапазон выходного сигнала 0...20 мА или 4...20 мА
3	Вкл.	Согласующий резистор 120 Ом подключен
	Выкл.	Согласующий резистор 120 Ом отключен

Преобразователь частоты AFD–E

Давайте для сравнения изучим, как реализован Modbus в преобразователе частоты AFD–E от компании KIPPRIBOR. Для этого воспользуемся документом «Преобразователи частоты KIPPRIBOR серии AFD–E/ Протокол связи Modbus RTU (интерфейс RS485)»:

https://kippribor.ru/?id=1297

1. Скорость обмена

AFD–E не поддерживает скорости 57600 и 115200, зато поддерживает нестандартную скорость 76800, которую редко можно встретить в других приборах.

1.1.2 Скорость передачи данных

В настройках доступно 7 значений скорости передачи данных: 1200 кбит/с, 2400 кбит/с, 4800 кбит/с, 9600 кбит/с, 19200 кбит/с, 38400 кбит/с, 76800 кбит/с.

2. Поддерживаемые функции Modbus

В отличие от ПЧВ1 [M01], данный преобразователь частоты поддерживает множество функций Modbus:

- все битовые функции (0x01, 0x02, 0x05, 0x0F);
- все типовые функции работы с регистрами (0x03, 0x04, 0x06, 0x10);
- функции диагностики (0x07, 0x08), о которых мы поговорим в следующих шагах;
- функцию 0x17, которая позволяет в рамках одного запроса провести как чтение, так и запись holding–регистров.

Табл. 1 – перечень поддерживаемых функций Modbus

Код функции (представление в шестнадцатеричном формате)	Код функции (представление в десятичном формате)	Функция
0x01	1	Чтение состояния одного бита
0x02	2	Чтение состояния нескольких бит
0x03	3	Чтение значений из нескольких регистров хранения
0x04	4	Чтение значений из нескольких регистров ввода
0x05	5	Запись состояния одного бита
0x06	6	Запись значения в один регистр хранения
0x07	7	Чтение состояния
0x08	8	Диагностика

0x0F	15	Запись состояния нескольких бит
0x10	16	Запись значений в несколько регистров хранения
0x17	23	Чтение / Запись нескольких регистров

3. Модель памяти

У AFD–E присутствуют все 4 возможные области памяти:

- coils;
- discrete inputs;
- input registers;
- holding registers.

Причём для каждой области памяти используется почти независимая адресация (почти – потому что последний адрес одной области совпадает с начальным адресом другой области). Такое техническое решение было совершенно необязательным – раз области памяти являются независимыми, то адреса их битов и регистров могут совпадать. Возможно, разработчики прибора посчитали, что так пользователю будет проще отличить области памяти друг от друга.

Табл. 2 – сводная таблица распределение адресов

Наименование	Диапазон адресов, представление в шестнадцатеричном формате (Hex)	Диапазон адресов, представление в десятичном формате (Dec)	Поддерживаемая функция Modbus
Биты командного слова Полупроводниковые дискретные выходы Релейные дискретные выходы	0x1000...0x1100	4096... 4352	0x01 0x05 0x0F
Биты слова состояния Дискретные входы	0x1100...0x1200	4352...4608	0x02
Аналоговые входы	0x1200...0x1300	4608...4864	0x04
Прикладные параметры, Параметры мониторинга, Командное слово, Слово состояния, Задаваемые значения Modbus, Карта параметров мониторинга, Карта параметров управления	0x1300...0x1400	4864...5120	0x03 0x06 0x10 0x17

4. Командное слово

Управление AFD–E реализовано с помощью командного слова.

Каждый бит командного слова соответствует одной команде, например:

- бит 1 – разрешение работы;
- бит 2 – разрешение запуска;
- бит 4 – пуск/стоп;
- и т. д.

Командное слово представлено как в области holding–регистров, так и в области coils (в виде 16 отдельных бит).

Аналогично обстоит дело со словом состояния – оно представлено как в виде input–регистра, как и виде 16 discrete inputs.

Функция 0x17 (Read/Write Multiple registers)

Функция **0x17 (Read/Write Multiple registers)** позволяет в рамках одного запроса объединить чтение и запись holding–регистров.
Эта функция не очень часто поддерживается устройствами – но, например, её можно использовать при опросе рассматриваемого нами ПЧВ AFD–E.

Давайте рассмотрим структуру запроса с данной функцией, взяв за основу пример из спецификации Modbus:

> *10 17 00 03 00 06 00 0E 00 03 06 00 FF 00 FF 00 FF 1A C4*

- 10h – адрес slave–устройства;
- 17h – код функции;
- 0003h – начальный адрес считываемых регистров;
- 0006h – количество считываемых регистров;
- 000Eh – начальный адрес записываемых регистров;
- 0003h – количество записываемых регистров;
- 06h – количество байт в записываемых регистрах (всегда в 2 раза больше, чем количество записываемых регистров);
- 00FFh – значение первого записываемого регистра;
- 00FFh – значение второго записываемого регистра;
- 00FFh – значение третьего записываемого регистра;
- 1AC4h – контрольная сумма.

Таким образом, в рамках этого запроса осуществляется:

- чтение 6 регистров, начиная с регистра с адресом 3;
- запись трёх регистров, начиная с регистра с адресом 14. В каждый из регистров записывается значение 255.

Можно заметить, что запрос по своей структуре является комбинацией запросов с кодом функции **0x03 (Read Holding Registers)** и **0x10 (Write Multiple Registers)**.

Структура ответа на этот запрос совпадает со структурой ответа для кода функции 0x03:

> *10 17 0C **00 FE 0A CD 00 01 00 03 00 0D 00 FF** CC 75*

Выделенное **жирным** шрифтом поле представляет собой данные ответа. Соответственно,

- регистр 3 имеет значение 254 (0x00FE);
- регистр 4 имеет значение 2765 (0x0ACD);
- регистр 5 имеет значение 1 (0x0001);
- регистр 6 имеет значение 03 (0x0003);
- регистр 7 имеет значение 13 (0x000D);
- регистр 8 имеет значение 255 (0x00FF).

Функция 0x07 (Read Exception Status)

Функция **0x07** позволяет получить один байт данных. Спецификация Modbus подразумевает, что этот байт представляет собой битовую маску, каждый бит которой соответствует одной из возможных ошибок устройства.

Структура запроса для данной функции предельно проста – он включает в себя только адрес slave–устройства, код функции и контрольную сумму:

> *10 07 4D B2*

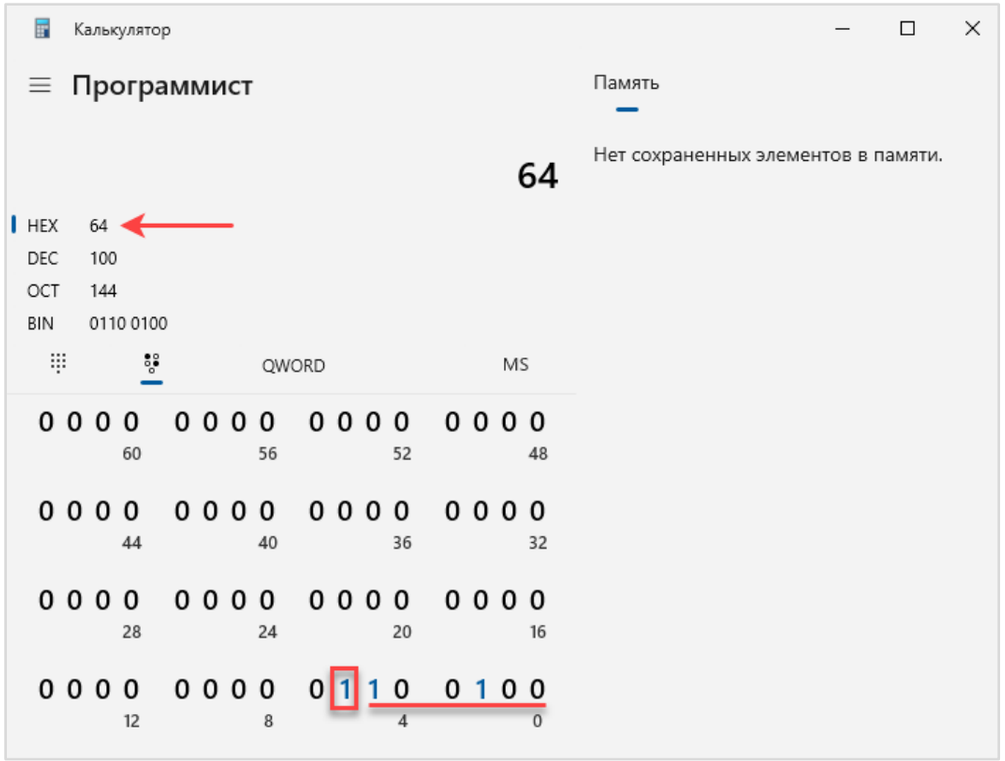
Ответ включает в себя один байт данных

> *10 07 64 73 DE*

В руководстве AFD–E указано, как следует интерпретировать значение этого байта:

- бит 7 соответствует признаку ошибки. Если он имеет значение TRUE (1) – значит, в процессе работы ПЧВ возникла ошибка;
- бит 6 соответствует признаку тревожной ситуации. Если он имеет значение TRUE (1) – значит, в процессе работы ПЧВ возникла подобная ситуация;
- остальные 5 содержат код ошибки или тревожной ситуации в виде целого числа (подразумевается, что в каждый момент времени может возникнуть либо ошибка, либо тревожная ситуация; одновременно этого произойти не может).

Значение **0x64** означает, что возникло тревожное сообщение с кодом **36** (**0x24**). Таблица с расшифровкой кодов приведена в документации на ПЧВ.


Калькулятор
Программист
Память
64
HEX 64
DEC 100
OCT 144
BIN 0110 0100
QWORD
MS
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
60 56 52 48
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
44 40 36 32
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
28 24 20 16
0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0
12 8 4 0
Нет сохраненных элементов в памяти.


Калькулятор
Программист
Память
24
HEX 24
DEC 36
OCT 44
BIN 0010 0100
QWORD
MS
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
60 56 52 48
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
44 40 36 32
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
28 24 20 16
0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0
12 8 4 0
Нет сохраненных элементов в памяти.

В настоящее время использование этой функции выглядит как атавизм. Она крайне редко поддерживается в приборах и ПО. Это означает, что будет трудно найти master–устройство, которое будет способно прочитать с ПЧВ информацию об ошибках. Примером подобного устройства является контроллер, который позволяет напрямую работать с COM–портом для отправки произвольных запросов и получения ответов. См. пример реализации этой функции (а также функции 0x08) в ПЛК с CODESYS V3.5 для опроса AFD–E:

https://owen.ru/forum/showthread.php?t=28167&p=414869&viewfull=1#post414869

Гораздо удобнее было бы предоставить эту информацию в виде двух регистров, доступных для чтения функцией **0x03** или **0x04**. В одном из регистров можно было бы разместить тип сообщения (например, 1 – ошибка, 2 – тревожное сообщение), а во втором – код сообщения.

Функция 0x08 (Read Diagnostics)

Функция **0x08** тоже является атавизмом и редко поддерживается устройствами. С помощью неё можно выполнить одну из диагностических команд (**subfunction**). Спецификация Modbus описывает 15 таких команд. AFD–E поддерживает 8 из них.

> *TPM201, опрос с которым мы настроили в уроке 2.6, тоже поддерживает эту функцию, но в рамках только одной команды – 0x00 (Return Query Data).*

Структура запроса выглядит следующим образом:

> *10 08 <код команды> <данные команды> <CRC>*

Код команды занимает 2 байта. Размер поля данных зависит от команды.

Ответ выглядит следующим образом:

> *10 08 <код команды> <данные> <CRC>*

Обзорно рассмотрим поддерживаемые функцией команды:

- **0x00 (Return Query Data)** – в ответ на команду возвращается её «эхо»; это может использоваться для проверки наличия связи со slave–устройством на этапе наладки обмена;
- **0x01 (Restart Communications Option)** – эта команда используется для переинициализации COM–порта slave–устройства. В этот момент все счётчики событий (см. о них ниже) обнуляются. Отправка этой команды – единственный способ вывести slave–устройство из «режима прослушивания» (см. ниже);
- **0x02 (Return Diagnostic Register)** – в ответ на команду возвращается значение регистра диагностики. Его содержимое зависит от конкретного slave–устройства;
- **0x03 (Change ASCII Input Delimiter)** – эта команда имеет смысл только в рамках протокола Modbus ASCII и позволяет указать символ, который будет являться признаком конца пакета (по умолчанию таким символом является спецсимвол <LF> с ASCII–кодом 0x0D);
- **0x04 (Force Listen Only Mode)** – эта команда переводит slave–устройство в «режим прослушивания». В этом режиме устройство перестает отвечать на какие–либо запросы. Для отключения «режима прослушивания» нужно использовать команду **0x01 (Restart Communications Option)**;
- **0x0A (Clear Counters and Diagnostic Register)** – эта команда используется для сброса счётчиков (см. ниже) и обнуления регистра диагностики. Надо отметить, что сброс счётчиков также должен происходить в случае перезагрузки устройства;
- **0x0B (Return Bus Message Count)** – в ответ на команду возвращается общее число пакетов, которые slave–устройство «увидело» на шине за время своей работы (включая пакеты, предназначенные для других устройств);

- **0x0C (Return Bus Communication Error Count)** – в ответ на команду возвращается число ошибок при проверке контрольной суммы, произошедших за время работы устройства;
- **0x0D (Return Bus Exception Error Count)** – в ответ на команду возвращается число ответов с кодом ошибки Modbus, отправленных данным slave–устройством за время его работы;
- **0x0E (Return Server Message Count)** – в ответ на команду возвращается общее число запросов, обработанных данным slave–устройством (включая широковещательные запросы) за время его работы;
- **0x0F (Return Server No Response Count)** – в ответ на команду возвращается общее число запросов, на которые slave–устройство по каким–то причинам не отправило ответ (ни «нормальный» ответ, ни ответ с кодом ошибки Modbus) за время его работы;
- **0x10 (Return Server NAK Count)** – в ответ на команду возвращается число ответов с кодом ошибки Modbus **0x07 (NEGATIVE ACKNOWLEDGE)**, отправленных данным slave–устройством за время его работы;
- **0x11 (Return Server Busy Count)** – в ответ на команду возвращается число ответов с кодом ошибки Modbus **0x06 (SERVER DEVICE BUSY)**, отправленных данным slave–устройством за время его работы;
- **0x12 (Return Bus Character Overrun Count)** – в ответ на команду возвращается число запросов, полученных данным slave–устройством, которые оно не смогло обработать из–за переполнения приёмного буфера COM–порта (это может быть вызвано тем, что символы поступают быстрее, чем могут быть сохранены, или потерей символов из–за аппаратной неисправности);
- **0x14 (Clear Overrun Counter and Flag)** – эта команда используется для обнуления счётчика команды **0x12** и сброса флага ошибки.

Надо отметить, что упомянутые выше счётчики ошибок сбрасываются в случае:

- перезагрузки slave–устройства;
- переинициализации драйвера Modbus устройства;
- получения запроса с кодом функции 0x08 и командой 0x0A (Clear Counters and Diagnostic Register).

ПЧВ AFD–E поддерживает следующие команды функции **0x08**:

- 0x00 (Return Query Data);
- 0x01 (Restart Communications Option);
- 0x04 (Force Listen Only Mode);
- 0x0A (Clear Counters and Diagnostic Register);
- 0x0B (Return Bus Message Count);
- 0x0C (Return Bus Communication Error Count);
- 0x0D (Return Bus Exception Error Count);
- 0x0E (Return Server Message Count).

***

С функцией **0x08** связаны ещё две функции Modbus:

- **0x0B (Get Comm Event Counter);**
- **0x0C (Get Comm Event Log).**

ПЧВ AFD–E не поддерживает ни одну из них, но давайте уделим им немного внимания.
Первая из этих функций позволяет получить слово статуса и количество событий.
Вторая функция позволяет получить то же самое, а также количество сообщений и информацию о последних событиях.

- слово статуса определяет, выполняет ли в данный момент устройство какой–то из полученных ранее «программных запросов». Оно имеет значение **0xFFFF**, если такой запрос выполняется, и **0x0000** – если не выполняется. «Программный запрос» – это запрос с кодом функции **0x13 (Program Controller)** или **0x14 (Poll Controller)**, которые исключены из современной версии спецификации Modbus. С этими же запросами связаны коды ошибок **0x06 (SERVER DEVICE BUSY)** и **0x07 (NEGATIVE ACKNOWLEDGE)**;
- счётчик событий содержит количество успешно обработанных запросов (в ответ на которые не был отправлен ответ с кодом ошибки Modbus). Этот счётчик может быть сброшен с помощью запроса с функцией **0x08** и командой **0x01 (Restart Communications Option)** или **0x0A**. Счётчик не учитывает запросы с кодом функции **0x0B** и **0x0C**;
- счётчик сообщений содержит то же самое значение, которое возвращается в ответ на запрос с кодом функции **0x08** и командой **0x0E (Return Server Message Count)**; см. информацию о нём выше.
- информация о последних событиях позволяет, как пример, узнать, что устройство недавно было переведено в «режим прослушивания» или получило диагностическую команду **0x01 (Restart Communications Option)**. Более подробная информация об этом приведена в спецификации Modbus.

Подведение итогов

Мы рассмотрели, как настроить опрос преобразователя частоты ПЧВ1 [M01] и обсудили основные параметры, характерные для любого подобного прибора:

- слово состояния (битовая маска, описывающая текущие события прибора);
- команда управления прибором (в некоторых ПЧВ она реализована в виде битовой маски, называемой командным словом);
- задаваемая частота.

Также на примере ПЧВ AFD–E мы познакомились с некоторыми специфическими функциями Modbus, которые редко поддерживаются в приборах:

- 0x17 (Read/Write Multiple registers);
- 0x07 (Read Exception Status);
- 0x08 (Read Diagnostics).

2.16 Тестовые задания

Ответы приведены в п. 9.

1. Какие резисторы используются для предотвращения отражения сигнала от конца линии связи?
    - Терминирующие
    - Ограничивающие
    - Шунтирующие
    - Подтягивающие

2. Какой номинал сопротивления обычно используется для терминирующих резисторов?
    - 120 Ом
    - 600 Ом
    - 480 Ом
    - 60 Ом
    - 240 Ом

3. Как называются устройства, подключаемые в шину RS–485 для усиления ослабленного сигнала?
    - Конвертеры
    - Повторители
    - Разветвители
    - Концентраторы

4. С точки зрения спецификации Modbus Serial – сколько slave–устройств можно подключить к последовательной линии связи?

2.17 Полезные утилиты

В прошлых практически уроках мы использовали MasterOPC Universal Modbus Server для проверки обмена с устройствами. Это очень удобное ПО, но в некоторых ситуациях его оказывается недостаточно. Примеры подобных ситуаций:

- требуется провести долгосрочное тестирование обмена (занимающее больше часа) с передачей большого количества данных (более 32 тегов). В этом случае триальная версия MasterOPC Universal Modbus Server не подойдет из–за своих ограничений – потребуется приобрести лицензию;
- в рамках тестирования требуется отправлять запросы на запись с заданным периодом (OPC–сервер отправляет запрос на запись только однократно, при изменении значения тега);
- требуется проанализировать, по каким причинам возникают ошибки обмена между уже настроенными устройствами. Для этого требуется «прослушивать» шину, а MasterOPC Universal Modbus Server не поддерживает такой функционал;
- требуется протестировать обмен без наличия реальных устройств (как master'а, так и slave–устройств). Как мы увидим позже – MasterOPC Universal Modbus Server подходит для такого сценария, но вместе с ним потребуется использовать дополнительное ПО.

Modbus Tester

«Modbus Tester» – это обобщенное название утилит для ПК, используемых для тестирования обмена по протоколу Modbus.

Они бывают разные:

- бесплатные / требующие приобретения лицензии;
- кроссплатформенные / работающие только в определённой операционной системе;
- работающие только в режиме master / работающие только в режиме slave / работающие в обоих режимах.

Мы рассмотрим одну из подобных утилит, которая так и называется – **Modbus Tester**. Она была разработана в середине 2000–х годов Вадимом Абрамовым.

Ссылка на утилиту: https://ftp.owen.ru/Soft/ModbusTester–Installer.exe

> *Выражаем благодарность Вадиму Абрамову за разрешение на распространение утилиты.*

Функционал утилиты:

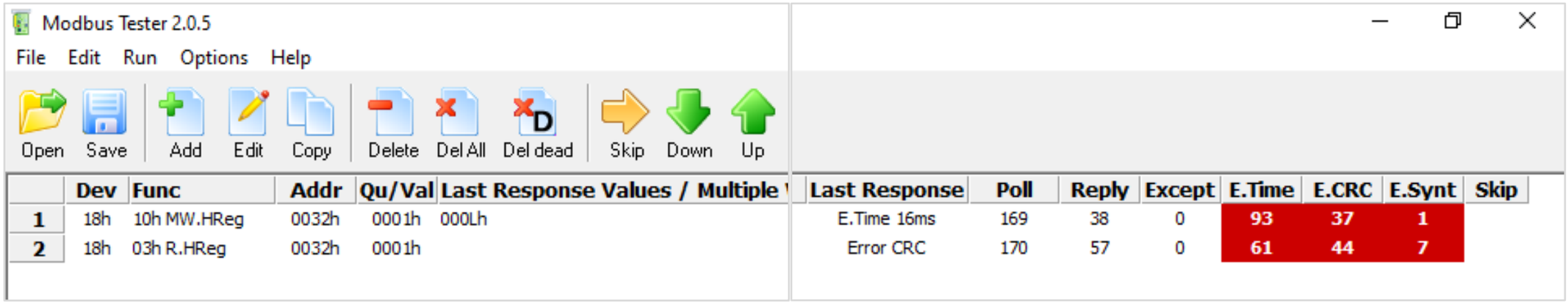
- поддерживается работа по протоколам Modbus RTU, Modbus ASCII и Modbus RTU over TCP в режиме Master (протокол Modbus TCP и работа в режиме Slave не поддерживается);
- отсутствуют ограничения на время работы и число настроенных запросов;
- отображается лог обмена и статистика за время работы (общее количество отправленных запросов, количество отсутствия ответов за время таймаута и т. д.);
- можно проверять обмен в интенсивном режиме (десятки запросов в секунду).

Принцип использования утилиты:

- в меню **Options** выбирается используемый интерфейс – **Serial line mode** (для RS–485 или RS–232) или **TCP/IP client mode**;
- в меню **Options** задаются настройки интерфейса – с помощью кнопки **Serial settings** (для Serial line mode) или **TCP/IP settings** (TCP/IP client mode);
- для **Serial line mode** в меню **Options** выбирается протокол – **Modbus RTU** или **Modbus ASCII**;

- запросы, которые будет отправлять утилита, настраиваются в меню **Edit** (команды **New Request** и **Edit request**) или с помощью ярлыков на панели инструментов. Для каждого запроса указывается адрес slave–устройства, код функции, адрес начального объекта (бита или регистра) и количество объектов в запросе. Для запросов записи можно настроить автоматическое изменение записываемых значений в рамках одного из доступных диапазонов;
- запуск опроса выполняется в меню **Run**. В нём же можно настроить режим опроса (циклически повторять все запросы, однократно выполнить все запросы и т. д.) и режим остановки (например, остановка опроса при получении первой ошибки по таймауту).

Вот так выглядит окно утилиты с двумя настроенными запросами:



Справа отображается статистика – код последней ошибки (**Last Response**), общее число отправленных запросов (**Poll**), количество «нормальных» ответов (**Reply**), количество ответов с кодом ошибки Modbus (**Except**), количество запросов, на которые за время таймаута не был получен ответ (**E.Time**), количество ответов с некорректной контрольной суммой (**E.CRC**) и количество ответов с некорректной структурой пакета (**E.Synt**).

> *Назначение столбца Skip неочевидно; документация на утилиту отсутствует.*

Настроенные запросы можно сохранить с помощью команды **File – Save**.

***

Существует множество утилит с близким функционалом. Из них отметим лишь некоторые:

- Modbus Poll (очень известная, но требующая приобретения лицензии);
- Modbus Guru (бесплатная, от российского разработчика);
- PultModbusTester (бесплатная, от разработчиков российской SCADA–системы Пульт.Онлайн);
- ModbusTools (бесплатная и кроссплатформенная);
- ModbusMechanic (бесплатная и кроссплатформенная);
- RILHEVA MODBUS POLL (бесплатная);
- Radzio! (бесплатная);
- Termite (бесплатная, от российских разработчиков).

Вы можете выбрать те из них, которые кажутся вам наиболее функциональными и удобными (или поискать в интернете другие варианты), и использовать их в своей деятельности.

Терминал COM–порта

Терминал COM–порта позволяет «прослушивать» последовательную линию связи и отправлять по ней произвольные наборы байт.

> *Когда говорят о терминале, работающем в режиме прослушивания, то его обычно называет «сниффером» (sniffer; «вынюхивающий»).*

В рамках настройки обмена по Modbus эти утилиты полезны в двух ситуациях:

- когда нужно разобраться, как происходит обмен в уже настроенной системе, к устройствам которой нет доступа (или этот доступ не предоставляет всей нужной информации). Терминал покажет все запросы и ответы, передаваемые по шине. В большинстве случаев у терминалов нет встроенного анализатора протокола Modbus, но вы можете воспользоваться для этой цели уже известным нам онлайн–парсером;
- когда нужно сгенерировать некорректный Modbus–пакет, чтобы проверить какую–либо гипотезу (например, гипотезу о том, что slave–устройство, работающее по протоколу Modbus ASCII, требует, чтобы в запросе не было стоп–символов; это, конечно, противоречит спецификации Modbus, но с таким устройством можно столкнуться в реальной жизни).

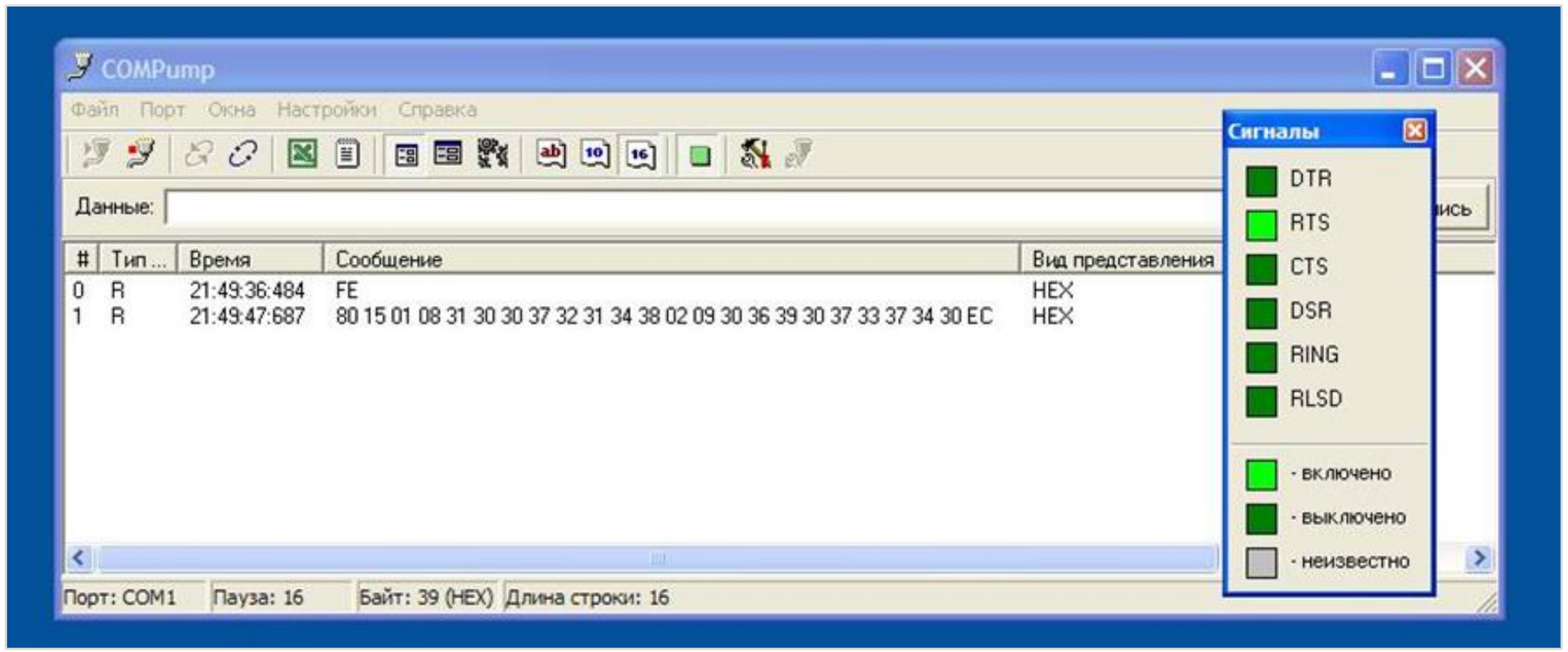
Из множества существующих утилит–терминалов стоит отметить следующие (все они являются бесплатными):

- Hercules Setup Utility;
- Terminal 1.9b;
- COMPump.

Доступные варианты для Linux описаны в этой статье:

https://developer.toradex.com/software/development–resources/serial–terminal–emulator/

(добавим к их списку tio)



Утилиты для создания виртуальных COM–портов

В прошлых уроках мы опрашивали реальные устройства, подключая их к ПК с помощью конвертера RS–485/USB AC4–M. Драйвер конвертера создавал на ПК виртуальный COM–порт, номер которого мы указывали в MasterOPC Universal Modbus Server.

Но в некоторых ситуациях под руками может не оказаться реальных устройств или конвертера.

В этом случае можно полностью проэмулировать обмен на ПК, воспользовавшись одной из утилит для создания виртуальных COM–портов.

Одной из таких утилит является **Virtual Serial Ports Emulator (VSPE)**. Её можно использовать без приобретения лицензии; в этом случае будет действовать ограничение: не более 5 одновременно созданных виртуальных «устройств»; кроме того, не получится подгружать сохранённые ранее файлы конфигурации. Для отладки обмена эти ограничения не являются существенными.

Другими примерами подобных утилит являются:

- Tibbo Device Server Toolkit;
- socat (для ОС Linux; пример использования).

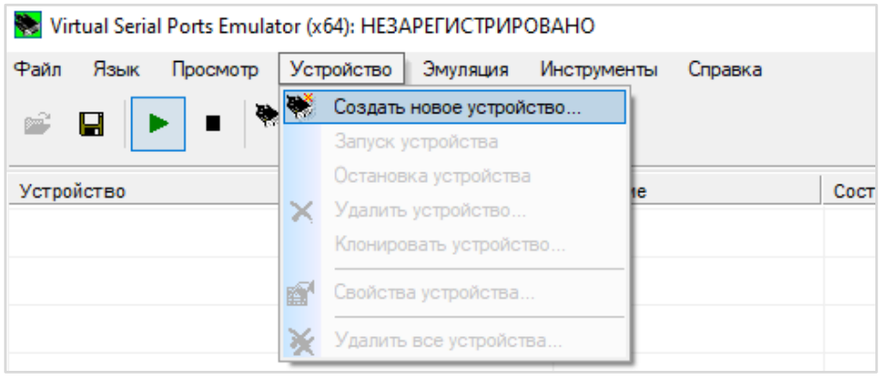
Давайте с помощью утилиты VSPE проэмулируем обмен без реальных устройств.

Настройка VSPE

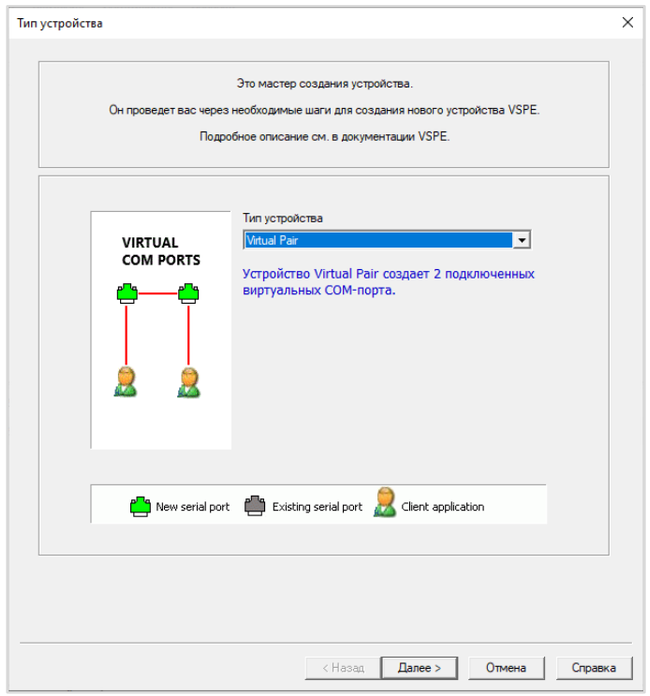
Установите и запустите VSPE:

https://eterlogic.com/Products.VSPE_Download.html

Нажмите кнопку **Устройство – Создать новое устройство**:



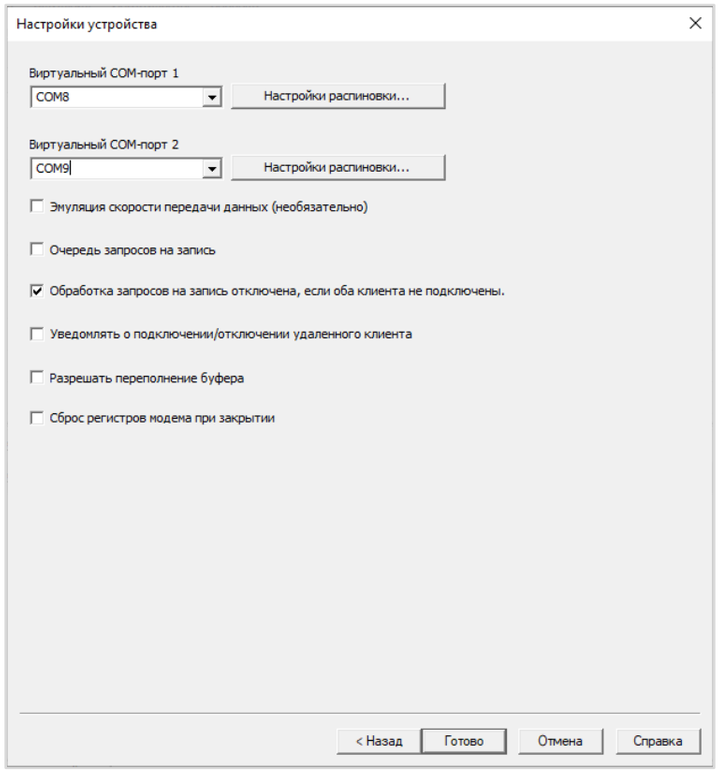
Выберите тип устройства **Virtual Pair** (пара виртуальных COM–портов, соединённых друг с другом).



Укажите номера для создаваемых виртуальных COM–портов.

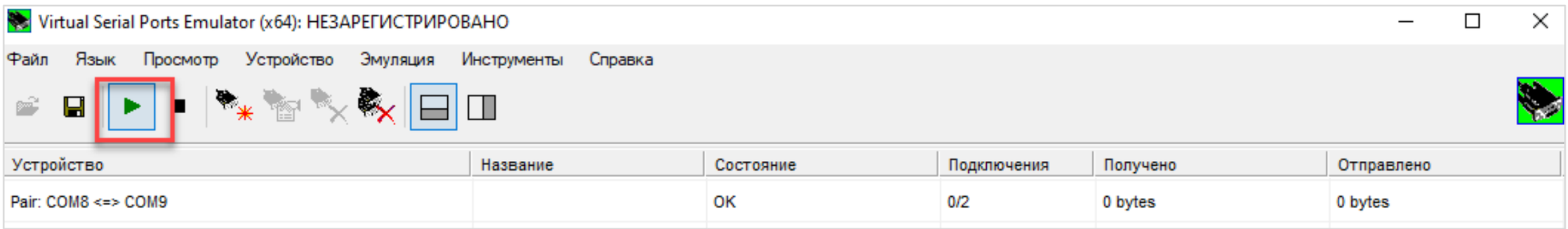
Эти номера не должны использоваться вашей операционной системой – сначала убедитесь, что они отсутствуют в диспетчере устройств на вкладке **Порты (COM и LPT)**. После нажатия на кнопку **Готово** созданные порты можно будет использовать (*даже если они не добавятся в диспетчере устройств – так происходило в ранних версиях утилиты*).

На нашем ПК мы используем номера COM8 и COM9.



Созданное устройство отобразится в интерфейсе утилиты.

Убедитесь, что утилита находится в режиме запуска:



Настройка обмена в MasterOPC Universal Modbus Server

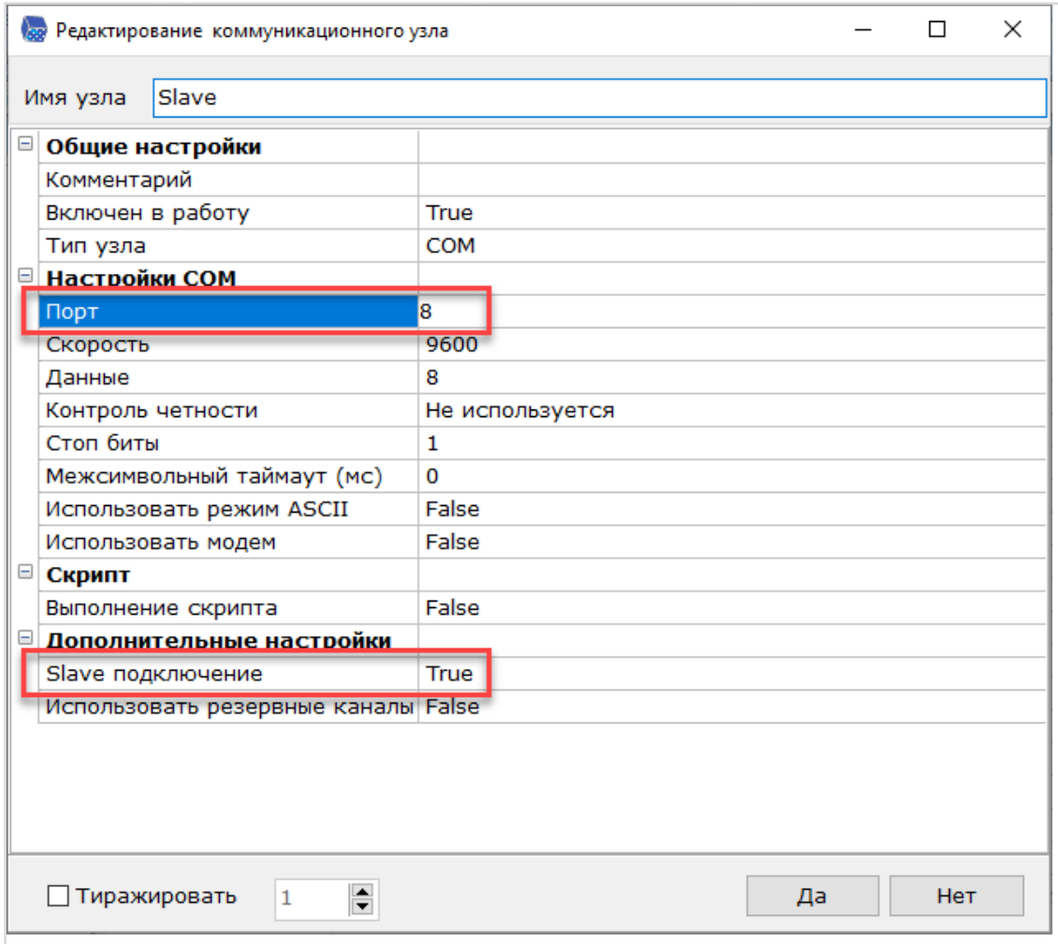
Создайте в MasterOPC Universal Modbus Server два коммуникационных узла.

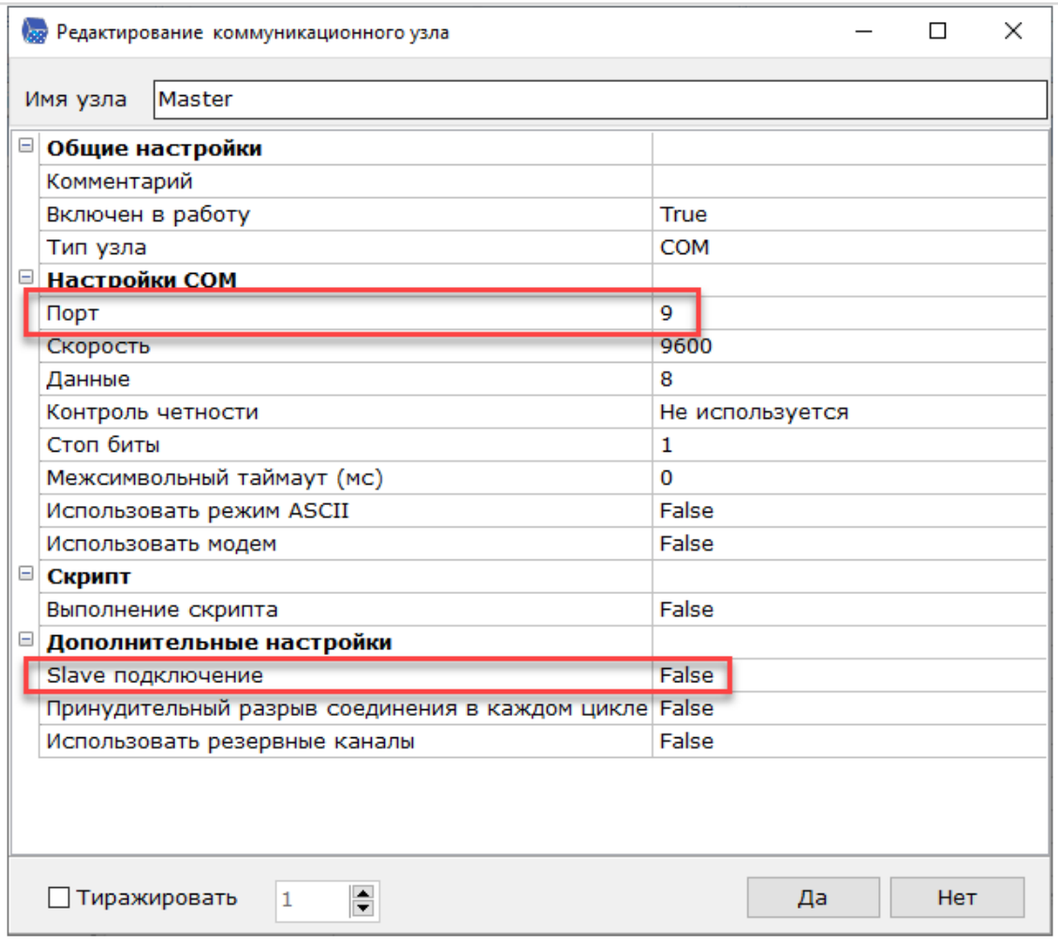
Один из узлов назовите **Slave**, другой – **Master**.

В настройках первого укажите **Slave подключение** = **True**, для второго = **False**.

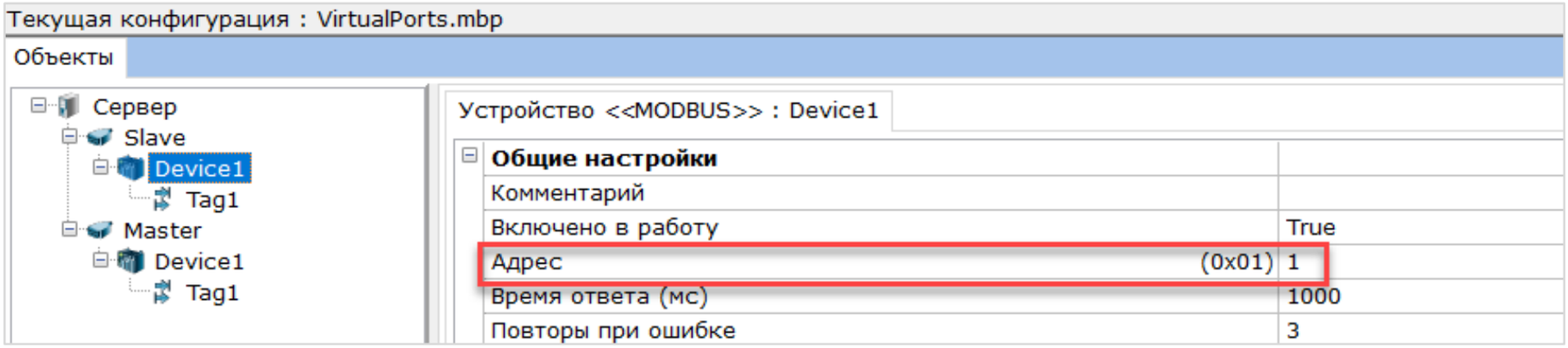
Для обоих узлов укажите тип узла – **COM** и введите номера виртуальных COM–портов, созданных на предыдущем шаге.

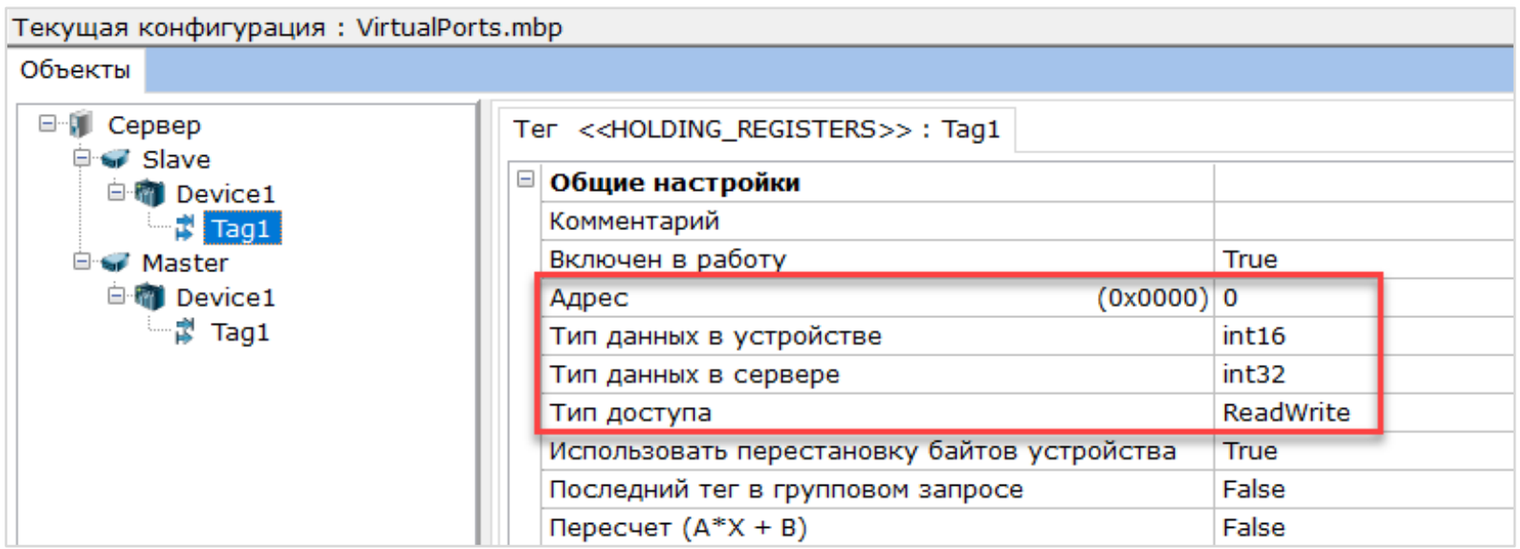
Настройки COM–портов не имеют значения, так как они являются виртуальными (мы исходим из предположения, что на прошлом шаге при создании устройства вы не установили галочку **Эмулировать скорость**).





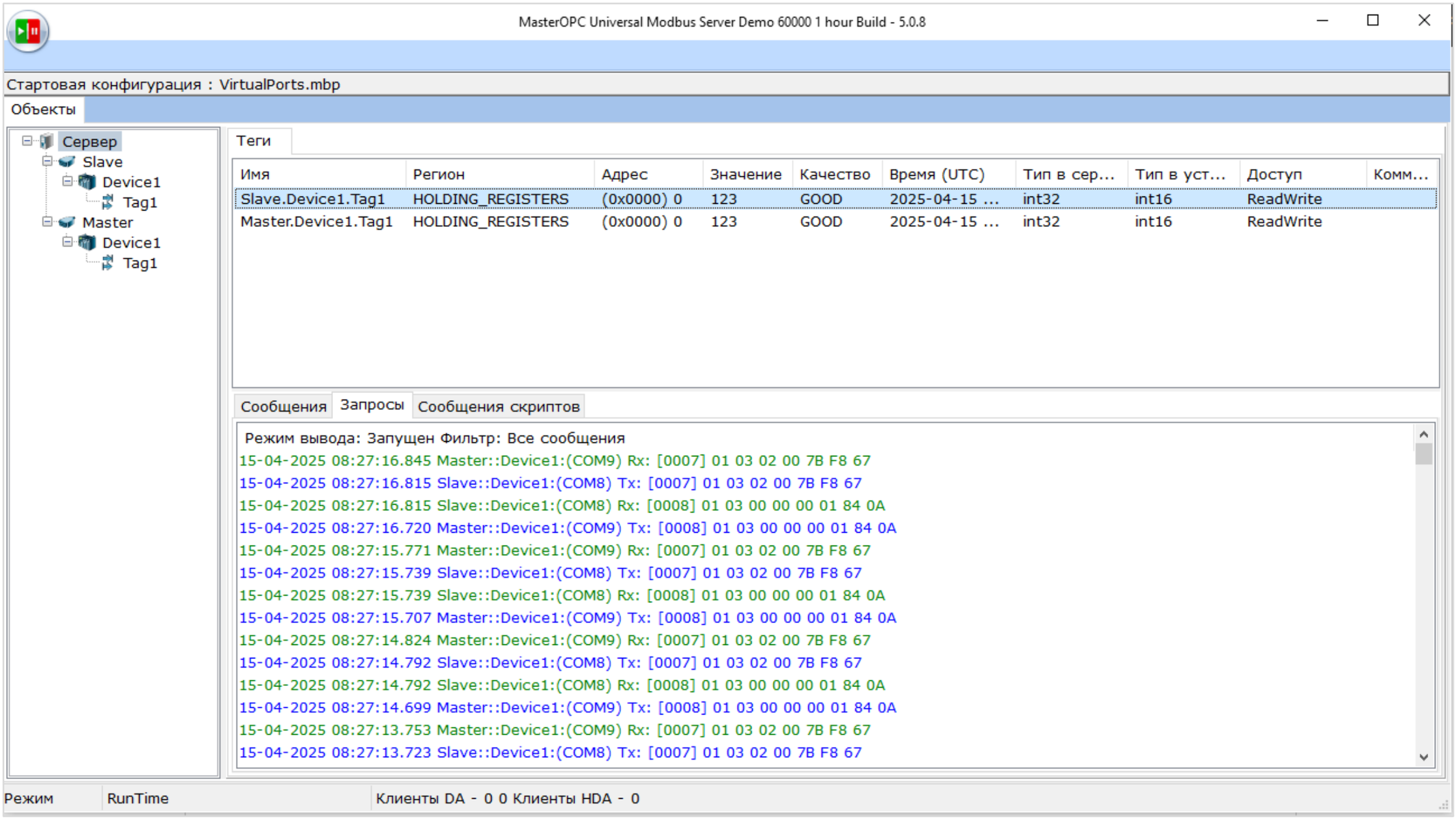
Добавьте в каждый узел устройство с одним и тем же адресом (например, 1), а в него – тег с одними и теми же настройками.





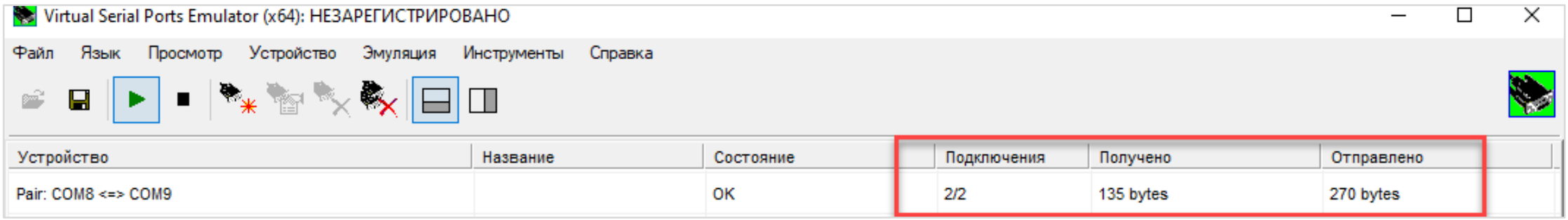
Запустите OPC–сервер.

Задайте любому из тегов новое значение. Если вы зададите значение тегу из узла **Slave** – то это значение будет считано узлом **Master** и помещено в его одноименный тег. Если вы зададите значение тегу из узла **Master** – то оно будет записано в одноименный тег узла **Slave**.



В интерфейсе VSPE в этот момент можно увидеть, что к обоим портам Virtual Pair произошло подключение (их используют коммуникационные узлы OPC–сервера), и увеличение счётчиков полученных и отправленных данных.

> *У нас нет объяснения, почему для нашего случая счётчик отправленных байт в 2 раза превышает значение счётчика принятых байт; это противоречит нашим ожиданиям, так как из лога OPC–сервера видно, что запрос на чтение занимает 7 байт, а ответ на него – 8 байт. Соответственно, мы ожидали, что значения счётчиков будут иметь пропорцию 8/7.*



Подведение итогов

Мы рассмотрели три основных категории утилит, имеющих ценность при отладке обмена по Modbus Serial:

- «Modbus Tester'ы» (на примере утилиты **Modbus Tester**);
- терминалы COM–порта;
- утилиты для создания виртуальных COM–портов (на примере утилиты **VSPE**).

С помощью утилиты VSPE мы создали на ПК пару соединённых друг с другом виртуальных COM–портов и организовали обмен по протоколу Modbus RTU без наличия каких–либо реальных устройств с помощью MasterOPC Universal Modbus Server.

Вы можете использовать виртуальные COM–порты совместно с эмуляторами контроллеров, панелей оператора и других устройств, которые поддерживают протокол Modbus.

В следующем уроке мы опять перейдём к практике и настроим опрос модулей ввода–вывода линейки Mx110 от компании ОВЕН.

2.18 Практика: опрос модулей Mx110

Mx110 – это линейка модулей ввода–вывода компании ОВЕН с интерфейсом RS–485 и поддержкой протоколов Modbus RTU и Modbus ASCII.

https://owen.ru/catalog/moduli_vvoda_vivoda/info/general_information_Mx110

Она включает в себя больше 20 модификаций модулей с различными типами входных и выходных сигналов:

- аналоговые входы;
- аналоговые выходы;
- дискретные входы;
- дискретные выходы;
- совмещённые дискретные входы и выходы;
- тензометрические входы (для систем измерения веса);
- входы для измерения параметров электрической сети (сила тока, напряжение, мощность и т. д.).

В рамках данного урока мы настроим опрос двух модулей:

- модуля аналогового ввода МВ110–8А;
- модуля дискретного вывода МУ110–16Р.

Эти модули подключены по топологии «шина»:

- клеммы A и B интерфейса RS–485 модуля МВ110–8А соединены с одноимёнными клеммами модуля МУ110–16Р;
- клеммы A и B интерфейса RS–485 модуля МВ110–16Р соединены с одноимёнными клеммами конвертера АС4–М, который подключен к ПК по интерфейсу USB.

Настройка модулей

Мы используем обновленные модули аппаратной ревизии H/W v.2.0.

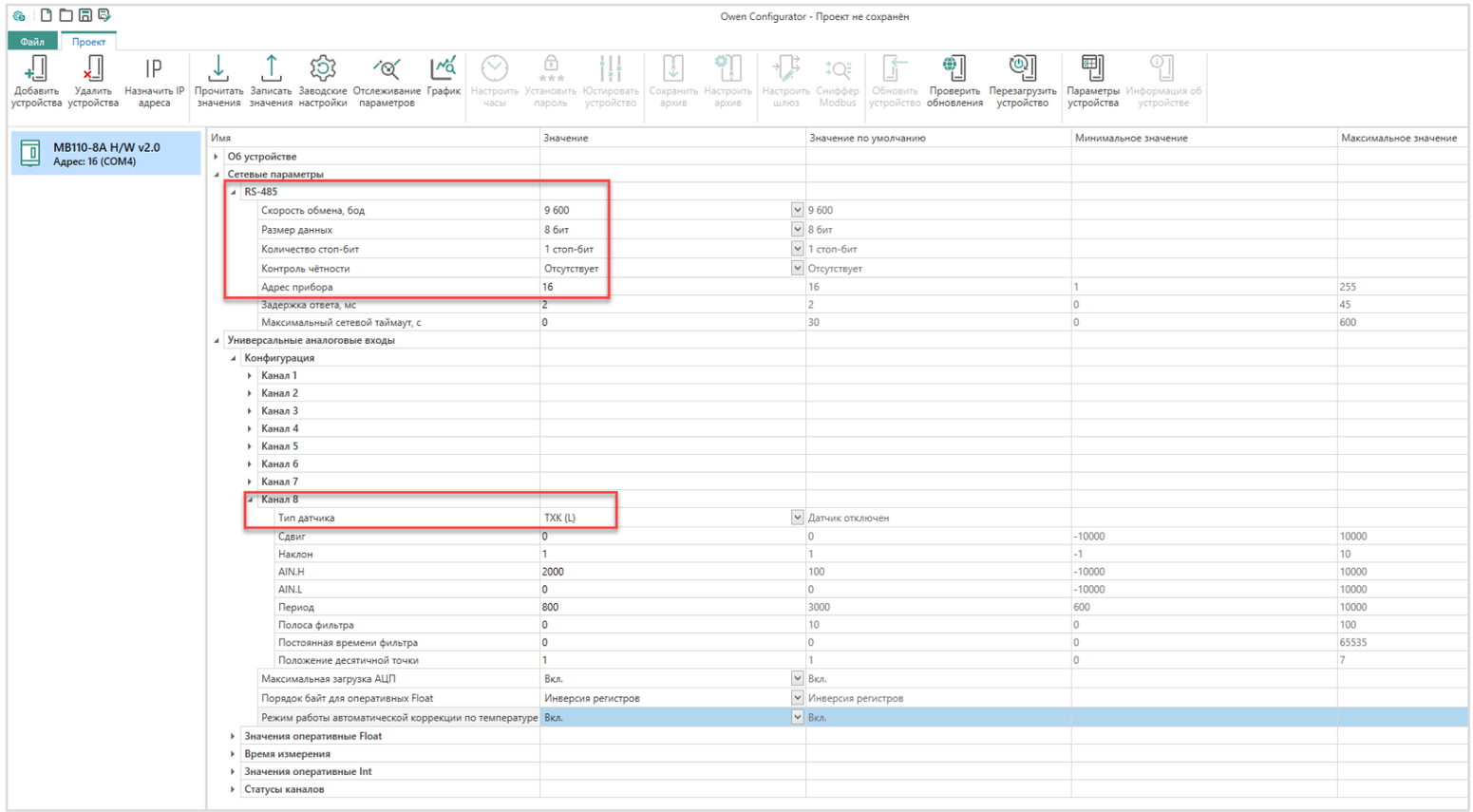
Номер ревизии указан на боковой грани модуля. Также на лицевой крышке обновленных модулей присутствует надпись «Настройка в новом конфигураторе» и QR–код со ссылкой на этот конфигуратор.

Под «новым конфигуратором» подразумевается утилита OWEN Configurator, используемая для настройки большинства современных приборов ОВЕН. Мы использовали её в уроке 2.9, когда подключались к ТРМ1–У3.

Мы используем модули со следующими настройками:

Модуль	Сетевые настройки	Адрес slave–устройства	Другие параметры
МВ110–8А	9600–8N1	16	Ко входу 8 подключена перемычка В настройках входа выбран тип датчика ТХК (L)
МУ110–16Р	9600–8N1	24	

Как и в случае с ТРМ – мы используем перемычку на аналоговом входе одновременно с выбором «термопарного» датчика, чтобы сымитировать значение (оно будет близко к значению комнатной температуры).



Карта регистров

Из документации на модули можно узнать, что они поддерживают следующие функции Modbus:

- **0x03 (Read Holding Registers)** и **0x04 (Read Input Registers)**. В модулях используется общая модель памяти – то есть все параметры могут быть считаны как функцией **0x03**, так и функцией **0x04**;
- **0x06 (Write Single Register)** и **0x10 (Write Multiple Registers)** для записи holding–регистров.

Фрагмент из руководства на модуль:

8.3 Протокол Modbus

Протокол Modbus поддерживает два режима передачи данных — ASCII или RTU.

Чтение осуществляется функциями 0x03 (Read Holding Registers) или 0x04 (Read Input Registers), запись – 0x06 (Write Single Register) или 0x10 (Write Multiple Registers).

Значение нужного нам 8 аналогового входа модуля МВ110–8А имеет тип **Float** и размещено в его регистрах **46–47**.

Значение на входе FLOAT 1	—	Float32	0x0004	4
Значение на входе FLOAT 2	—	Float32	0x000A	10
Значение на входе FLOAT 3	—	Float32	0x0010	16
Значение на входе FLOAT 4	—	Float32	0x0016	22
Значение на входе FLOAT 5	—	Float32	0x001C	28
Значение на входе FLOAT 6	—	Float32	0x0022	34
Значение на входе FLOAT 7	—	Float32	0x0028	40
Значение на входе FLOAT 8	—	Float32	0x002E	46

Значение выходов модуля МУ110–16Р представлено в виде битовой маски типа Uint16 и размещено в регистре **50**.

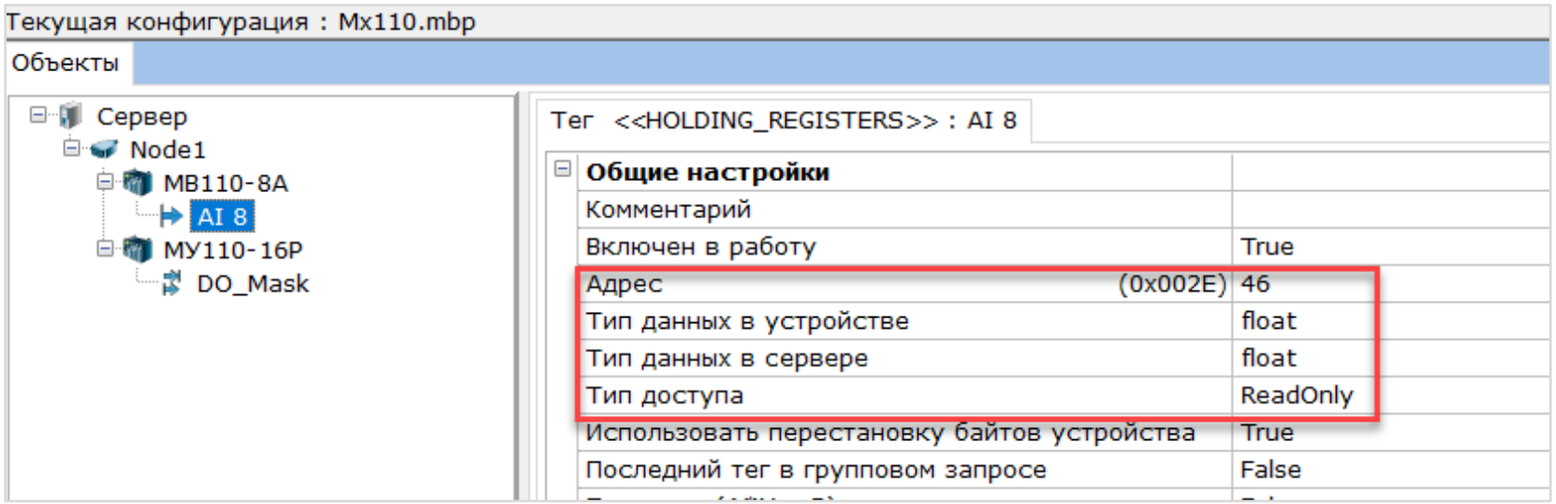
Период ШИМ на выходе 16	1…900 с	Uint16	0x0021	0047
Битовая маска значений выходов	0…65535	Uint16	0x0032	0050

Настройка опроса в MasterOPC Universal Modbus Server

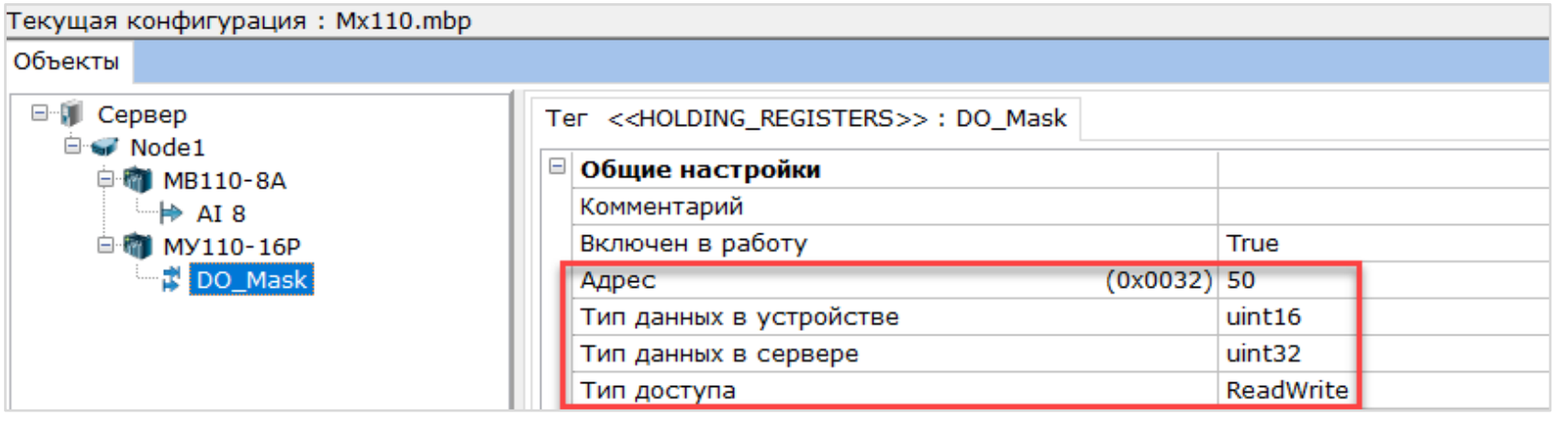
Мы уже не раз настраивали опрос в MasterOPC Universal Modbus Server, поэтому сейчас остановимся только на ключевых моментах:

- создайте коммуникационный узел. Задайте в нём номер виртуального COM–порта, созданного вашим конвертером RS–485/USB, и введите сетевые настройки, совпадающие с настройками модулей (у нас это 9600–8N1);
- добавьте два устройства – MB110–8A с адресом **16** и МУ110–16Р с адресом **24**.

В MB110–8A добавьте тег со следующими настройками:



В МУ110–16Р добавьте тег со следующими настройками:



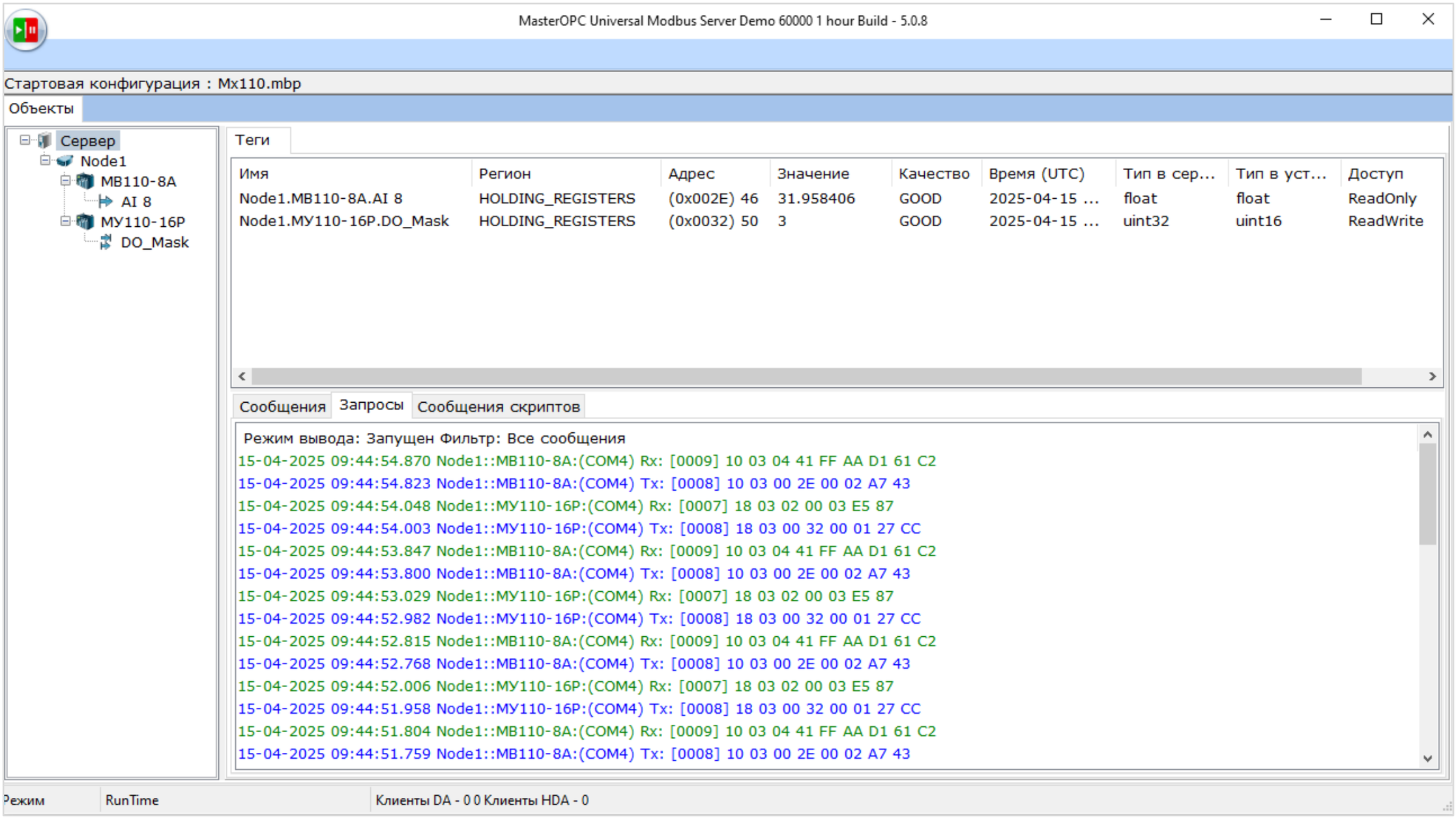
Ссылка на готовую конфигурацию:

https://ftp.owen.ru/ModbusCourse/01_OpcTemplates/Mx110.mbp

Проверка обмена

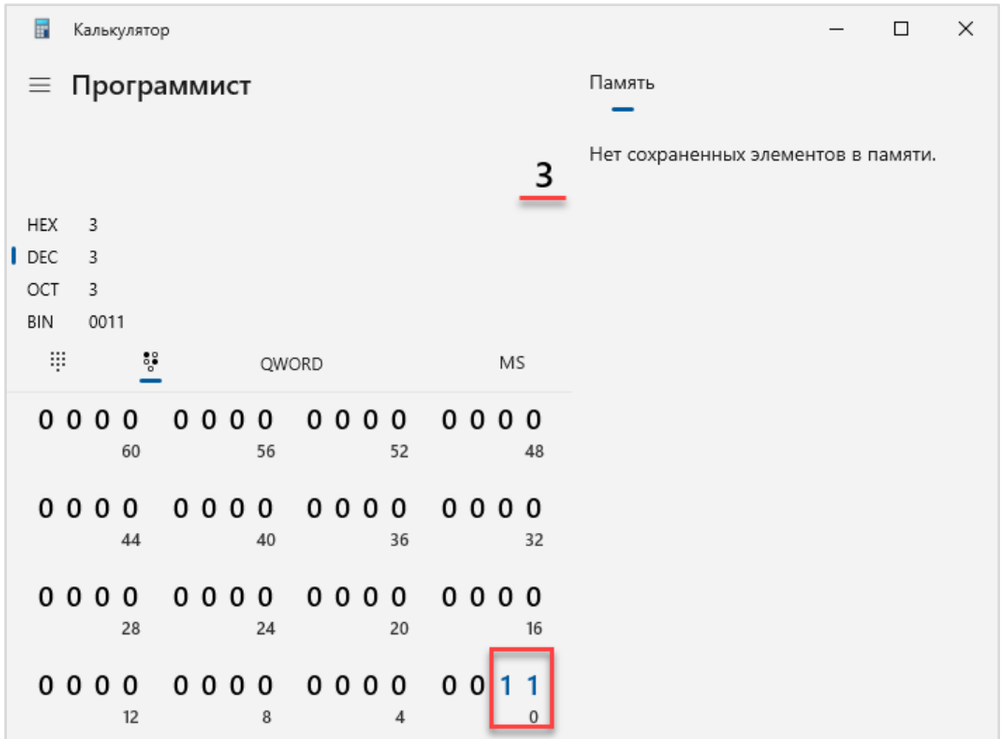
Запустите OPC–сервер.

В теге AI 8 отобразится текущее измеренное значение 8 аналогового входа модуля MB110–8A.



Задайте новое значение тегу **DO_Mask** для переключения выходов модуля.

Значение задаётся в виде битовой маске; соответственно, запись в тег числа 3 приведёт к включению дискретных выходов 1 и 2.



Коды ошибок MB110–8A

Модуль MB110–8A может детектировать наличие ошибок на своих аналоговых входах – например, обрыв датчика, отсутствие связи с АЦП измерительного канала и т. д.

Эти коды ошибок могут быть получены по Modbus двумя способами:

- путём чтения отдельных регистров статуса;
- путём выделения кода ошибки из считанного Float–значения.

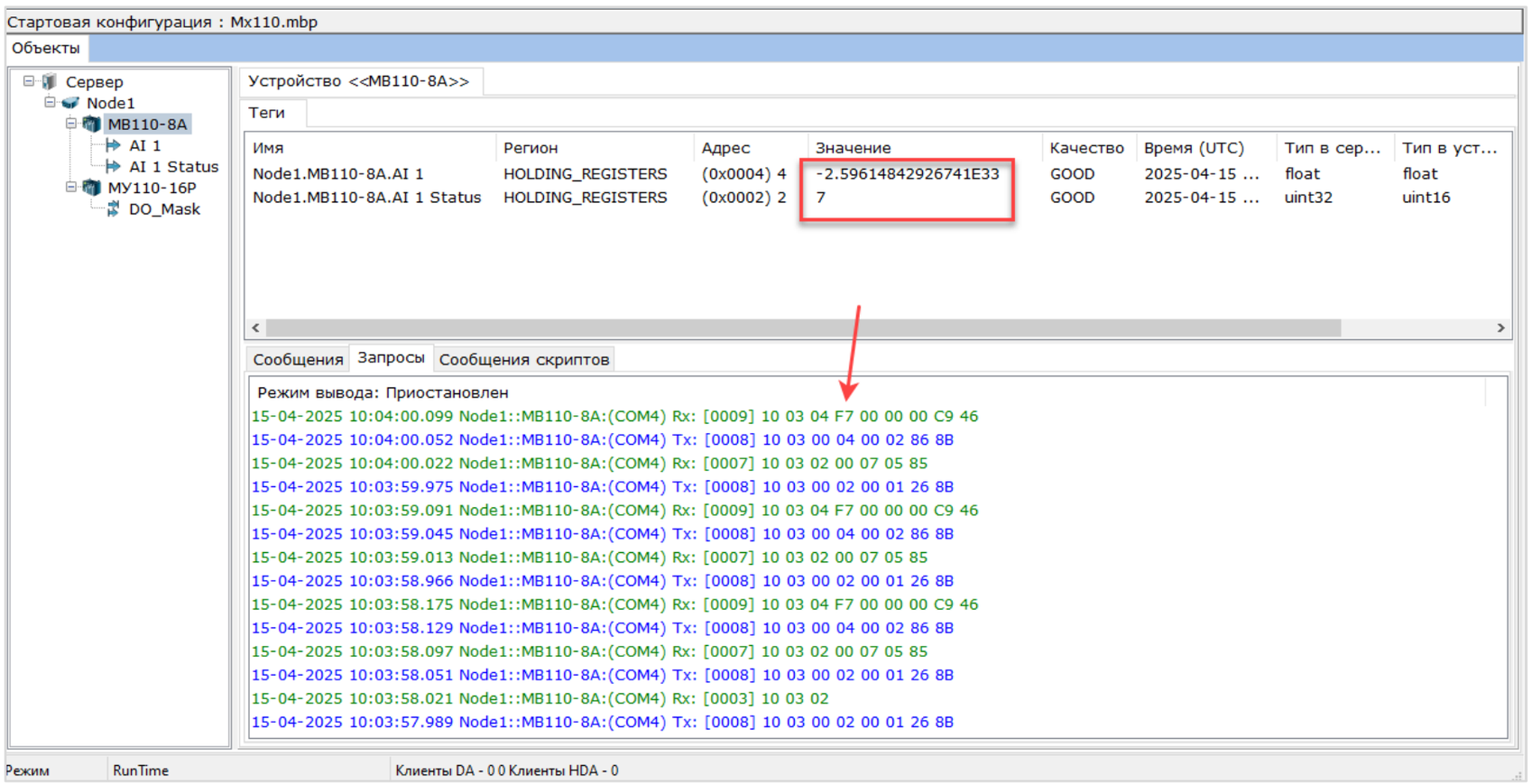
Первый способ не должен вызывать вопросов.

Второй способ основан на том факте, что в случае возникновения ошибки её код записывается в старший байт Float–значения.

Давайте рассмотрим это на примере.

У нашего модуля к первому аналоговому входу не подключено никакого датчика, и в настройках в параметре **Тип датчика** выбрано значение **Датчик отключен**. Считаем значение первого аналогового входа (регистры 4–5) и его регистра статуса (регистр 2).

В теге значения мы увидим необычное число, а в теге статуса – значение **7**.



Если мы изучим лог обмена, то увидим, что значение аналогового входа в «сыром» виде выглядит следующим образом:

> *0xF7000000*

Его старший байт равен **0xF7**.

Всё это соответствует информации из руководства на модуль:

Если произошла исключительная ситуация (например, обрыв датчика), то при исправном приборе происходит передача специализированного пакета.

В случае передачи кода исключительной ситуации во время обмена по протоколу ОВЕН передается пакет, в поле данных которого идет однобайтовая посылка. Байт содержит первые 4 бита равные 1, вторые 4 бита содержат код исключительной ситуации.

В случае возникновения ошибки на входе в регистр статуса входа и в старший байт регистра значения (тип float) передается код ошибки (см. таблицу 8.3).

Таблица 8.3 – Коды статусов входа

Статус входа	Для протокола Modbus и ОВЕН: значение в старшем байте регистра измерения (тип Float 32)	Для протокола Modbus: значение в регистре статуса
Измерение успешно	—	0x00
Значение заведомо неверно	0xF0	—
Данные не готовы. Следует дождаться результатов первого измерения после включения прибора	0xF6	0x06
Датчик отключен	0xF7	0x07
Велика температура свободных концов ТП	0xF8	0x08
Мала температура свободных концов ТП	0xF9	0x09

Другие интересные моменты, связанные с модулями MB110–8A

1. Модулю может быть назначен адрес **1...255,** что является незначительным отступлением от диапазона **1...247**, указанным в спецификации Modbus.

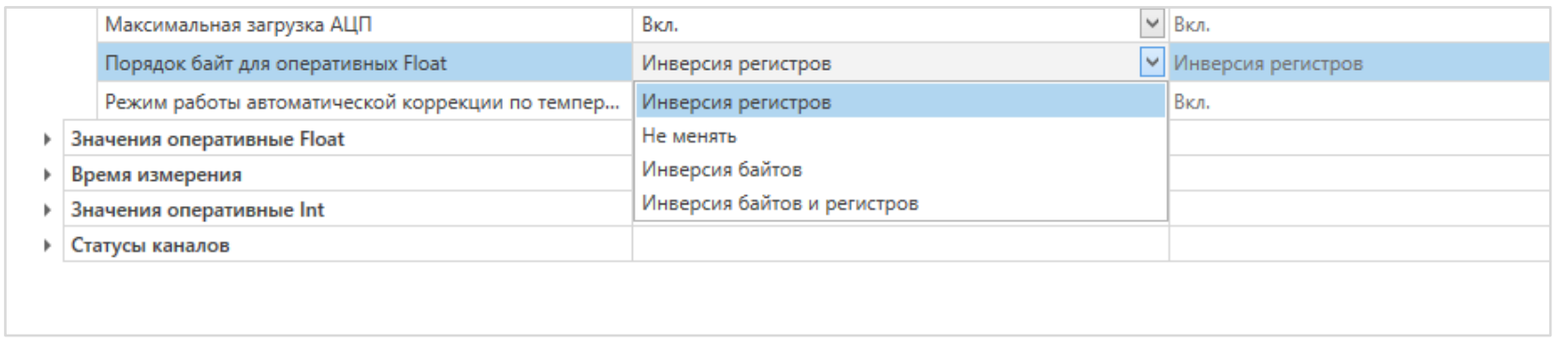
Протокол Modbus	
Диапазон значений базового адреса	от 1 до 255

2. У модулей имеется параметр **Rs.dl** (Задержка ответа по RS–485).

Как мы уже неоднократно упоминали – эта задержка может потребоваться, чтобы master–устройство успело переключить свой COM–порт из режима передачи в режим приёма.

3. У модуля MB110–8A имеется параметр **Порядок байт для оперативных Float**.

Он позволяет «подстроить» порядок байт и регистров в отправляемых значениях типа Float под порядок master–устройства. Это крайне полезно в тех случаях, когда master–устройство не имеет аналогичной настройки.



4. У модулей выходов имеются параметры **Максимальный сетевой тайм–аут** и **Безопасное состояние выхода №**.

Первый параметр определяет интервал времени – если в течение него модулю не поступает ни одного запроса от master–устройства, то выходы модуля переключаются в безопасное состояние. Безопасное состояние настраивается индивидуально для каждого выхода. Для модуля МУ110–16Р эти параметры имеют тип Uint – что позволяет в безопасном состоянии использовать их в режиме ШИМ с заданным коэффициентом заполнения.

2.19 Тестовые задания

Ответы приведены в п. 9.

1. В уроке 2.17 мы с помощью утилиты VSPE мы создали на ПК **два** виртуальных COM–порта и организовали обмен по протоколу Modbus RTU с помощью MasterOPC Universal Modbus Server. В рамках примера master–устройство опрашивало одно slave–устройство. Сколько виртуальных COM–портов нам потребуется для опроса 4 slave–устройств?
   - 2
   - 8
   - 5
   - 4

2. Как называется программа–терминал, работающая в режиме прослушивания?

2.20 Обзор Modbus TCP

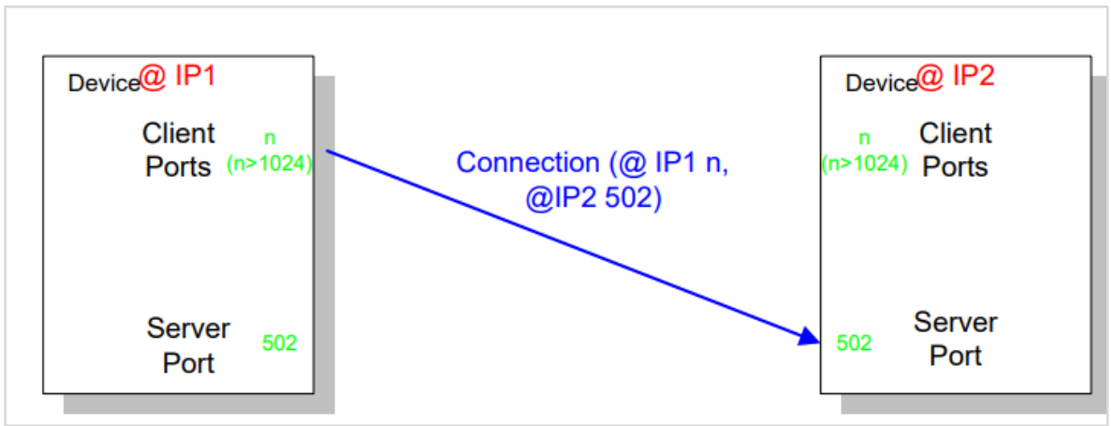
В прошлых уроках мы говорили о протоколах Modbus RTU и Modbus ASCII, используемых поверх последовательных интерфейсов связи (в основном – RS–485).

Теперь мы рассмотрим протокол Modbus TCP, реализуемый поверх стека протоколов TCP/IP; в качестве интерфейса чаще всего используется Ethernet (реже – Wi–Fi и другие технологии). Многие аспекты, которые мы обсудим далее, касаются именно TCP/IP.

Протокол TCP основан на концепции **соединений**. Для начала сеанса обмена master–устройство (клиент) устанавливает соединение со slave–устройством (сервером). Для установки соединения master–устройству требуется знать IP–адрес slave–устройства и номер его сетевого порта, выделенный для обмена по Modbus.

> *Сетевой порт – это идентификатор конкретного сервиса (службы), запущенного на сервере. Он позволяет организовать обмен данными именно с этим интересующим вас сервисом.*

Согласно спецификации – для протокола Modbus TCP должен использоваться порт **502**. Некоторые slave–устройства позволяют настроить используемый номер порта. На стороне master–устройства – номер его используемого порта выбирается произвольным образом из незадействованных и не входящих в диапазон «общеизвестных» (1...1023); обычно он неизвестен пользователю и не представляет для него интереса.



В большинстве реализаций master–устройство устанавливает соединение со slave–устройством в момент своего запуска и держит его открытым в течение всего времени работы; это соответствует рекомендации из спецификации Modbus TCP ([3, п. 4.2.1.1]. Мы обсудим различные ситуации, связанные с закрытием соединений, в одном из следующих шагов.

В рамках Modbus TCP количество участвующих в обмене устройств не имеет явных ограничений; косвенные ограничения связаны с программными и аппаратными ограничениями конкретных устройств, а также количеством IP–адресов, доступных в рамках конкретной сети.

Кроме того, некоторые устройства поддерживают:

- одновременную работу в обоих режимах – и как master, и как slave;
- в режиме slave – несколько одновременных соединений с master–устройствами.

Некоторые устройства имеют специфические ограничения, связанные с реализацией используемого в них сетевого стека. Например, контроллеры ОВЕН ПЛК1xx в режиме Slave поддерживают несколько одновременных соединений, но при этом для каждого соединения нужно указать отдельный номер порта (например – 502, 503 и т. д.). С другой стороны, большинство современных контроллеров позволяют обрабатывать несколько отдельных соединений в рамках одного TCP–порта.

Таким образом, при знакомстве с новым устройством имеет смысл изучить следующие вопросы:

- для режима master: какое ограничение на количество одновременно опрашиваемых slave–устройств?
- для режима slave: сколько одновременных соединений можно установить с данным устройством?
- может ли данное устройство одновременно работать и в режиме master, и в режиме slave?

Структура пакета Modbus TCP

Структура пакета Modbus TCP крайне близка к структуре пакета Modbus RTU и имеет только два отличия:

- отсутствует контрольная сумма (потому что целостность пакета обеспечивается стеком TCP/IP, и расчёт контрольной суммы происходит на его стороне);
- присутствует заголовок, называемый MBAP Header.

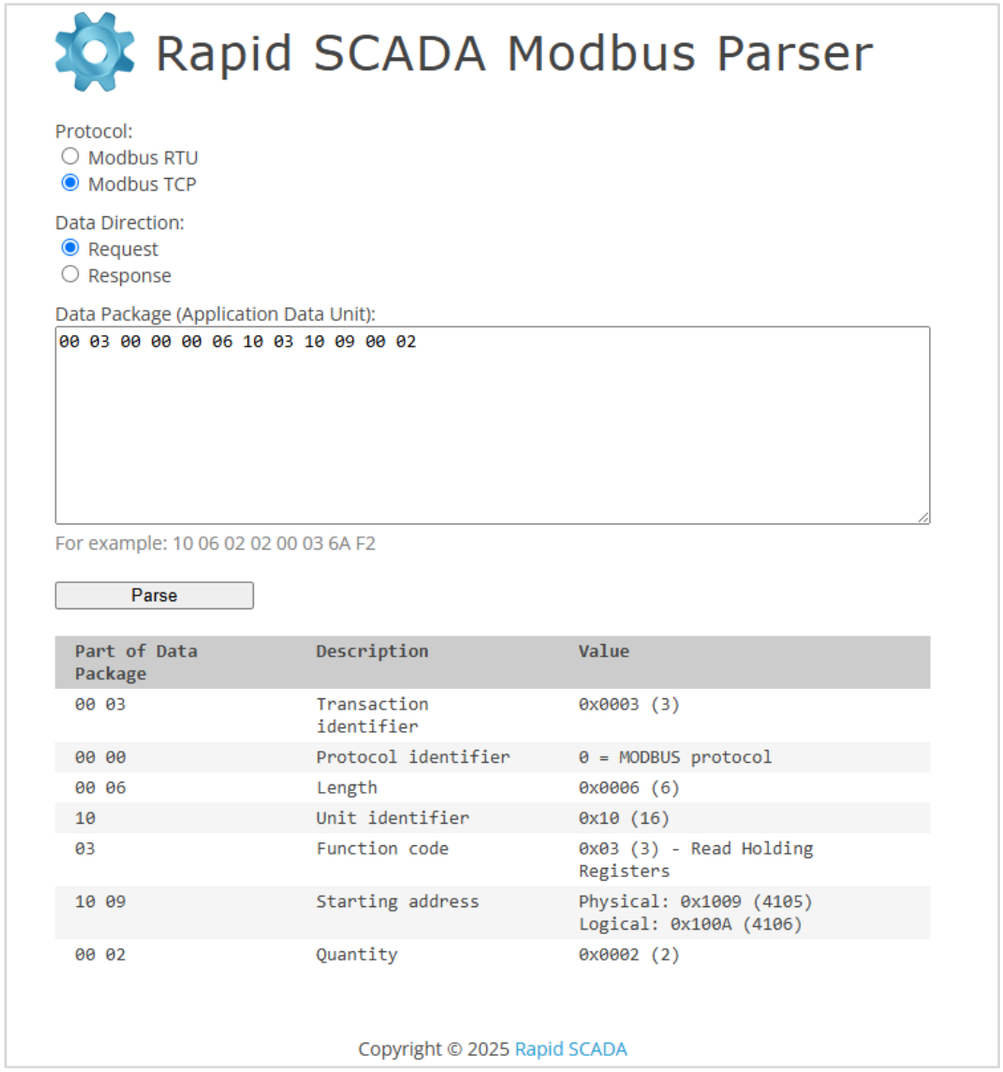
> *MBAP означает Modbus Application Protocol.*

Рассмотрим пакет протокола Modbus RTU:

> *10 03 10 09 00 02* ***13 88***

Соответствующий ему пакет протокола Modbus TCP будет выглядеть следующим образом:

> ***00 03 00 00 00 06 10*** *03 10 09 00 02*



Первые 7 байт пакета являются его заголовком, включающим в себя 4 поля:

- 0003h – идентификатор транзакции (2 байта);
- 0000h – идентификатор протокола (2 байта);
- 0006h – число байт далее (2 байта);
- 10h – адрес slave–устройства (1 байт).

Поля заголовка (MBAP Header)

1. Идентификатор транзакции (Transaction Identifier)

В рамках Modbus TCP обычно воспроизводится модель обмена Modbus Serial:

- master–устройство отправляет запрос и ждёт ответа от slave–устройства;
- после получения ответа или истечения периода таймаута master–устройство отправляет следующий запрос.

Технически TCP/IP позволяет отступить от этого принципа и отправить сразу несколько запросов slave–устройству. Slave–устройство, соответственно, отправит несколько ответов; они могут быть отправлены одновременно или с небольшими паузами, но так или иначе – порядок их получения непредсказуем. Поэтому возникает вопрос: как установить соответствие между конкретным запросом и конкретным ответом?

Именно для этой цели предназначен идентификатор транзакции.

В типовой реализации [2, п. 4.4.1.2]: при каждом запросе master–устройство должно инкрементировать значение этого поля. Slave–устройство сохраняет его значение в своём ответе – и это позволяет установить соответствие между запросом и ответом.

Когда идентификатор транзакции получает значение **FFFFh** – то он переполняется, и следующая транзакция будет иметь идентификатор **0000h**.

2. Идентификатор протокола (Protocol Identifier)

Это поле зарезервировано для «внутрисистемного мультиплексирования» (что бы это ни значило).

В реальной жизни оно всегда имеет значение **0000h** (и в запросе, и в ответе), что соответствует протоколу Modbus TCP.

3. Число байт далее (Length)

Это поле содержит число байт, размещённых следом за этим полем. Оно вычисляется master–устройством при формировании запроса и slave–устройством при формировании ответа.

4. Адрес slave–устройства (Unit Identifier)

В рамках Modbus TCP slave–устройство однозначно идентифицируется IP–адресом и номером TCP–порта – поэтому, казалось бы, никакой дополнительный «адрес устройства» не нужен.

Но это поле сохранено для совместимости с протоколами Modbus RTU / Modbus ASCII и используется для обмена со шлюзами Modbus TCP/Modbus Serial, о которых мы поговорим в модуле 5. Это поле позволяет указать адрес slave–устройства (slave id), расположенного за шлюзом.

Спецификация Modbus TCP указывает [2, п. 4.4.1.2], что при обмене с «обычными» (не являющимися шлюзами) устройствами данное поле должно быть установлено в значение **FFh** или **00h**. Тем не менее, некоторые slave–устройства отступают от спецификации и, например, требуют, чтобы это поле имело именно значение **01h** или какое–то иное.

Обработка разрыва соединения

Мы уже упоминали, что в большинстве реализаций master–устройство устанавливает соединение со slave–устройством в момент своего запуска и держит его открытым в течение всего времени работы.

И master–, slave–устройство может инициировать закрытие соединения, отправив другому устройству пакет с флагом **FIN** (нормальное завершение соединения) или **RST** (экстренное завершение соединения); эти флаги касаются не Modbus TCP, а протокола TCP. Например, OPC–сервер может отправить пакет с флагом FIN, когда пользователь решит закрыть его.

Но если slave–устройство перезагрузится по питанию – то master–устройство не получит никаких пакетов и будет держать соединение активным, тратя на это свои ресурсы. Такое «подвисшее» соединение называется «полуоткрытым» (half–open).

Master–устройству достаточно легко определить разрыв соединения – оно сделает это при неуспешной попытке отправить очередной запрос.

Slave–устройству сделать это гораздо сложнее; отсутствие запросов от master–устройства может являться нормальной ситуацией, если запросы отправляются не циклически, а по каким–то событиям. Поэтому в разумно спроектированных slave–устройствах есть настройка **Таймаут безопасного состояния**, которую мы уже упоминали ранее. Применительно к Modbus TCP – после истечения этого времени нужно не только выполнить заданные операции (например, перевести выходы устройства в безопасное состояние, обнулить значения каких–то регистров и т. д.), но и закрыть текущее соединение, чтобы избежать утечки ресурсов. Спецификация Modbus TCP рекомендует использовать для реализации этого таймера механизм **Keep Alive** протокола TCP.

Более подробная информация по рассматриваемым вопросам приведена в [2, п. 4.2.2 и 4.3.2]

Широковещательный запрос

Так как в протоколе Modbus TCP обмен между master– и slave–устройством происходит в рамках установленного между ними соединения, то возможность отправки широковещательного запроса отсутствует.

Протокол Modbus TCP Security

Спецификация протокола Modbus TCP Security была выпущена в 2018 году. Она разработана для систем, в которых требуется обеспечить конфиденциальность и подлинность передаваемых данных. Для этой цели передаваемые пакеты зашифровываются с помощью криптографического протокола TLS. Структура пакетов Modbus TCP при этом сохраняется.

В качестве номера TCP–порта на стороне slave–устройства по умолчанию используется **802**.

Более подробная информация приведена в спецификации протокола:

https://ftp.owen.ru/ModbusCourse/00_ModbusSpecs/MB–TCP–Security–v36_2021–07–30.pdf

В настоящий момент он не получил широкого распространения и поддержан в крайне ограниченном числе приборов и программ.

Протокол Modbus UDP

Некоторые приборы и ПО позволяют использовать протокол **Modbus UDP**; этот протокол не описан в спецификации Modbus и поддержан ограниченным числом устройств.

Его единственным отличием от Modbus TCP является то, что вместо стека TCP/IP он использует стек UDP/IP. За счёт этого:

- в протоколе Modbus UDP нет понятия «соединения». Запросы и ответы отправляются в рамках UDP–пакетов без гарантии их доставки (TCP обеспечивание контроль доставки пакетов);
- технически возможна отправка широковещательных запросов (UDP broadcast и UDP multicast).

Познакомиться с мнением различных технических специалистов об этом протоколе можно в теме на форуме ОВЕН:

https://owen.ru/forum/showthread.php?t=37634

Протокол Modbus RTU over TCP

Протокол **Modbus RTU over TCP** не описан в спецификации Modbus, но с ним вполне можно встретиться на практике.

Самое главное – не путать его с Modbus TCP:

- Modbus TCP – это отдельный от Modbus RTU **протокол**, имеющий отличия в структуре пакетов (наличие MBAP Header, отсутствие контрольной суммы);
- Modbus RTU over TCP – это **принцип** передачи пакетов протокола Modbus RTU (без MBAP Header и с контрольной суммой) по сети TCP. Он используется совместно с конвертерами интерфейсов COM/Ethernet, которые мы рассмотрим в модуле 5.

Публичный тестовый Modbus TCP Slave

- Разработчики российской SCADA–системы Пульт.Онлайн создали публичный тестовый Modbus TCP Slave, который хорошо подходит для задач обучения и тестирования:
- https://modbus.pult.online/
- По ссылке выше указан IP–адрес сервера, который поддерживает протокол Modbus TCP. Вы можете подключиться к нему и считывать/записывать данные.
- Кроме того, сервер позволяет считать текущее время – то есть приборы, которые имеют доступ в интернет и поддерживают работу в режиме Modbus TCP Master (*но не поддерживают другие средства синхронизации времени – например, протокол NTP*), могут синхронизировать с ним своё системное время.

Публичный тестовый сервер Modbus-TCP

Бесплатная сред

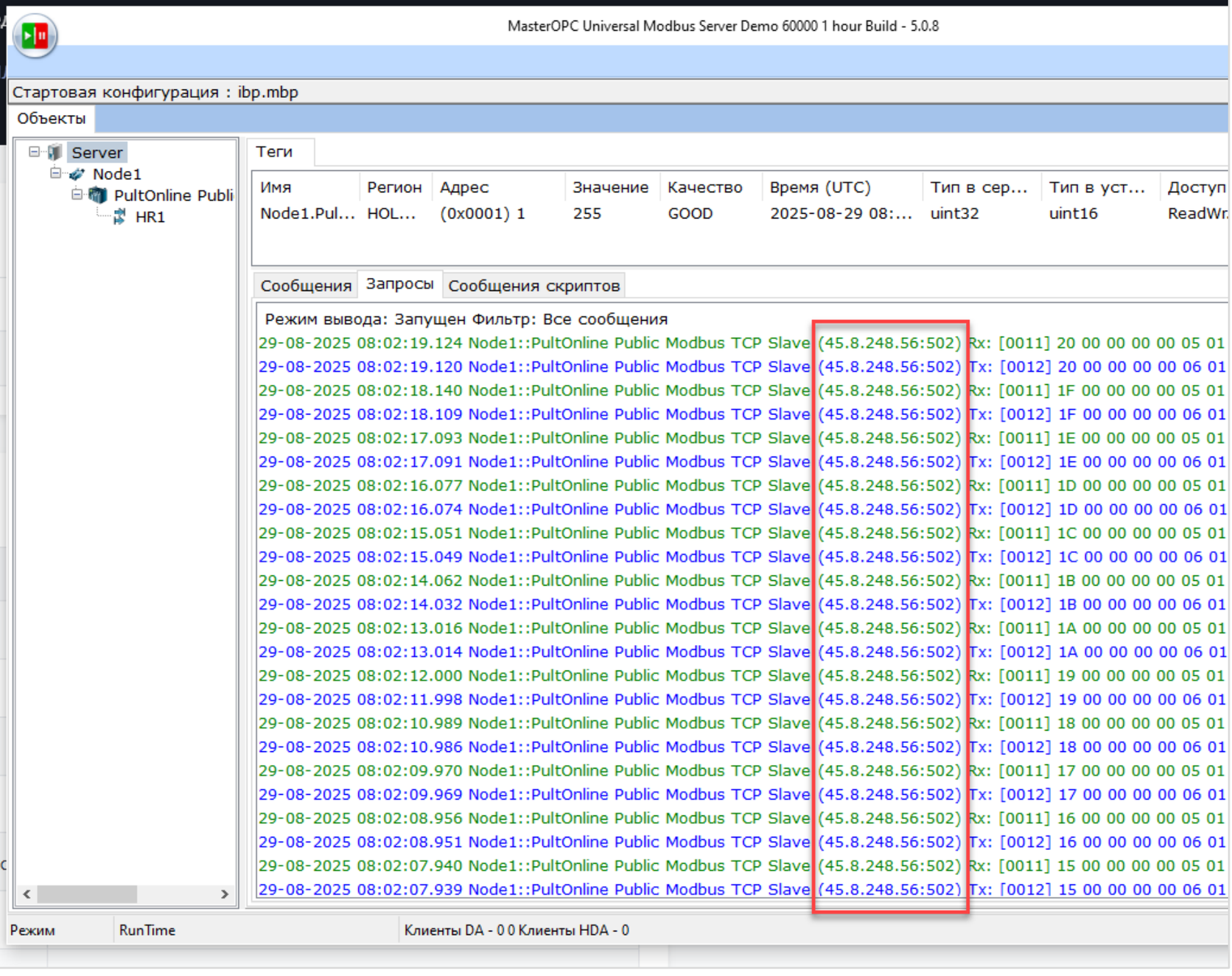
SCADA ПУЛЬТ.ОН

Данные для подключения

IP-адрес	45.8.248.56
Порт Modbus	502

Поддерживаемые команды

Код	Название
0x01	Чтение дискретных выходов
0x02	Чтение дискретных входов
0x03	Чтение регистров хранения
0x04	Чтение входных регистров
0x05	Запись одного дискретного выхо
0x06	Запись одного регистра



Подведение итогов

Мы рассмотрели основные аспекты протокола Modbus TCP.

- выяснили, что этот протокол основан на стеке TCP/IP и поэтому основан на механизме соединений;
- обсудили отличие структуры пакетов для Modbus TCP и Modbus RTU;
- рассмотрели назначение каждого поля заголовка пакета Modbus TCP (MBAP Header);
- немного поговорили про нюансы обработки разрыва соединений;
- коротко упомянули протоколы Modbus UDP и Modbus RTU over TCP.

Теперь перейдём к практике – настроим опрос по протоколу Modbus TCP с модулями Mx210 от компании ОВЕН.

2.21 Практика: опрос модулей Mx210

Mx210 – это линейка модулей ввода–вывода компании ОВЕН.

https://owen.ru/catalog/moduli_vvoda_vivoda/info/general_information_Mx210

Она является развитием рассмотренной нами в уроке 2.18 линейки Mx110.

Модули линейки Mx210 имеют интерфейс Ethernet и поддерживают протокол Modbus TCP; допускается до 4 одновременных соединений с master–устройствами.

> *Также Mx210 поддерживают и другие протоколы – MQTT (применяется в системах диспетчеризации и, например, домашней автоматизации) и SNMP (применяется в системах мониторинга сетевого оборудования – например, в центрах обработки данных).*

Модули Mx210 подключаются по топологии daisy–chain («цепочка») – каждый модуль имеет два разъема Ethernet, представляющих собой порты встроенного коммутатора. Таким образом, можно подключать модули друг к другу, повторяя топологию «шина», характерную для RS–485. Это позволяет обойтись без использования отдельных коммутаторов.

В случае отключения питания модуля – его порты физически замыкаются, образуя повторитель Ethernet; то есть отключение одного или нескольких модулей не приведёт к разрыву «цепочки».

***

В уроке по Mx110 мы настраивали опрос:

- модуля аналогового ввода МВ110–8А;
- модуля дискретного вывода МУ110–16Р.

В данном уроке рассмотрим опрос аналогичных модулей линейки Mx210:

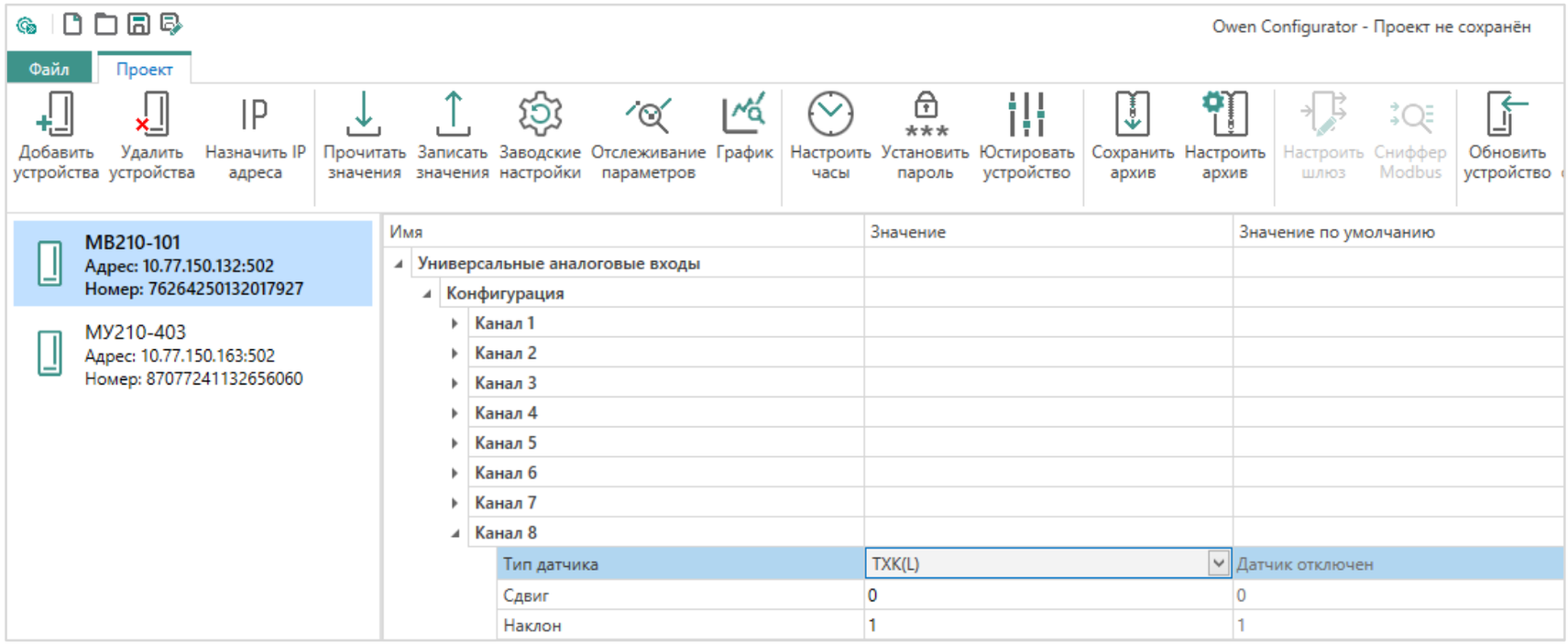
- модуля аналогового ввода МВ210–101;
- модуля дискретного вывода МУ210–403.

Настройка модулей

Как и модули Mx110 аппаратной ревизии H/W v.2.0 – модули Mx210 настраиваются через утилиту OWEN Configurator. Подключиться к модулю можно по Ethernet или (*что удобнее на этапе первоначальной настройки*) – через интерфейс MicroUSB (в этом случае на ПК будет создан виртуальный COM–порт).

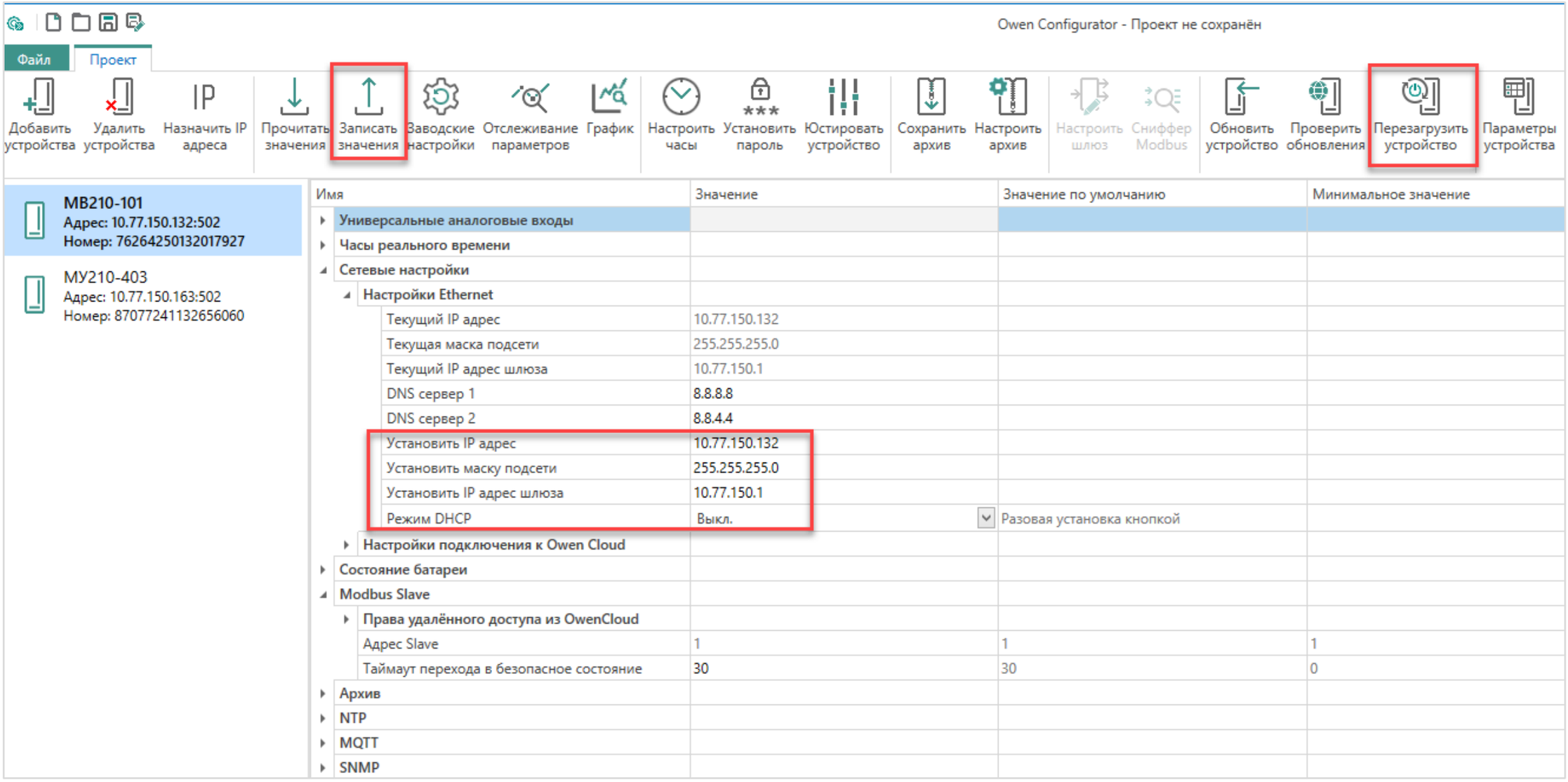
> *OWEN Configurator устанавливает соединение с модулем по протоколу Modbus TCP для отображения и записи его параметров.*

Ко входу 8 модуля MB210–101 подключена перемычка, а в настройках входа выбран тип датчика **ТХК (L)**; это уже известный нам способ имитации значения на аналоговом входе в случае отсутствия реального датчика.



Изменение сетевых настроек модуля происходит во вкладке **Сетевые настройки/Настройки Ethernet**.

После ввода новых значений настроек нужно нажать кнопку **Записать параметры**, а затем – кнопку **Перезагрузить устройство**, чтобы новые настройки вступили в силу. Мы задали обоим модулям различные IP–адреса из нашей локальной подсети; в эту же подсеть подключен ПК, с которого мы будем производить опрос модулей.



Модуль имеет всего два параметра, связанных с Modbus:

- **Адрес Slave** – этот параметр недоступен для редактирования и отображает его «стандартный» (с точки зрения модуля) адрес (Unit ID);
- **Таймаут перехода в безопасное состояние** – уже известный нам параметр, который позволяет переключить выходы модуля в безопасное для технологического процесса состояние в случае отсутствия запросов от master–устройства в течение заданного интервала времени.

Карта регистров

Из документации на модули можно узнать, что они поддерживают следующие функции Modbus:

- **0x03 (Read Holding Registers)** и **0x04 (Read Input Registers)**. В модулях используется общая модель памяти – то есть все параметры могут быть считаны как функцией **0x03**, так и функцией **0x04**;
- **0x06 (Write Single Register)** и **0x10 (Write Multiple Registers)** для записи holding–регистров.

Фрагмент из руководства на модуль:

6.5.1 Работа по протоколу Modbus TCP

Таблица 6.3 – Чтение и запись параметров по протоколу Modbus TCP

Операция	Функция
Чтение	3 (0x03) или 4 (0x04)
Запись	6 (0x06) или 16 (0x10)

Значение нужного нам 8 аналогового входа модуля МВ210–101 имеет тип **Float** и размещено в его регистрах **4021–4022**.

Запись регистров осуществляется командой **16 (0x10)**, чтение – командами **3 (0x03)** или **4 (0x04)**.

Таблица 6.3 – Регистры обмена по протоколу ModBus

Параметр	Значение (ед. изм.)	Адрес регистра		Тип доступа	Формат данных
		DEC	HEX		
Значение (float) на входе 1	—	4000	0xFA0	Только чтение	FLOAT 32
Циклическое время измерения входа 1	0...65535 (миллисекунд)	4002	0xFA2	Только чтение	UINT 16
Значение (float) на входе 2	—	4003	0xFA3	Только чтение	FLOAT 32
Циклическое время измерения входа 2	0...65535 (миллисекунд)	4005	0xFA5	Только чтение	UINT 16
Значение (float) на входе 3	—	4006	0xFA6	Только чтение	FLOAT 32
Циклическое время измерения входа 3	0...65535 (миллисекунд)	4008	0xFA8	Только чтение	UINT 16
Значение (float) на входе 4	—	4009	0xFA9	Только чтение	FLOAT 32
Циклическое время измерения входа 4	0...65535 (миллисекунд)	4011	0xFAB	Только чтение	UINT 16
Значение (float) на входе 5	—	4012	0xFAC	Только чтение	FLOAT 32
Циклическое время измерения входа 5	0...65535 (миллисекунд)	4014	0xFAE	Только чтение	UINT 16
Значение (float) на входе 6	—	4015	0xFAF	Только чтение	FLOAT 32
Циклическое время измерения входа 6	0...65535 (миллисекунд)	4017	0xFB1	Только чтение	UINT 16
Значение (float) на входе 7	—	4018	0xFB2	Только чтение	FLOAT 32
Циклическое время измерения входа 7	0...65535 (миллисекунд)	4020	0xFB4	Только чтение	UINT 16
Значение (float) на входе 8	—	4021	0xFB5	Только чтение	FLOAT 32
Циклическое время измерения входа 8	0...65535 (миллисекунд)	4023	0xFB7	Только чтение	UINT 16

Значение выходов модуля МУ210–403 представлено в виде битовой маски типа **Uint32** и размещено в регистрах **470–471**.

Выбор типа **Uint32** связан с тем, что модуль имеет 24 выхода, и для размещения их битовой маски требуется 2 регистра (в одном из которых 8 бит являются неиспользуемыми и всегда имеют значение **FALSE**).

Нужно отметить, что при чтении битовой маски следует считывать одним запросом оба её регистра; при попытке считать только один из них (т. е. «половинку» битовой маски) – модуль вернёт ошибку Modbus **0x02 ILLEGAL DATA ADDRESS.** Это касается и других параметров, которые занимают больше одного регистра.

DO1–DO24					
Битовая маска установки состояния выходов DO1–DO24	0…16777215	470	0x1D6	Чтение и запись	UINT 32
Безопасное состояние выхода DO1	0…1000 (0,10 %)	474	0x1DA	Чтение и запись	UINT 16
Безопасное состояние выхода DO2	0…1000 (0,10 %)	475	0x1DB	Чтение и запись	UINT 16

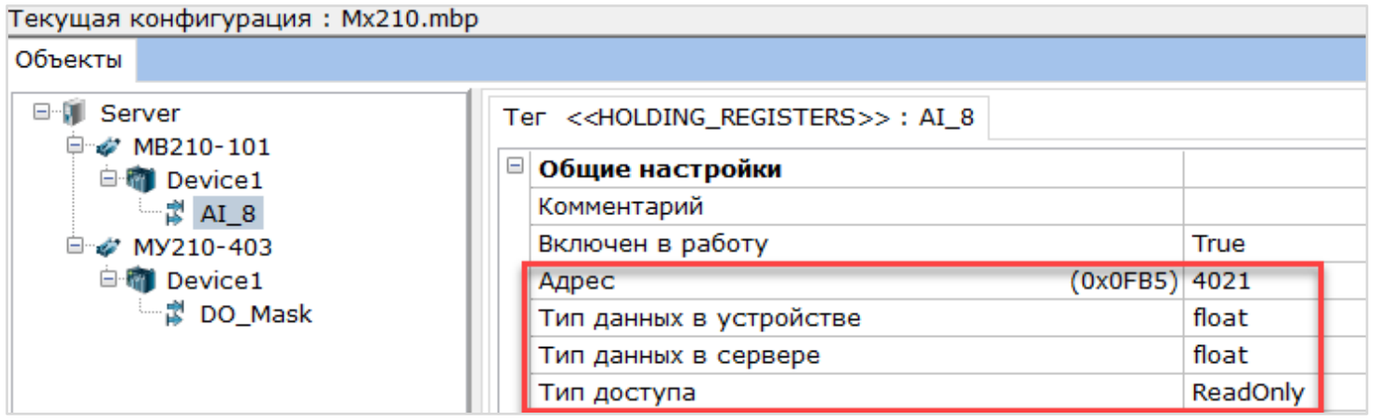
Настройка опроса в MasterOPC Universal Modbus Server

Мы уже не раз настраивали опрос в MasterOPC Universal Modbus Server, поэтому сейчас остановимся только на ключевых моментах:

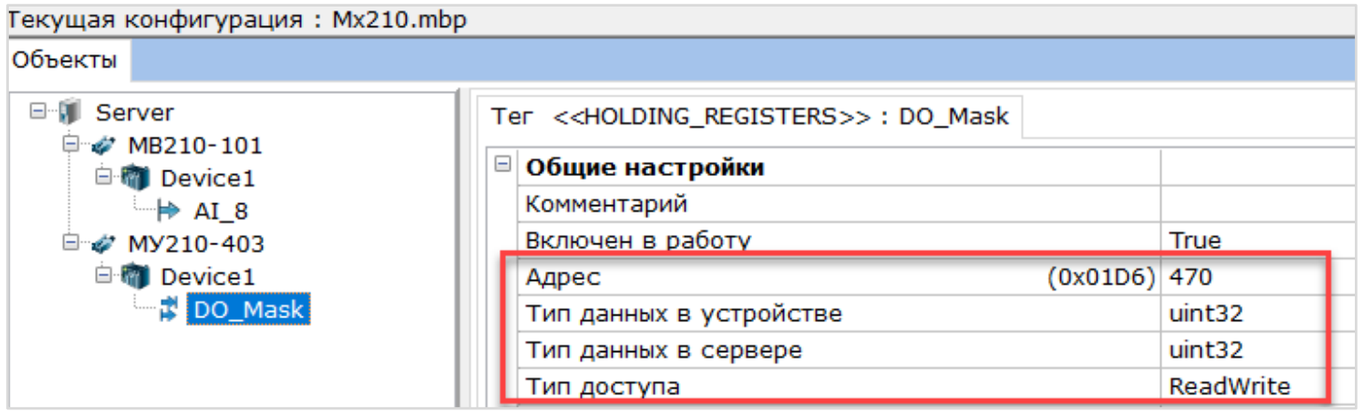
- создайте два коммуникационных узла с типом **TCP/IP**. В каждом укажите IP–адрес соответствующего модуля и номер порта – **502**;
- добавьте в каждый узел устройство. Адрес устройства (Unit ID) может быть задан любым числом из диапазона **1...255**; модуль не отвечает на запрос с **Unit ID** = **0**, что является отступлением от спецификации Modbus TCP [2, п. 4.4.1.2].

> *В старых версиях прошивок (вплоть до версии 1.0) модули отвечали на запрос **только** в том случае, если в запросе использовался Unit ID = 1. Это делало невозможным опрос модуля с помощью тех master–устройств, которые не позволяли пользователю настроить Unit ID в запросе и всегда устанавливали его в значение 0xFF или 0x00, как указано в спецификации Modbus TCP.*

В MB210–101 добавьте тег со следующими настройками:



В МУ210–403 добавьте тег со следующими настройками:



Ссылка на готовую конфигурацию:

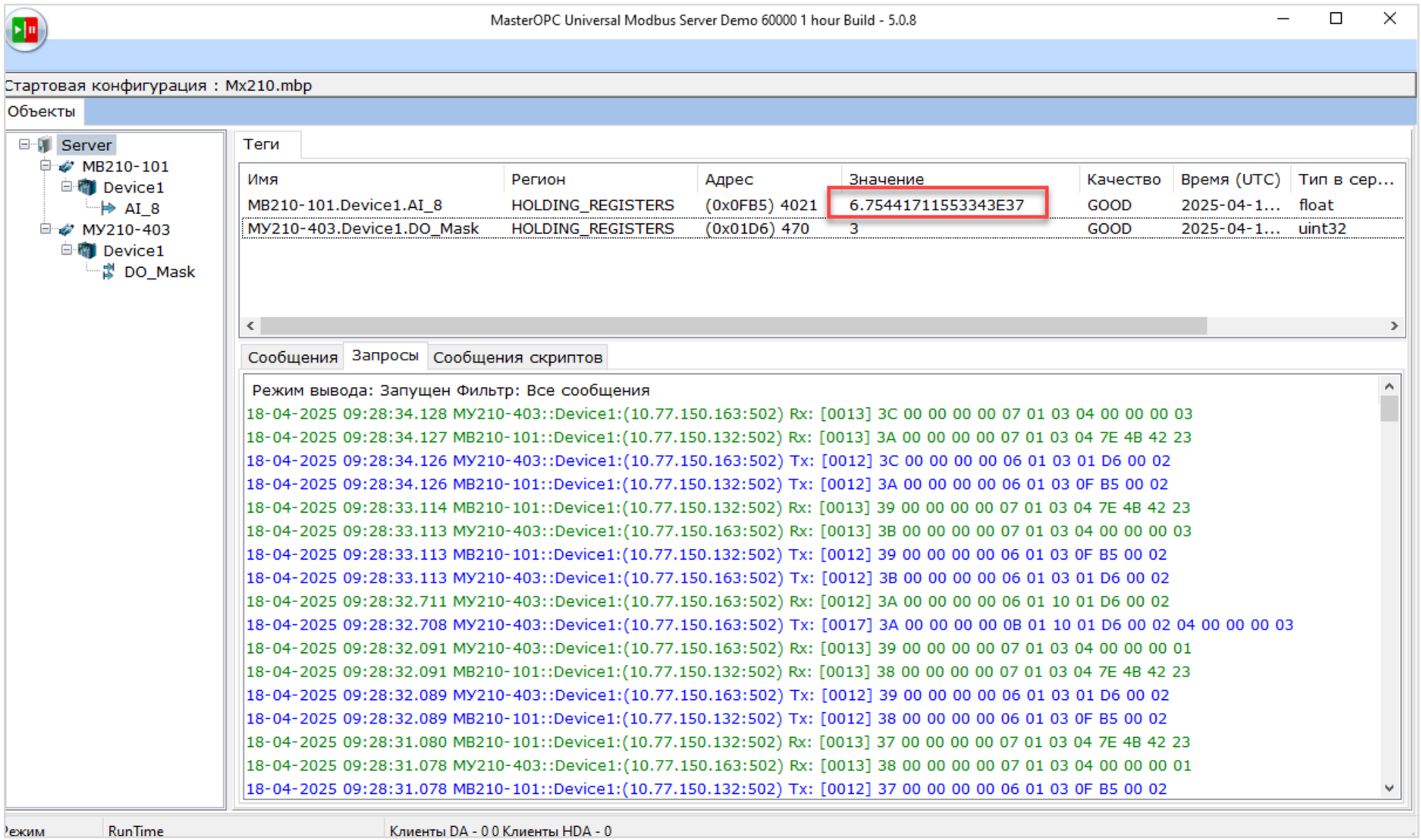
https://ftp.owen.ru/ModbusCourse/01_OpcTemplates/Mx210.mbp

Проверка обмена

Запустите OPC–сервер.

В теге **AI_8** отобразится текущее измеренное значение 8 аналогового входа модуля MB210–101.

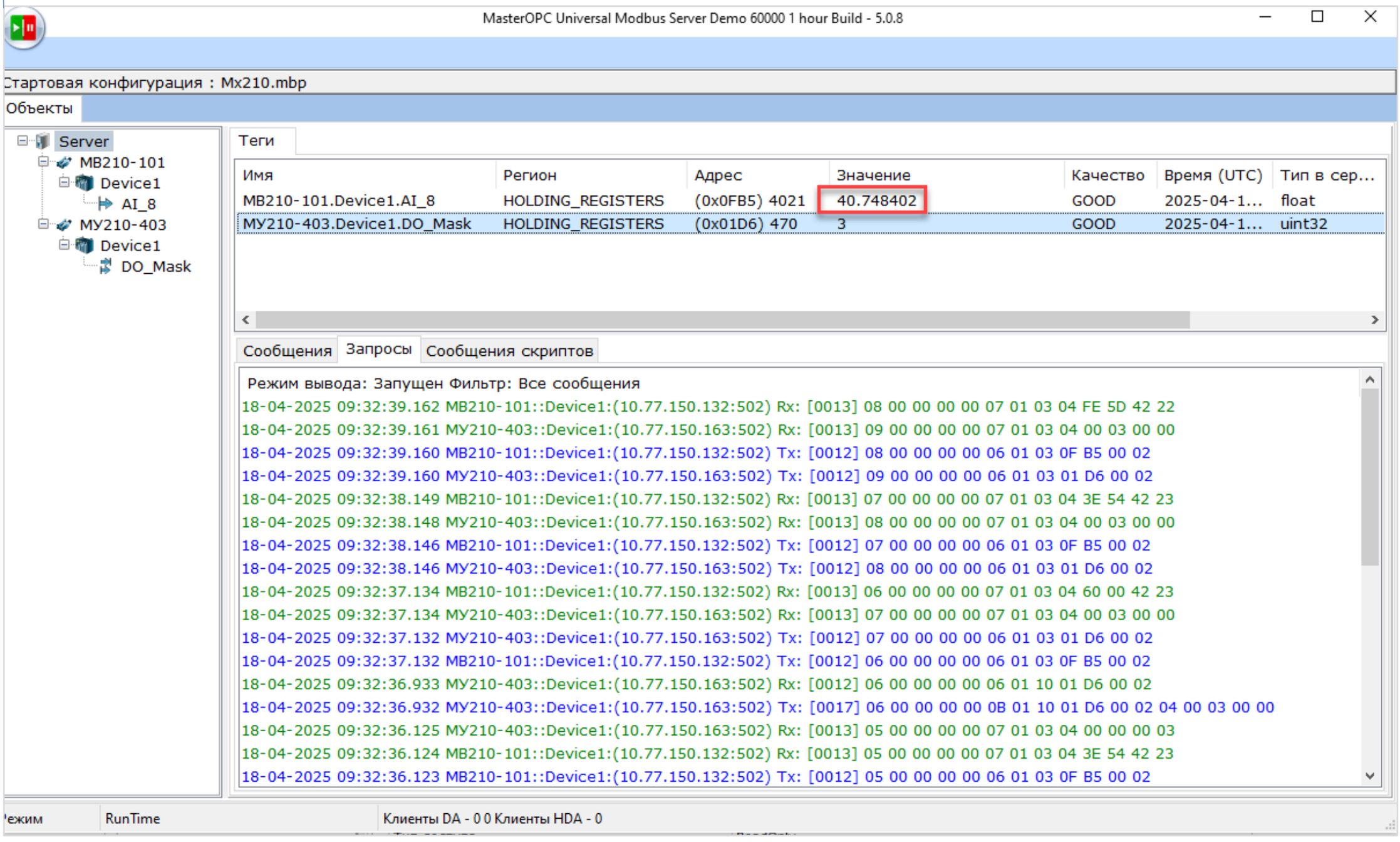
Но оно будет не таким, как мы ожидали:



Если мы запишем в тег **DO_Mask** значение 3, то включатся не 1 и 2 выход модуля, а 17 и 18.

Мы уже встречались с подобной ситуацией и знаем, что дело в порядке регистров. Для обоих тегов изменим значение настройки **Использовать перестановку байт устройства** на **False**. Это решит проблему.

Механизм кодирования ошибок аналоговых входов в модуле MB210–101 совпадает с тем, который используется в модуле MB110–8A (см. урок 2.18).



WireShark

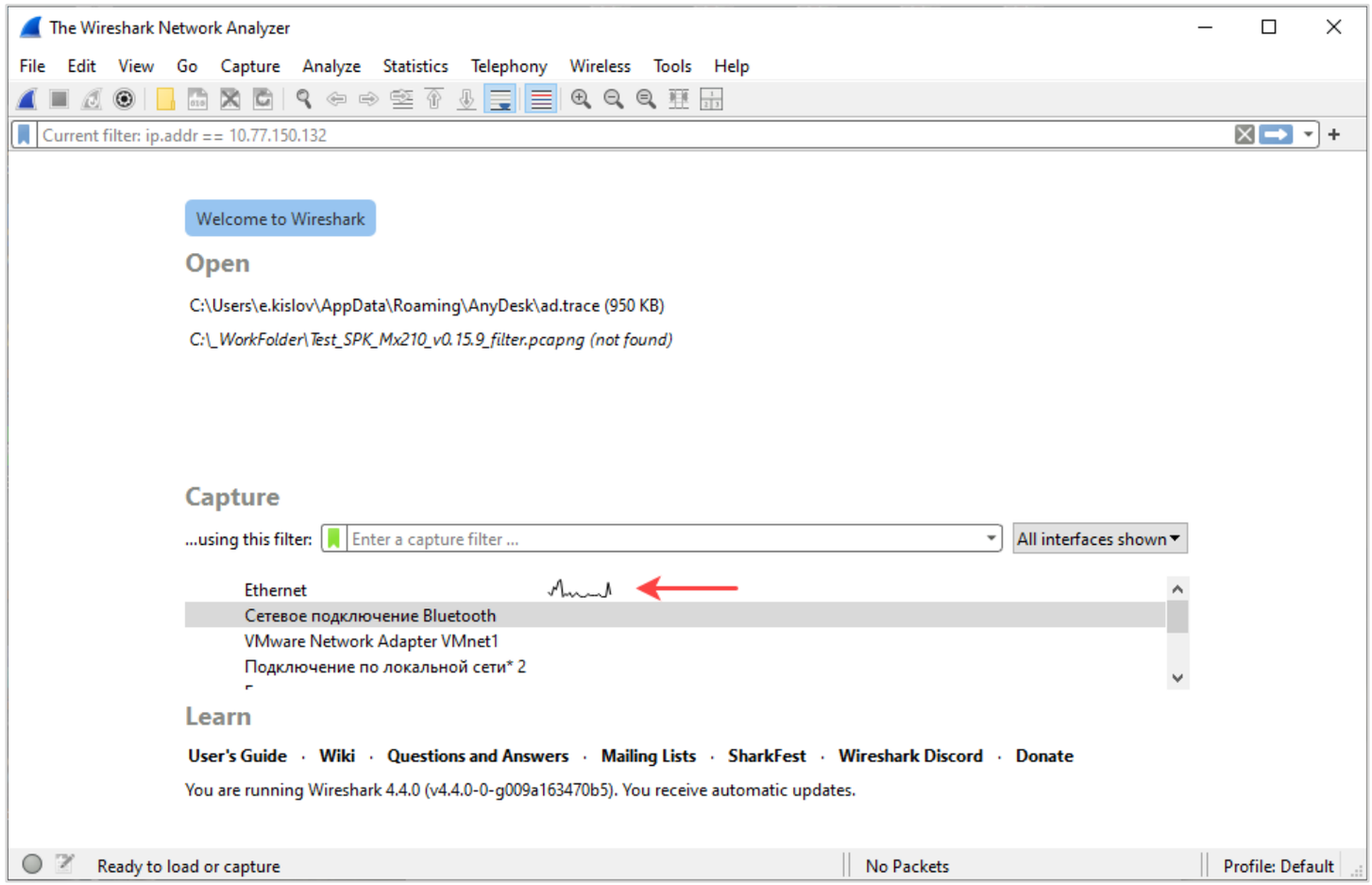
В уроке 2.12 мы рассмотрели утилиты–снифферы COM–порта, которые позволяют увидеть, какие пакеты передаются по последовательной линии связи.

Для интерфейса Ethernet подобные утилиты называются «анализаторами трафика». Самой известной из них является **WireShark**.

https://www.wireshark.org/

WireShark позволяет увидеть только те пакеты, которые поступили на сетевой интерфейс вашего компьютера. Это не создает проблем, когда одним из участников обмена является программа на ПК – например, MasterOPC Universal Modbus Server. Но если, предположим, в сети есть контроллер, который опрашивает модули ввода–вывода, и вы хотите увидеть передаваемые между ними пакеты – то вам придётся «завернуть» трафик этих устройств на сетевую карту своего ПК; для этого потребуется специализированный коммутатор.

После запуска WireShark нужно выбрать сетевой интерфейс ПК. Рядом с «активными» сетевыми интерфейсами, по которым в данный момент передаются пакеты, отображается пиктограмма в виде графика.



После выбора интерфейса вы увидите все пакеты, поступающие и отправляемые через данный интерфейс. Их может быть очень много – поэтому лучше воспользоваться строкой фильтра, расположенной в верхней части утилиты.

Давайте отобразим только пакеты, связанные с опросом модуля MB210–101, который в нашем случае имеет IP–адрес 10.77.150.132:

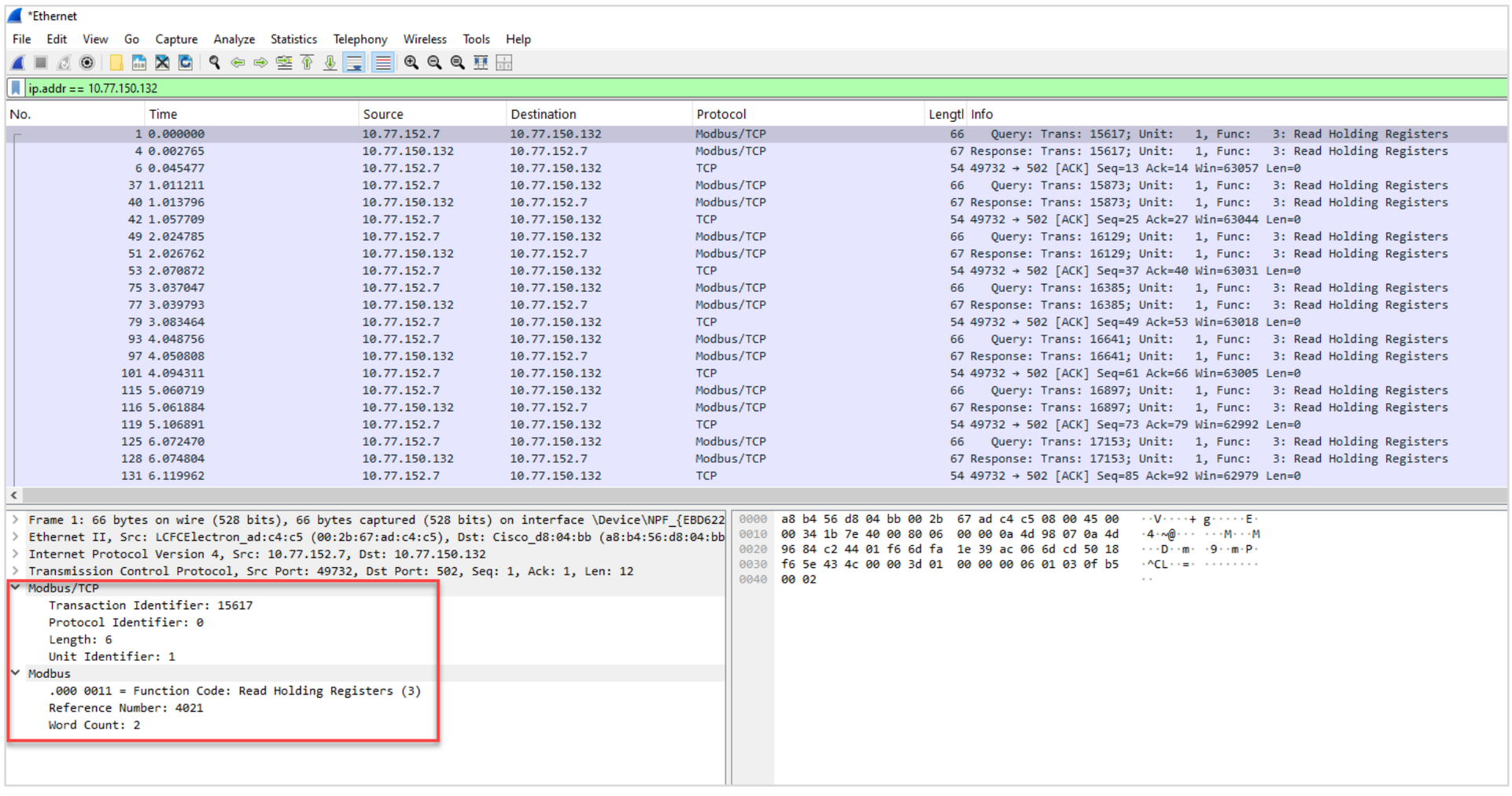
> *ip.addr == 10.77.150.132*

WireShark позволяет осуществить фильтрацию и по другим признакам – номерам портов, протоколам, размерам пакетов и т. д. Более подробная информация о настройке фильтров приведена в документации на утилиту:

https://wiki.wireshark.org/DisplayFilters

https://www.stationx.net/wireshark–cheat–sheet/

При выборе в окне лога отдельного пакета – отобразится его содержимое, включая значения полей протокола Modbus TCP:



Функции 0x14 (Read File Record) и 0x15 (Write File Record). Часть 1

Интересной особенностью модулей Mx210 является поддержка функций Modbus **0x14 (Read File Record)** и **0x15 (Write File Record)**.

Эти функции позволяют:

- считать дерево конфигурации устройства (*именно таким образом OWEN Configurator считывает из модуля список его параметров с их типами, диапазонами и т. д.*);
- считать архив модуля;
- выполнить какую–то специфическую операцию – например, программную перезагрузку модуля.

Все эти операции реализованы путём чтения и записи файлов.

«Файлы», описанные в спецификации Modbus, не имеют явного отношения к тем файлам, к которым мы привыкли при работе на ПК.

С точки зрения спецификации Modbus:

- файл – это объект, состоящий из записей;
- каждый файл может содержать до 10000 записей;
- неявно подразумевается, что запись – это регистр;
- каждый файл имеет номер в диапазоне 1...65535.

Регистры, из которых состоят эти файлы, обычно не могут быть считаны с помощью функций **0x03 (Read Holding Registers)** или **0x04 (Read Input Registers)** и записаны с помощью функций **0x06 (Write Single Register)** или **0x10 (Write Multiple Registers)**.

Часто в виде таких файлов представлены архивы теплосчетчиков и других измерительных приборов.

Например, функция **0x14 (Read File Record)** поддержана теплосчетчиками ВИС.Т (НПО Тепловизор), приборами компании Термотроник (ТВ–7, Питерфлоу) и другими подобными устройствами.

Структура запроса для функции **0x14** выглядит следующим образом:

- адрес slave–устройства (1 байт);
- код функции 0x14 (1 байт);
- число байт в запросе далее (1 байт);
- подзапрос 1: тип ссылки (1 байт). Согласно спецификации – данное поле всегда должно иметь значение 0x06;
- подзапрос 1: номер файла (2 байта). Согласно спецификации – файлы нумеруются с 1;
- подзапрос 1: адрес начальной считываемой записи (2 байта), нумерация с 0;
- подзапрос 1: количество считываемых записей (2 байта);
- <параметры следующих подзапросов>;
- контрольная сумма (2 байта).

Запрос с кодом функции **0x14** может включать в себя несколько подзапросов, но размер и запроса, и ожидаемого ответа не должны превышать 256 байт.

Пример запроса с кодом функции **0x14** из спецификации Modbus, в рамках которого выполняется два подзапроса к slave–устройству с адресом **16 (0x10)**:

- **первый подзапрос**: чтение из файла с номером **4** двух записей, начиная с записи с адресом **1**;
- второй подзапрос: чтение из файла с номером **3** двух записей, начиная с записи с адресом **9**.

> *10 14 0E **06 00 04 00 01 00 02** 06 00 03 00 09 00 02 A8 A8*

Пример ответа на этот запрос:

> *10 14 0C 05 06 **0D FE 00 20** 05 06 33 CD 00 40 A8 AD*

- 05h – это количество байт, считанных в рамках подзапроса, включая тип ссылки (06h), который тоже указывается в ответе;
- 0DFE0020h – значения двух регистров, считанных в рамках первого подзапроса;
- 33CD0040h – значения двух регистров, считанных в рамках второго подзапроса.

Функции 0x14 (Read File Record) и 0x15 (Write File Record). Часть 2

Структура запроса и ответа для функции **0x15 (Write File Record)** выглядит аналогичным образом – с той очевидной разницей, что записываемые данные входят в состав запроса и отсутствуют в ответе. Более подробную информацию об этой функции вы можете найти в [3, п. 6.15]

***

С функциями **0x14** и **0x15** связана ошибка протокола Modbus **0x08 (MEMORY PARITY ERROR)**. Она должна возвращаться slave–устройством в случае сбоя при обращении к памяти, в которой хранятся данные записей.

***

Если вы решите организовать чтение архивов с модулей Mx210, то помимо составления запросов и разбора ответов вам придётся решить ещё две задачи:

- расшифровку считанных данных (они будут зашифрованы с помощью алгоритма DES/CBC);
- их преобразование в соответствии со структурой архива, набором его переменных (который зависит от конкретного модуля и т. д.).

Вся необходимая для этого информация приведена в пункте «Запись архива», который присутствует в руководстве по эксплуатации на любой из модулей линейки Mx210. В пункте «Коды ошибок для протокола Modbus» приведён список ошибок, которые могут возникнуть при работе с архивом.

2.22 Тестовые задания

Ответы приведены в п. 9.

1. Какие протоколы определены в спецификации Modbus?
   - Modbus TCP
   - Modbus RTU over TCP
   - Modbus UDP
   - Modbus ASCII over TCP

2. Какой TCP–порт должен использоваться на стороне slave–устройства для подключения по протоколу Modbus TCP? (согласно спецификации)

3. Для чего в запросе Modbus TCP используется адрес slave–устройства? (Unit ID)
   - Для идентификации Modbus TCP slave-устройства, которому нужно доставить данный пакет
   - Для возможности опроса устройств, которые имеют одинаковый IP-адрес
   - Поле адреса не используется, зарезервировано под будущие расширения протокола
   - Для возможности пересылки запроса через шлюз Modbus TCP/Modbus RTU

4. Какие типы запросов не поддерживаются в протоколе Modbus TCP?
   - Запросы с битовыми функциями
   - Широковещательные
   - Групповые
   - Запросы с функциями записи

5. Какое поле пакета протокола Modbus RTU отсутствует в пакете Modbus TCP?
   - Код функции
   - Адрес slave-устройства
   - CRC (контрольная сумма)

2.23 Заключение

В данном модуле мы познакомились со спецификацией протокола Modbus и некоторыми аспектами интерфейса RS–485.

В рамках практических уроков мы настраивали обмен с различными устройствами; все эти устройства работали в режиме slave.

В следующем модуле мы рассмотрим, какие особенности могут возникнуть при настройке обмена устройств, которые поддерживают работу как в режиме slave, так и в режиме master. Сделаем это на примере программируемых реле и контроллеров компании ОВЕН.

3. Modbus в программируемых устройствах ОВЕН

3.1 Программируемые реле ПР (среда Owen Logic)

Программируемые реле ОВЕН – это линейка компактных свободно программируемых устройств, предназначенных для решения локальных и малых задач автоматизации в системах с количеством сигналов, не превышающих 150–200.

Информация об устройствах линейки доступна на официальном сайте ОВЕН:

https://owen.ru/catalog/programmiruemie_rele

Пользовательские алгоритмы создаются в среде Owen Logic. В ней же настраивается обмен по Modbus:

https://owen.ru/product/sreda_programmirovaniya_owen_logic

Поколения ПР

В настоящий момент можно выделить два поколения ПР:

Первое поколение, к которому относятся ПР200, ИПП120, ПР100 и ПР102.

Они имеют интерфейс **RS–485** и поддерживают работу по протоколам **Modbus RTU** и **Modbus ASCII** в режимах **Master** и **Slave**.

Второе поколение, к которому относятся ПР103, ПР205 и ПР225.

Они имеют интерфейсы **RS–485** и **Ethernet**, и поддерживают работу по протоколам **Modbus RTU**, **Modbus ASCII** и **Modbus TCP** в режимах **Master** и **Slave**.

2 поколение

ПР205

ПР103

ПР225

RS-485 | Modbus RTU/ASCII
режимы Master и Slave

Ethernet | Modbus TCP
режимы Master и Slave

> *Можно ещё упомянуть «нулевое» поколение, к которому относились приборы ПР110 и ПР114. Их продажи были прекращены в 2024 году. Эти ПР поддерживали только режим Modbus Slave (при использовании дополнительного коммуникационного модуля ПР–МИ485); реализация была близка к той, которая в дальнейшем использовалась в ПР первого поколения.*

Вебинар «Настройка обмена по Modbus для программируемых реле ПР в среде Owen Logic»

В рамках вебинара была рассмотрена настройка ПР как первого, так и второго поколения в режимах Master и Slave.

Запись вебинара доступна на:

- сайте компании ОВЕН: https://owen.ru/media/video/webinar_220252025
- YouTube: https://youtu.be/muYPoC2–DTw?si=5H2k–aS53UHRnCQV
- RuTube: https://rutube.ru/video/aaf6d752f3b2beaa66a1437ff93a9611/
- VK Видео: https://vk.com/video–68314714_456240297

В записи расставлены таймкоды.

Дополнительную информацию о реализации Modbus в программируемых реле ПР можно найти в руководствах по эксплуатации ПР (см. раздел **Документация** на странице соответствующего ПР), а также документации среды Owen Logic:

https://owen.ru/product/sreda_programmirovaniya_owen_logic

По этой же ссылке доступна памятка **Протокол Modbus в ОВЕН ПР**, в которой перечислены особенности и ограничения, связанные с реализацией Modbus в программируемых реле ОВЕН.

Ссылка на вебинар

https://vkvideo.ru/video–68314714_456240297

3.2 Программируемые контроллеры ПЛК1хх (среда CoDeSys V2.3)

ПЛК1хх – это линейка программируемых контроллеров, предназначенных для решения малых и средних задач автоматизации.

Информация об устройствах линейки доступна на официальном сайте ОВЕН:

https://owen.ru/catalog/programmiruemie_logicheskie_kontrolleri/info/general_information_100_150_154

https://owen.ru/catalog/programmiruemie_logicheskie_kontrolleri/info/general_information_110_160

В качестве среды разработки используется **CoDeSys V2.3**. В ней же настраивается обмен по Modbus:

https://owen.ru/product/codesys_v2

Поколения ПЛК1хх

История линейки ПЛК1хх началась в 2006 году, когда состоялся старт продаж моделей **первого поколения** – ПЛК100 и ПЛК150. Чуть позже к ним добавился ПЛК154. Выпуск контроллеров первого поколения был прекращен в 2024 году.

В 2009–2010 годах вышло **второе поколение** ПЛК1хх, включающее в себя модели ПЛК110 и ПЛК160.

Спустя несколько лет аппаратная платформа ПЛК110 была обновлена; новая модификация получила обозначение ПЛК110 [М01].

В 2016 году состоялся следующий этап обновления, результатом которого стала модификация ПЛК110 [М02]. В 2019 состоялся старт продаж ПЛК160 [М02]. Эти приборы относятся к **третьему поколению** ПЛК1хх. На данный момент (*лето 2025 года*) это последние представители линейки ПЛК1хх, выпуск которых ещё не прекращен.

***

Все перечисленные контроллеры имеют интерфейсы **RS–485**, **RS–232** и **Ethernet**, и поддерживают работу по протоколам **Modbus RTU**, **Modbus ASCII** и **Modbus TCP** в режимах **Master** и **Slave**. Реализация Modbus во всех поколениях ПЛК1хх приблизительно одинакова.

Отдельно стоит упомянуть линейку ПЛК63/73. Контроллеры этой линейки тоже программируются в среде CoDeSys V2.3, но реализация Modbus в них принципиально отличается от ПЛК1xx; этот вопрос не рассматривается в рамках данного модуля. Выпуск ПЛК63/73 прекращён в 2024 году.

Вебинар «Настройка обмена по Modbus для программируемых контроллеров ПЛК1xx в среде CoDeSys V2.3»

В рамках вебинара была рассмотрена настройка ПЛК110 [M02] в режимах Master и Slave.

Запись вебинара доступна на:

- сайте компании ОВЕН: https://owen.ru/media/video/webinar_280525
- YouTube: https://youtu.be/uJdXSd4d8Ug?si=FUGg1oOh2Wzx8R6W
- RuTube: https://rutube.ru/video/daf23649c208d0f908a32c8885a9595c/
- VK Видео: https://vkvideo.ru/video–68314714_456240306

В записи расставлены таймкоды.

Дополнительную информацию о реализации Modbus в ПЛК1xx [M02] можно найти в документе **Руководство пользователя ПЛК1xx [M02]**:

https://owen.ru/product/plk110_m02/documentation

Документация, библиотеки и примеры, связанные со средой CoDeSys V2.3, доступны по ссылке:

https://owen.ru/product/codesys_v2

Ссылка на вебинар

https://vkvideo.ru/video–68314714_456240306

3.3 Программируемые контроллеры ПЛК2xx и СПК (среда CODESYS V3.5)

Современное поколение контроллеров ОВЕН представлено двумя основными направлениями:

- моноблочными программируемыми логическими контроллерами ПЛК2xx;
- сенсорными панельными контроллерами СПК.

ПЛК2xx включает в себя две линейки:

- ПЛК210–1x и относящийся к ней ПЛК210–4G – современная линейка ПЛК на новой аппаратной платформе;
- ПЛК210–0x и ПЛК200 – стандартная и упрощенная линейки ПЛК на предыдущей аппаратной платформе.

Каждая линейка включает в себя ряд модификаций, отличающихся типом и количеством входов и выходов.

СПК тоже включает в себя две линейки:

- СПК210 – современная линейка СПК на новой аппаратной платформе;
- СПК1xx [M01] – линейка СПК на предыдущей аппаратной платформе.

В качестве среды разработки используется **CODESYS V3.5**. В ней же настраивается обмен по Modbus:

https://owen.ru/product/codesys_v3

> *Для ПЛК210 существуют модификации, которые поддерживают программирование в других средах разработки – например, линейка ПЛК210–PL программируется в среде Полигон.*

Вебинар «Настройка обмена по Modbus для программируемых контроллеров ПЛК2xx в среде CODESYS V3.5»

В рамках вебинара была рассмотрена настройка ПЛК210 в среде CODESYS V3.5 в режимах Master и Slave с помощью стандартных средств конфигурации, добавляемых в дерево проекта.

Запись вебинара доступна на:

- сайте компании ОВЕН: https://owen.ru/media/video/webinar_100625
- YouTube: https://youtu.be/ls8kTR7Z12o?si=2hRK2exH1CimGO61
- RuTube: https://rutube.ru/video/6b6961c2693a6268746d51d231118217/
- VK Видео: https://vkvideo.ru/video–68314714_456240317

В записи расставлены таймкоды.

Список материалов, упомянутых в ходе вебинара:

https://ftp.owen.ru/CoDeSys3/99_ForumFiles/LinksForWebinarModbusCodesys35_june2025.pdf

Дополнительную информацию о реализации Modbus в CODESYS V3.5 можно найти в документе **CODESYS V3.5. Протокол Modbus**:
https://owen.ru/product/codesys_v3/documentation

По этой же ссылке доступна памятка **Протокол Modbus в CODESYS V3.5**, в которой перечислены особенности и ограничения, связанные с реализацией Modbus в программируемых реле ОВЕН.

Документация, библиотеки и примеры, связанные со средой CODESYS V3.5, доступны по ссылке:

https://owen.ru/product/codesys_v3

Ссылка на вебинар:

https://vkvideo.ru/video–68314714_456240317

Вебинар «Настройка обмена по Modbus для ПЛК2xx в среде CODESYS V3.5 с помощью библиотеки OwenCommunication

В рамках вебинара была рассмотрена настройка ПЛК210 в среде CODESYS V3.5 в режимах Master и Slave с помощью функциональных блоков библиотеки OwenCommunication.

Запись вебинара доступна на:

- сайте компании ОВЕН: https://owen.ru/media/video/webinar_170625
- YouTube: https://youtu.be/GtmDufpQJd4?si=o8mOmlaZ3fP1O3rB
- RuTube: https://rutube.ru/video/11ed8e2631c0fd105127eca46178cc35/
- VK Видео: https://vkvideo.ru/video–68314714_456240320

В записи расставлены таймкоды.

Дополнительную информацию о реализации Modbus в CODESYS V3.5 можно найти в документе **CODESYS V3.5. Протокол Modbus**:

https://owen.ru/product/codesys_v3/documentation

Документация, библиотеки и примеры, связанные со средой CODESYS V3.5, доступны по ссылке:

https://owen.ru/product/codesys_v3

Ссылка на вебинар

https://vkvideo.ru/video–68314714_456240320

4. Modbus в панелях оператора

4.1 Основная информация

Панель оператора – это устройство, предназначенное для отображения информации о состоянии технологического оборудования и управления им.

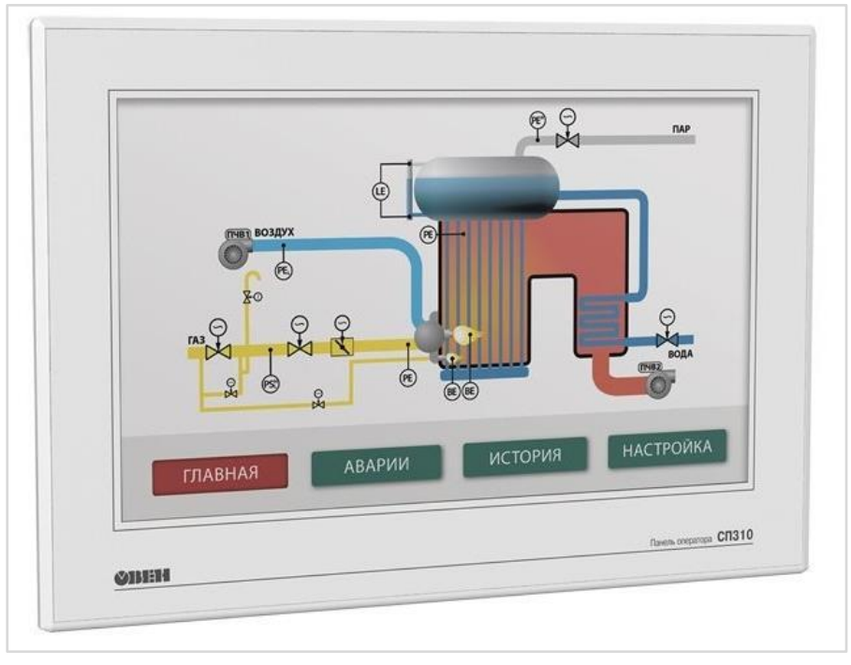
В большинстве случаев панель оператора подключается к ПЛК или другому программируемому устройству (например, ПР). ПЛК отвечает за выполнение программы управления технологическим процессом, а панель оператора – за предоставление обслуживающему персоналу информации о ходе этого процесса и возможности изменения его параметров (например, уставок регуляторов).

Значительно реже панель оператора используется как «центральное» устройство системы автоматизации, которое производит опрос модулей ввода–вывода и других устройств (регуляторов, измерительных приборов и т. д.) и иногда даже выполняет обработку полученных данных и формирование сигналов управления с помощью простых программ (которые обычно называются «макросами» или «скриптами»).

Связь между панелями оператора и другими устройствами осуществляется по одному из поддерживаемых ими протоколов обмена. Подавляющее большинство панелей оператора поддерживают протокол Modbus.

Мы рассмотрим использование этого протокола на примере линейки панелей СП3хх от компании ОВЕН.

https://owen.ru/product/sp3xx



Линейка панелей оператора СП3хх

Линейка панелей СП3хх состоит из трёх моделей, отличающихся диагональю экрана – СП307 (7"), СП310 (10.1") и СП315 (15.6").

СП307 и СП310 имеют две модификации – базовую (с постфиксом –Б) и расширенную (с постфиксом –Р).

СП315 выпускается только в расширенной модификации (СП315–Р).

Расширенная модификация отличается от базовой наличием интерфейсов Ethernet и USB A.

Создание проектов для панелей выполняется в среде **Конфигуратор СП300**.

ПО, документация и примеры доступны по ссылке:

https://owen.ru/product/sp3xx/software

Видеокурс по панелям СП3хх доступен по ссылке:

https://owen.ru/product/sp3xx/video

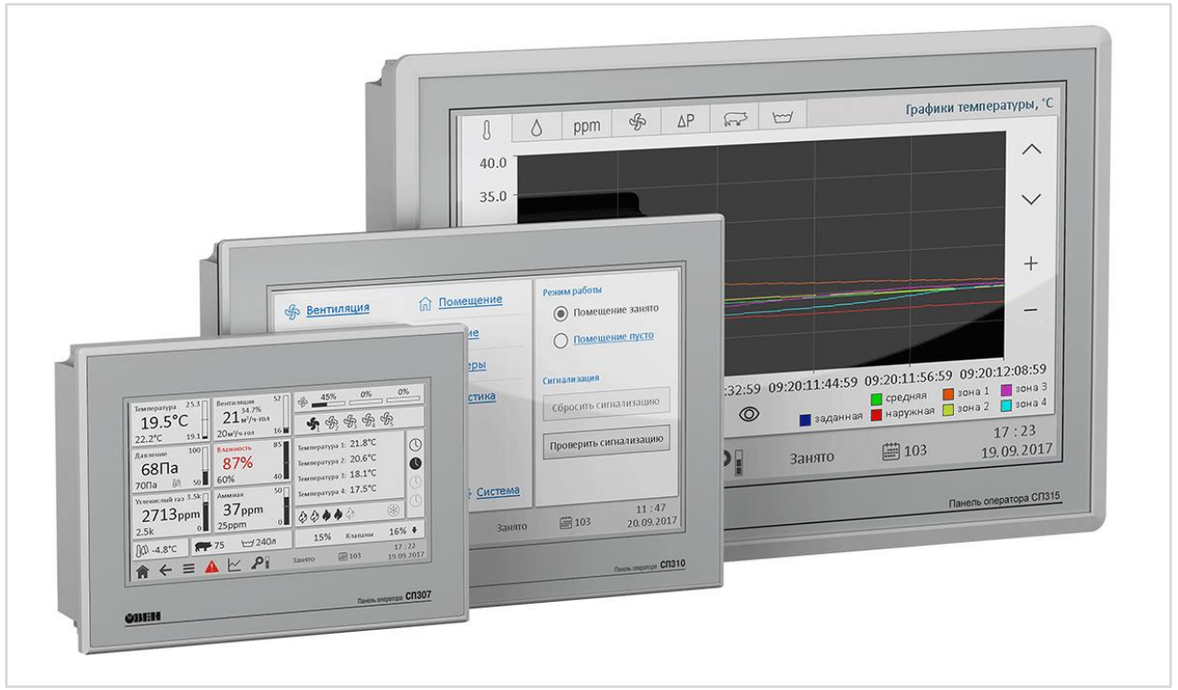
В следующих шагах будут рассматриваться только аспекты настройки обмена по протоколу Modbus. Подразумевается, что читатель ознакомился с документацией и видеоуроками по приведённым выше ссылкам и имеет базовое представление о создании проектов для панелей СП3хх.

***

Все панели линейки СП3хх поддерживают:

- работу по протоколу **Modbus RTU** в режимах **Master** и **Slave**;
- работу по протоколу **Modbus ASCII** в режимах **Master**.

Панели расширенных модификаций также поддерживают работу по протоколу **Modbus TCP** в режимах **Master** и **Slave**.


Графики температуры, °C
ppm
ΔP
40.0
35.0
09:20:11:44:59
09:20:11:56:59
09:20:12:08:59
средняя
наружная
зона 1
зона 2
зона 3
зона 4
заданная
Занято
103
17 : 23
19.09.2017
Панель оператора СП315
Вентиляция
Помещение
Режим работы
Помещение занято
Помещение пусто
Сигнализация
Сбросить сигнализацию
Проверить сигнализацию
Система
Занято
103
11 : 47
20.09.2017
Панель оператора СП310
Температура
19.5°C
22.2°C
19.1
Вентиляция
21 м³/ч пол
20 м³/ч пол
Давление
68Па
70Па
50
Влажность
87%
60%
Углекислый газ
2713ppm
2.5k
Аммиак
37ppm
25ppm
Температура 1: 21.8°C
Температура 2: 20.6°C
Температура 3: 18.1°C
Температура 4: 17.5°C
45%
0%
0%
15%
Клапаны
16%
-4.8°C
75
240л
Занято
103
17 : 22
19.09.2017
Панель оператора СП307
OBEH

Настройки интерфейса RS–485/RS–232. Часть 1

Панель имеет два последовательных порта – **PLC–порт** и **Download–порт**.

Оба порта являются **независимыми** и могут использоваться одновременно; Download–порт помимо обмена с другими устройствами может использоваться как резервный способ загрузки проекта в панель (основной способ – через кабель USB).

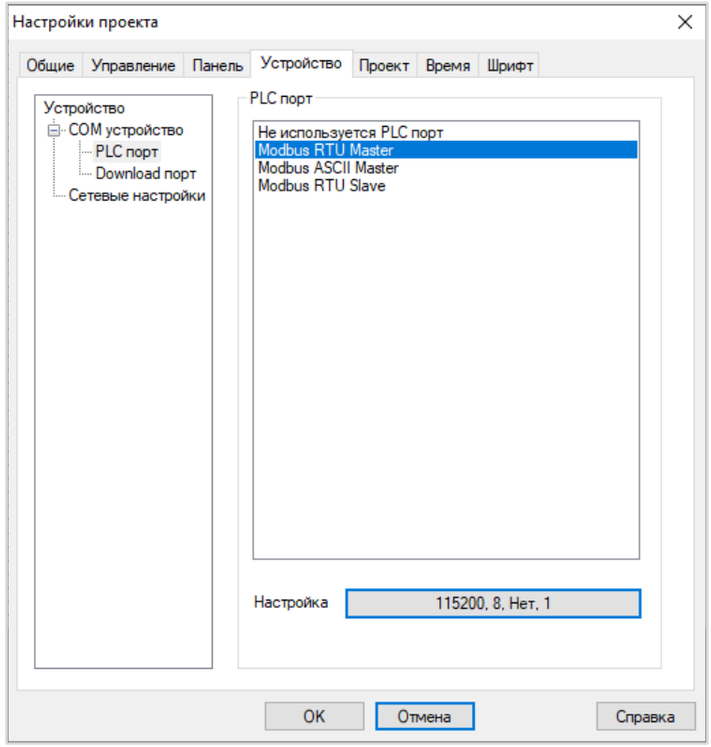
На каждый порт выведено два интерфейса – **RS–485** и **RS–232.**

Эти интерфейсы являются **созависимыми** – в режиме **Modbus Master** настроенные в конфигураторе для конкретного порта панели запросы одновременно отправляются по обоим его интерфейсам. Поэтому в большинстве случаев в рамках каждого порта используется только один из интерфейсов.

Настройки интерфейсов производятся в меню **Файл – Настройки проекта – Устройство**.

Для каждого из интерфейсов выбирается один из режимов работы:

- не используется;
- Modbus RTU Master;
- Modbus ASCII Master;
- Modbus RTU Slave.



Настройки интерфейса RS–485/RS–232. Часть 2

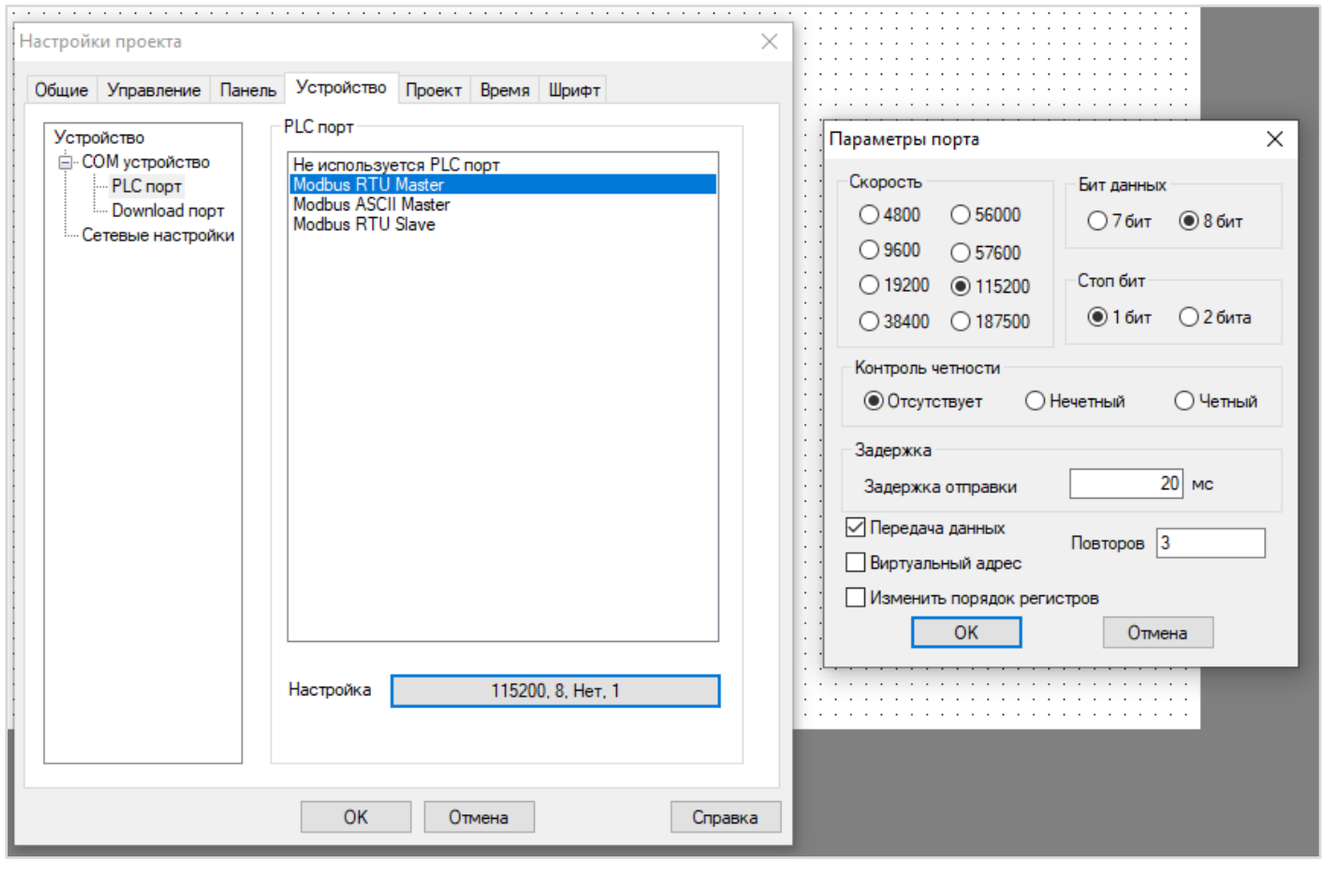
Нажатие на кнопку **Настройка**, расположенную в нижней части вкладки **Устройство**, открывает окно настроек интерфейса.

В этом окне производится настройка параметров выбранного интерфейса:

- Скорость обмена, количество бит данных, режим контроля чётности и количество стоп–бит;
- **Задержка отправки (мс)** – в режиме **Master** этот параметр определяет задержку между получением ответа от slave–устройства и отправкой следующего запроса (turnaround delay), а в режиме **Slave** – задержку между получением запроса и отправкой ответа (response delay). Мы обсуждали их в уроке 2.12;
- **Количество повторов** – количество повторных попыток отправки запроса в том случае, если на него не был получен ответ от slave–устройства;

> *Настройка таймаута ожидания ответа в этом меню отсутствует, но его можно настроить через системные регистры; мы обсудим это в уроке 4.5.*

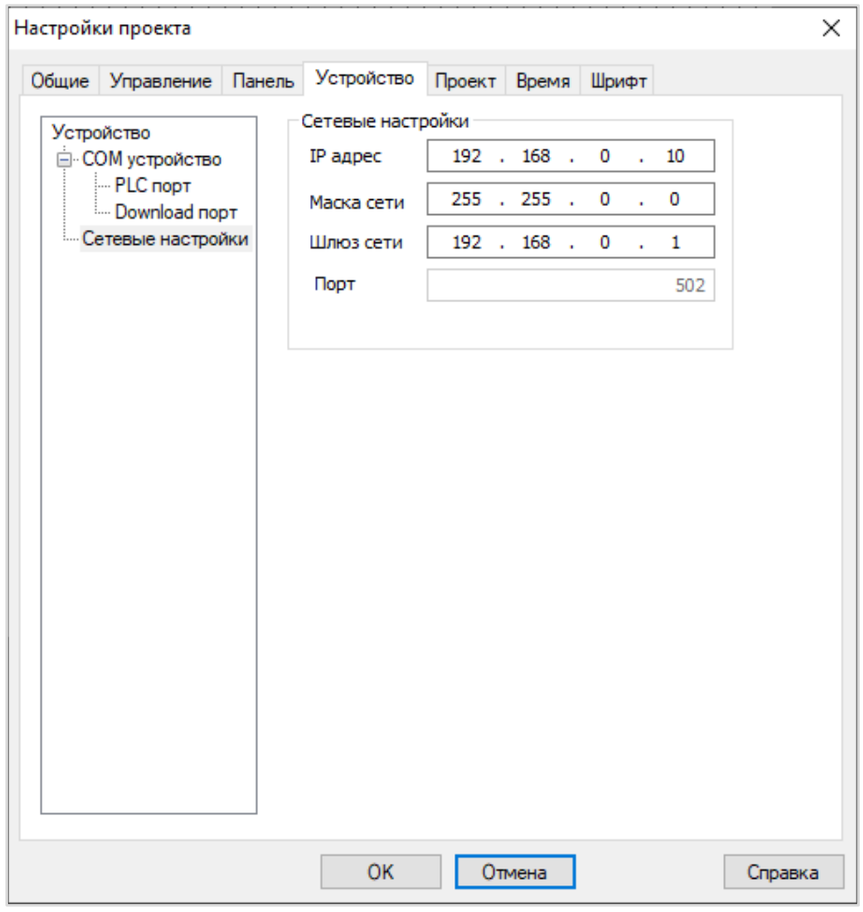
- **Виртуальный адрес** – специфическая настройка, не используемая в современных версиях ПО панели;
- галочка **Изменить порядок регистров** позволяет изменить порядок регистров при считывании и записи параметров, которые занимают больше одного регистра (например, типа Float, DWORD и т. д.);



Настройки интерфейса Ethernet

> ***Примечание**: интерфейс Ethernet присутствует только у расширенных модификаций панелей (СП307–Р, СП310–Р, СП315–Р).*

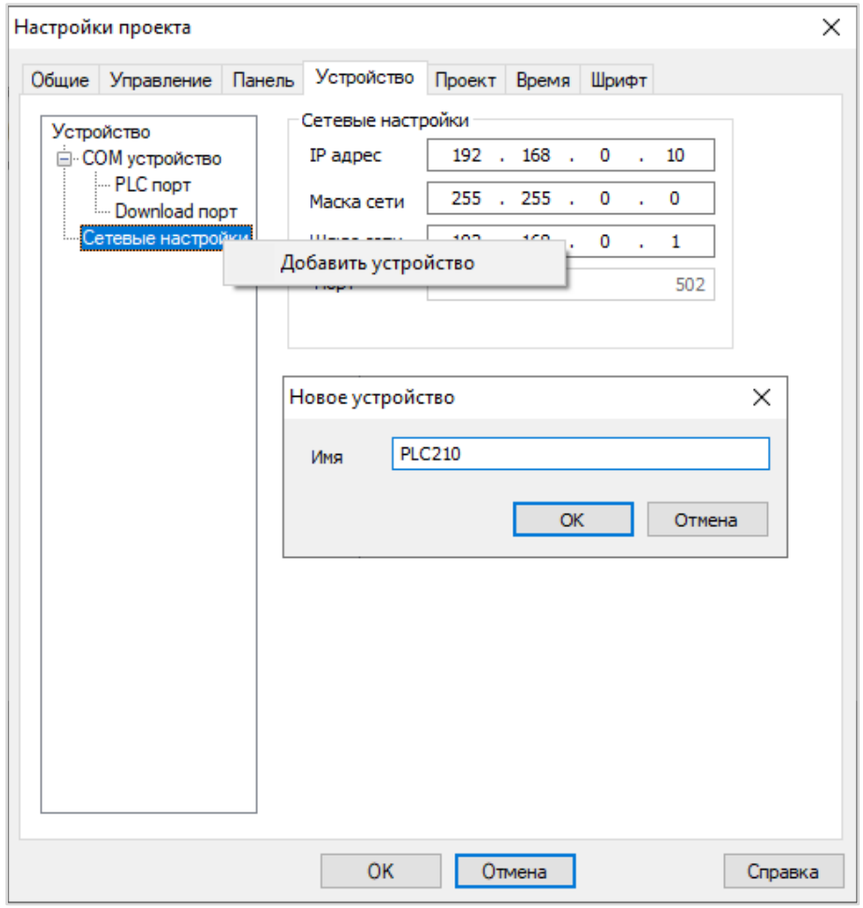
Сетевые настройки панели (IP–адрес, маски подсети и шлюз) задаются в меню **Файл – Настройки проекта – Устройство – Сетевые настройки**.



Режим **Modbus TCP Slave** активен всегда; поддерживается до 10 одновременных клиентских подключений.

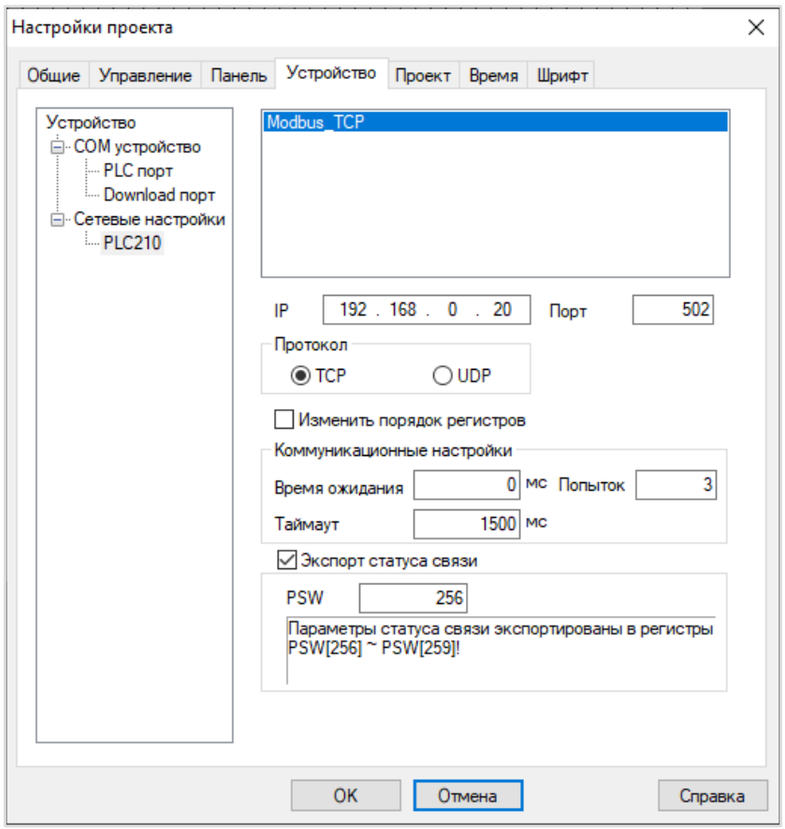
Для настройки панели в режиме **Master** нужно нажать правой кнопкой мыши на узел **Сетевые настройки** и выбрать команду **Добавить устройство**. Поддерживается до **8** slave–устройств. Если добавить больше устройств – то после загрузки проекта на экране панели будет отображена ошибка с кодом **32**.

> ***Примечание**: системные биты диагностики обмена доступны только для первых **6** добавленных slave–устройств.*



Для добавленного slave–устройства нужно задать настройки опроса:

- IP–адрес и порт slave–устройства;
- **Тип протокола** – Modbus TCP или Modbus UDP;
- галочка **Изменить порядок регистров** позволяет изменить порядок регистров при считывании и записи параметров, которые занимают больше одного регистра (например, типа Float, DWORD и т. д.);
- **Время ожидания –** время, выделяемое на установку TCP–соединения со slave–устройством;
- **Таймаут** – время ожидания ответа на отправленный Modbus–запрос;
- **Количество попыток** – количество повторных попыток отправки запроса в случае, если на него не был получен ответ от slave–устройства;
- галочка **Экспорт статуса связи** позволяет указать адрес первого из четырёх PSW–регистров панели, в которые будет записываться диагностическая информация об обмене с данным slave–устройством. Мы обсудим их в уроке 4.5.



> ***Примечание**: если предполагается, что в процессе работы системы автоматизации функционирование канала связи может быть нестабильным (например, кабель Ethernet будут отключать и подключать), то рекомендуется установить следующие значения настроек:*
>
> - *время ожидания = 10;*
> - *количество попыток = 10;*
> - *таймаут = 15000.*
>
> *Это обеспечит возобновление обмена после восстановления работоспособности канала связи.*

Варианты опроса в режиме Modbus Master

Если требуется, чтобы панель работала в режиме **Modbus Master**, то существует три варианта настройки опроса:

- опрос через элементы – этот способ прост в настройке, но наименее эффективен с точки зрения производительности;
- опрос через функциональную область – настройка чуть сложнее, а эффективность несколько выше;
- опрос через макросы – наиболее сложный в настройке и наиболее эффективный способ.

В уроке 4.2 мы рассмотрим настройку опроса через элементы и функциональную область, а в уроке 4.3 – настройку опроса через макросы. В уроке 4.4 рассмотрим настройку панели в режиме Modbus Slave.

4.2 Настройка опроса в режиме Modbus Master

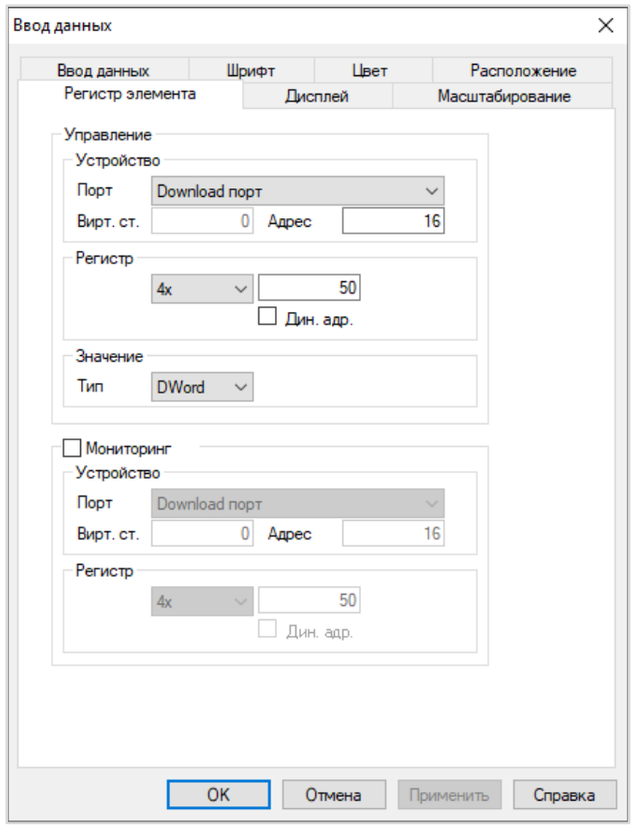
Перед началом настройки опроса требуется настроить используемый интерфейс панели. Как это сделать – показано в предыдущем уроке.

Рассмотрим настройку опроса на примере использования элемента **Цифровой ввод**.

Этот элемент позволяет отобразить числовое значение и изменить его. Предположим, что это значение с плавающей точкой (типа **Float**), которое хранится в **holding**–регистрах **50–51** slave–устройства с адресом **16**, которое подключено к **Download–порту** панели.

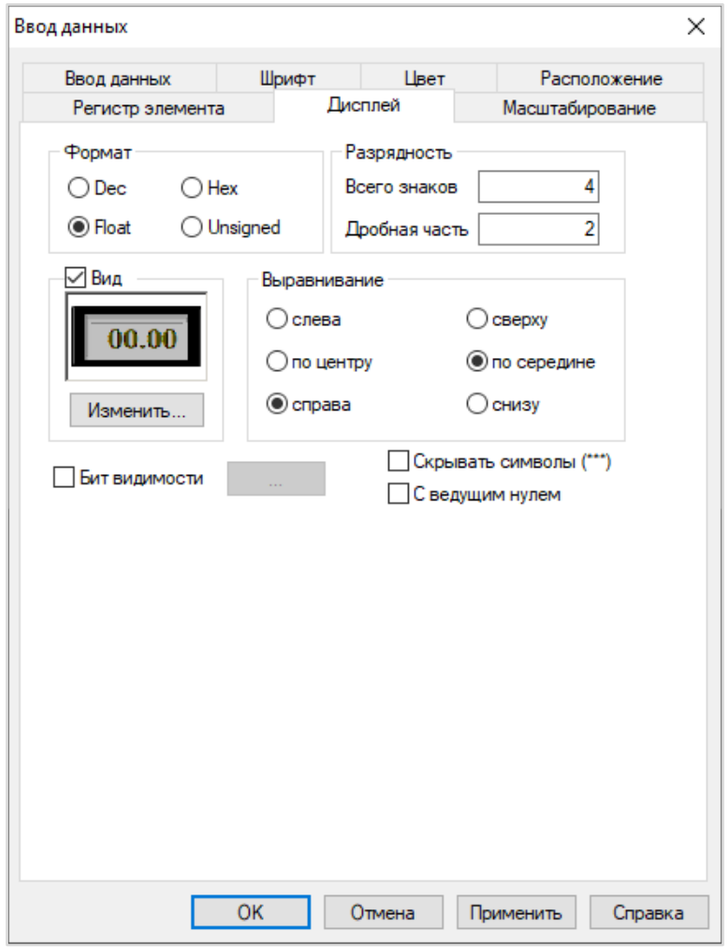
Добавим на экран элемент **Цифровой ввод**, двойным кликом на нём откроем окно настроек и на вкладке **Регистр элемента** зададим следующие параметры:

- **Порт** – в этом выпадающем списке нужно выбрать последовательный порт, к которому подключено slave–устройство (для протокола **Modbus RTU** и **Modbus ASCII**), или конкретное slave–устройство (для протокола **Modbus TCP**);
- **Адрес** – адрес опрашиваемого slave–устройства;
- **Регистр** – в этом выпадающем списке нужно выбрать идентификатор области памяти slave–устройства, к которой будет производиться доступ (**важно запомнить** – это не код функции Modbus, которая будет использована в запросе). Мы обсудим эти идентификаторы в следующем шаге;
- **Начальный адрес регистра** параметра, опрос которого будет производить элемент. Адресация регистров ведётся с **0**;
- **Тип** – количество регистров в запросе (Word – 1, Dword – 2).



На вкладке **Дисплей** следует указать тип параметра и, в случае необходимости, другие настройки отображения. Наиболее часто используемыми из них является **Всего знаков** (общее количество цифр, которое будет отображать элемент) и **Дробная часть** (количество знаков, которые будут отображаться после десятичной точки).

> *Сочетания настроек **Тип/Формат** соответствуют следующим привычным для программистов типам данных:*
>
> - *Word/Dec – Int16;*
> - *Word/Unsigned – Uint16;*
> - *Word/Hex – Uint16 (с отображением в шестнадцатеричной системе счисления);*
> - *Dword/Dec – Int32;*
> - *Dword/Unsigned – Uint32;*
> - *Dword/Hex – Uint32 (с отображением в шестнадцатеричной системе счисления);*
> - *Word/Float – неподдерживаемое сочетание;*
> - *Dword/Float – Float.*



Идентификаторы области памяти Modbus

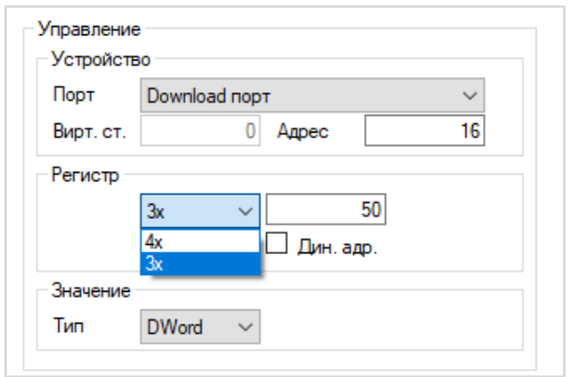
Во многих панелях оператора нет возможности в явном виде указать функцию Modbus, которая должна использоваться для опроса параметра (как, например, это можно сделать в ПЛК, ПР и других программируемых устройствах). Вместо этого указывается идентификатор области памяти, на основании которого автоматически происходит выбор функции. Давайте вспомним таблицу из урока 2.5:

ID	Обозначения	Применяемые функции	Описание
0x	Coils Дискретные выходы Ячейки, катушки, обмотки	0x01, 0x05, 0x0F	Биты, доступные для чтения и записи
1x	Discrete Inputs (DI) Дискретные входы	0x02	Биты, доступные только для чтения
3x	Input Registers (IR) Input-регистры Регистры ввода Аналоговые входы	0x04	Регистры, доступные только для чтения
4x	Holding Registers (HR) Holding-регистры Регистры хранения Аналоговые выходы	0x03, 0x06, 0x10, 0x16, 0x17	Регистры, доступные для чтения и записи

У нашего цифрового ввода доступно два варианта:

- **3x** – input–регистры. Для чтения используется функция **0x04 (Read Input Registers)**, запись не поддерживается – потому что такие регистры доступны только для чтения;
- **4x** – holding–регистры. Для чтения используется функция **0x03 (Read Holding Registers)**, для записи – функция **0x06 (Write Single Register)** для типа **Word** и **0x10 (Write Multiple Registers)** – для типа **Dword**.

> *Некоторые элементы (например, **Цифровой дисплей**) не поддерживают возможность ввода данных и, соответственно, используют только функцию чтения.*



> ***Примечание**: некоторые slave–устройства не поддерживают функцию записи **0x06**. В этом случае в панели можно активировать использование функции **0x10** для записи значений типа **Word**. Для этого нужно установить бит соответствующего системного регистра:*
>
> - *Download–порт: PFW26.2*
> - *PLC–порт: PFW36.2.*

Принцип опроса через элементы

Когда оператор с помощью элемента **Цифровой ввод** устанавливает новое значение – то оно **однократно** записывается в соответствующий параметр slave–устройства.

Запрос на чтение значения параметра отправляется циклически. У пользователя нет возможности повлиять на его периодичность; это зависит от «нагруженности» проекта, загруженного в панель.

Проект панели может содержать множество экранов, но Modbus–запросы формируют только те элементы, которые размещены на открытом в данный момент экране панели.

Для оценки (в рамках проекта с одним настроенным элементом, который формирует один запрос чтения):

- последовательный порт, скорость обмена = 115200, задержка отправки = 0: период опроса составляет 50 мс;
- Ethernet: период опроса составляет 15 мс.

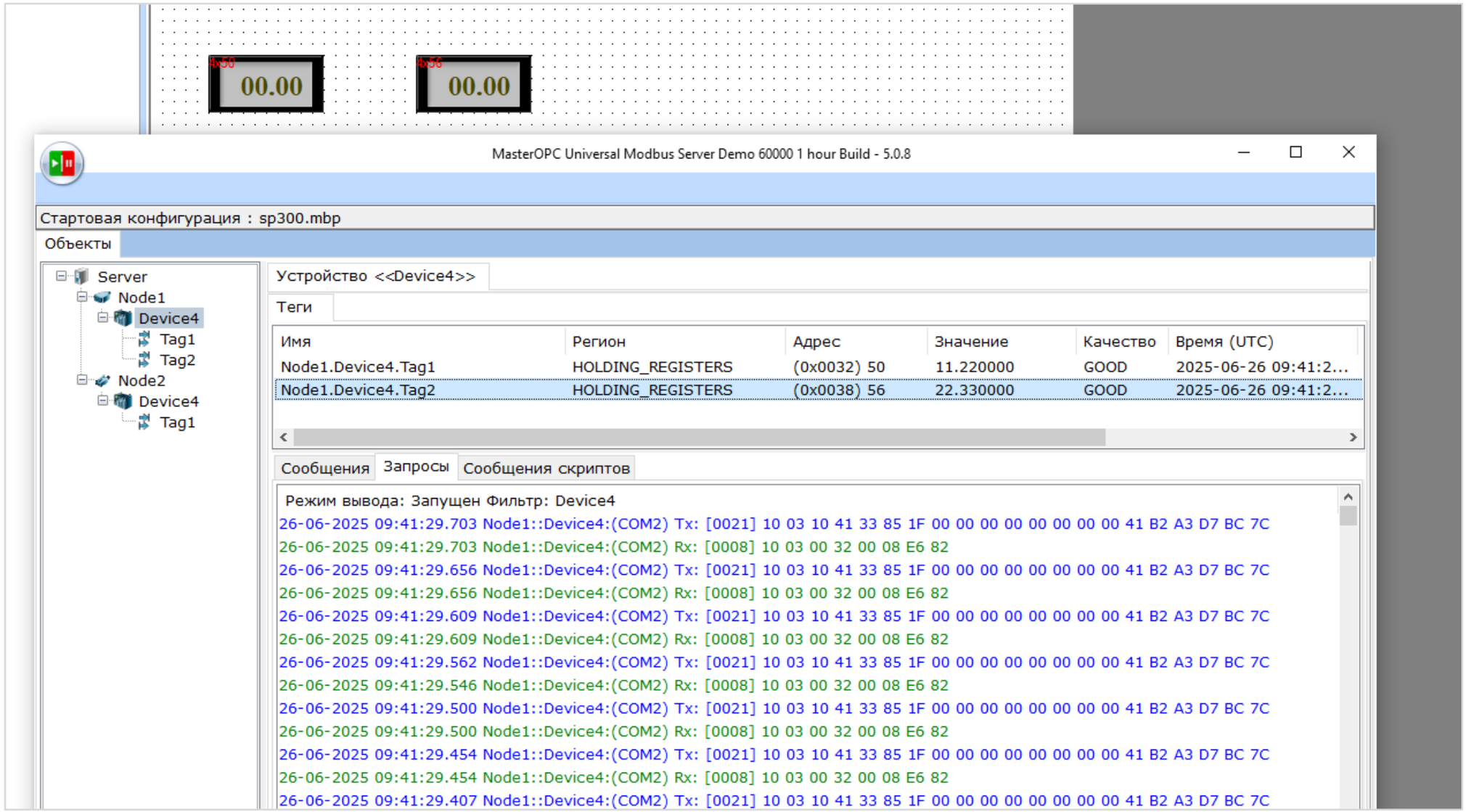
Групповой запрос

Панель автоматически формирует групповые запросы к slave–устройствам, если разность между адресами опрашиваемых регистров не превышает 6. Иными словами, регистры 50 и 56 будут считаны одним групповым запросом, а 50 и 57 – двумя одиночными.

Наглядная демонстрация:

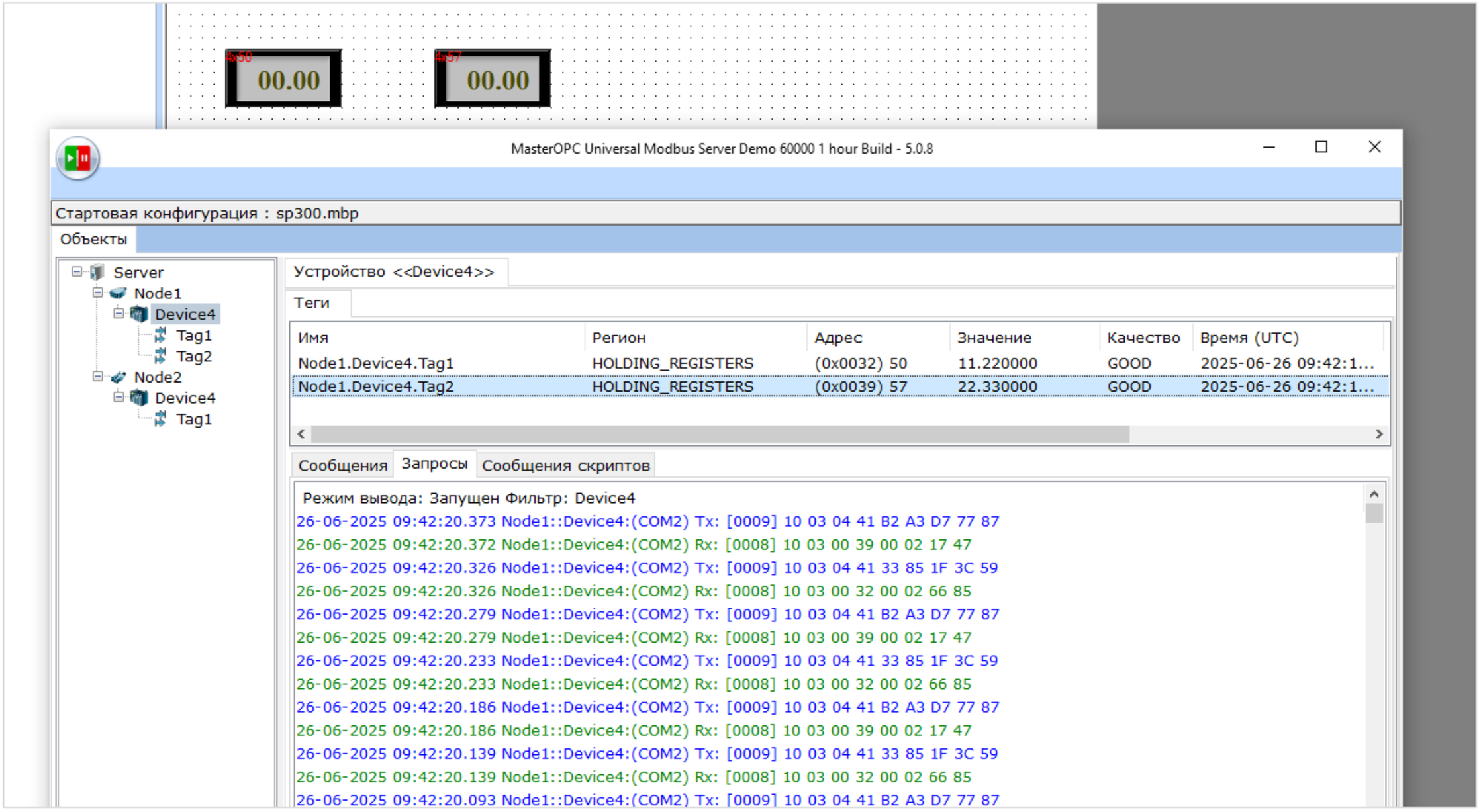
Добавим на экран ещё один **Цифровой ввод** с адресом регистра **4x56 (Dword/Float**).

Лог обмена со slave–устройством будет содержать только один групповой запрос:



Заменим адрес регистра на **4x57** (**Dword/Float)**.

Теперь мы увидим в логе два отдельных запроса.

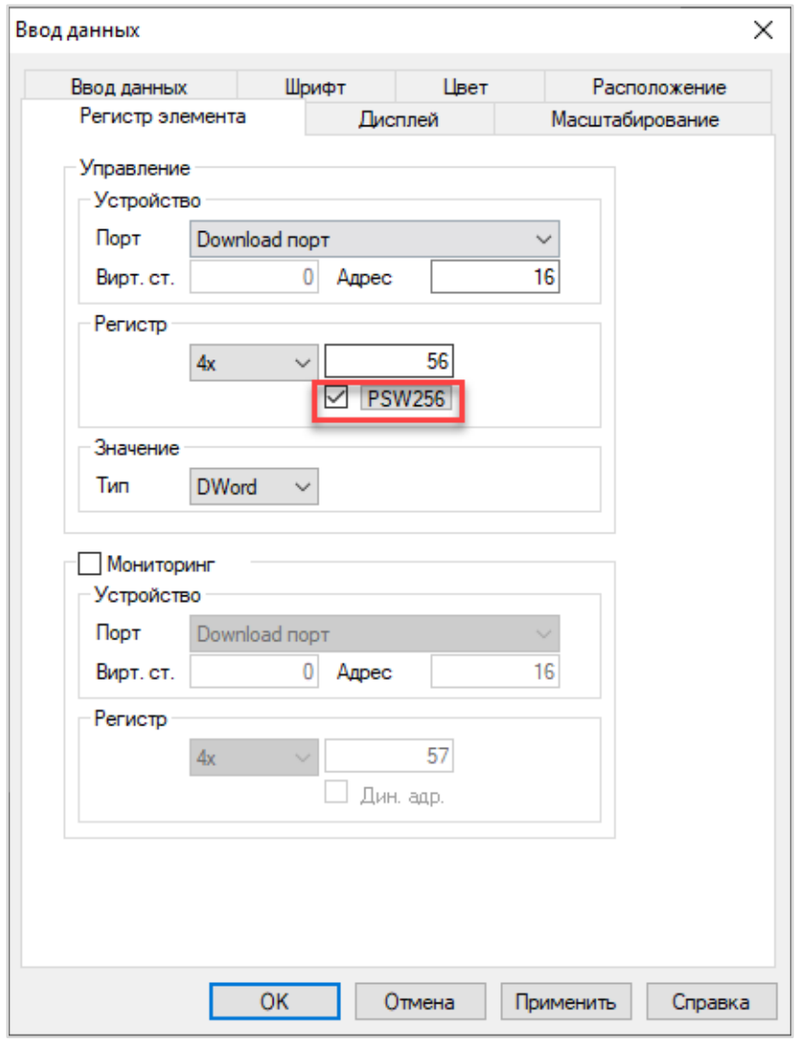


В некоторых случаях такое автоматическое формирование групповых запросов может быть неудобным.

Например, если в первом примере в slave–устройстве нет holding–регистров с адресами **51–55** (хотя бы одного из них), то в ответ на групповой запрос будет отправлен ответ с кодом ошибки **0x02 (ILLEGAL DATA ADDRESS)**. В этом случае нужно отключить автоматическое формирование групповых запросов. Если регистр элемента не

должен включаться в состав группового запроса, то в его настройках нужно установить галочку **Дин. адр**. и указать регистр, который не используется в проекте панели (т. е. регистра, значение которого всегда равно **0**).

В рамках нашего примера мы сделаем это для второго цифрового ввода с адресом регистра **4x56** – в качестве динамического адреса укажем локальный регистр панели PSW256, который мы вообще не используем в нашем проекте.



Теперь панель будет формировать два отдельных запроса, несмотря на «близость» адресов регистров, привязанных к нашим элементам.

> ***Примечание:*** *выше рассмотрен только частный случай использования динамической адресации для отключения автоматического формирования групповых запросов. Основная задача механизма динамической адресации – предоставить пользователю возможность динамически (в процессе работы проекта) менять адреса регистров, привязанных к элементам экрана – например, чтобы с помощью одного набора элементов обеспечить возможность работы с группой однотипных технологических объектов. Более подробно механизм динамической адресации рассмотрен в Руководстве пользователя программы Конфигуратор СП300.*

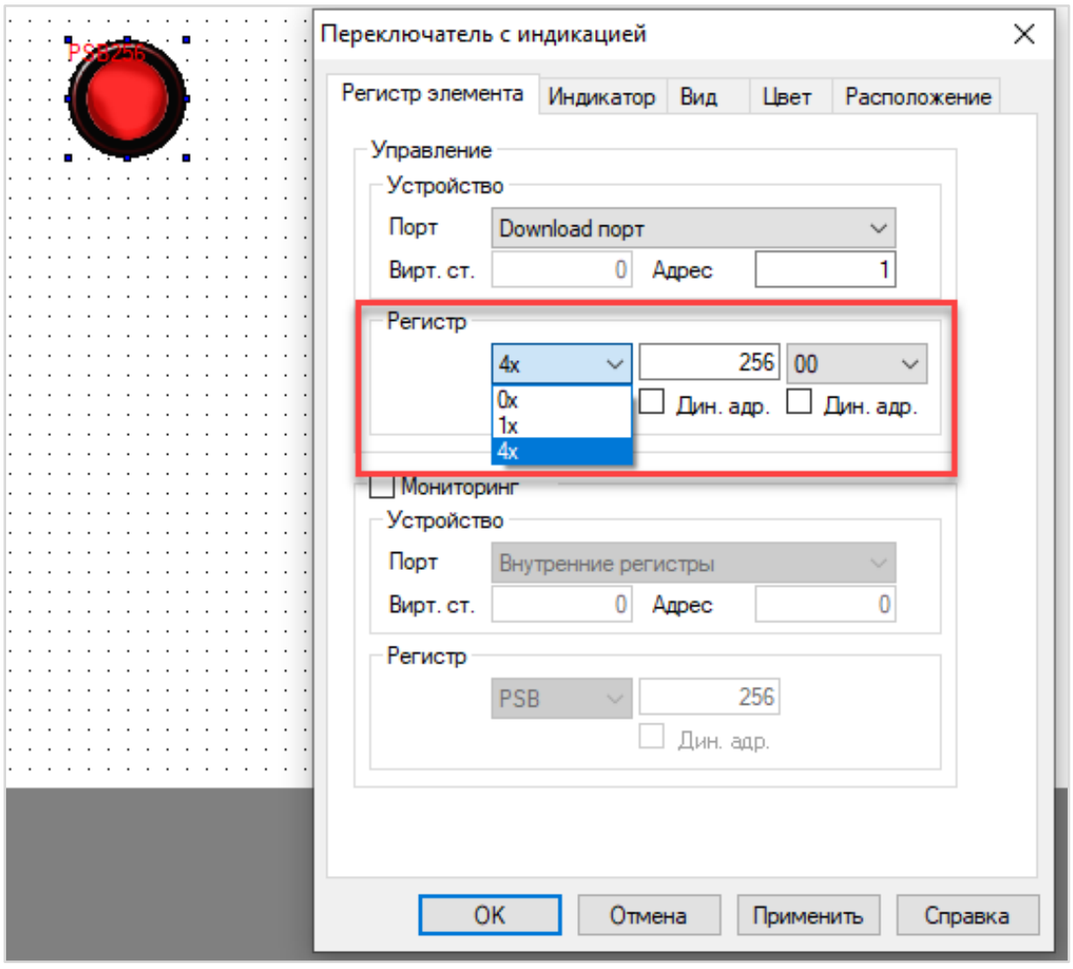
Битовые элементы и функции Modbus

В прошлых шагах мы говорили про элемент **Цифровой ввод**.

Теперь рассмотрим элемент **Переключатель с индикацией,** используемый для отображения и изменения состояния битового параметра.

В нём доступно три области памяти:

- **0x** – coils. Для чтения используется функция **0x01 (Read Coils)**, для записи – **0x05 (Write Single Coil)**;
- **1x** – discrete inputs. Для чтения используется функция **0x02 (Read Discrete Inputs)**, запись не поддерживается – потому что такие биты доступны только для чтения;
- **4x** – бит holding–регистра. Для чтения используется функция **0x03 (Read Holding Registers)**, для записи – функция **0x06 (Write Single Register)**. В рамках чтения происходит чтение holding–регистра с указанным адресом и отображение состояния его бита с указанным номером. В рамках записи – в считанном holding–регистре изменяется значение бита с указанным номером, после чего происходит запись holding–регистра с указанным адресом.



> ***Примечания**:*
>
> - *доступ к битам input–регистров не поддерживается (вариант **3x** отсутствует). Для преодоления этого ограничения можно использовать опрос через функциональную область, который мы рассмотрим в следующих шагах;*
> - *в случае побитового доступа к holding–регистру – запись всегда производится функцией **0x06 (Write Single Register)**. Запись функцией **0x10 (Write Multiple Registers)** не поддерживается;*
> - *запись бита функцией **0x0F (Write Multiple Coils)** не поддерживается.*

Битовые элементы тоже подвержены автоматическому формированию групповых запросов. При их опросе панель создаёт групповые запросы с кратностью адресов и количества опрашиваемых бит = 8.

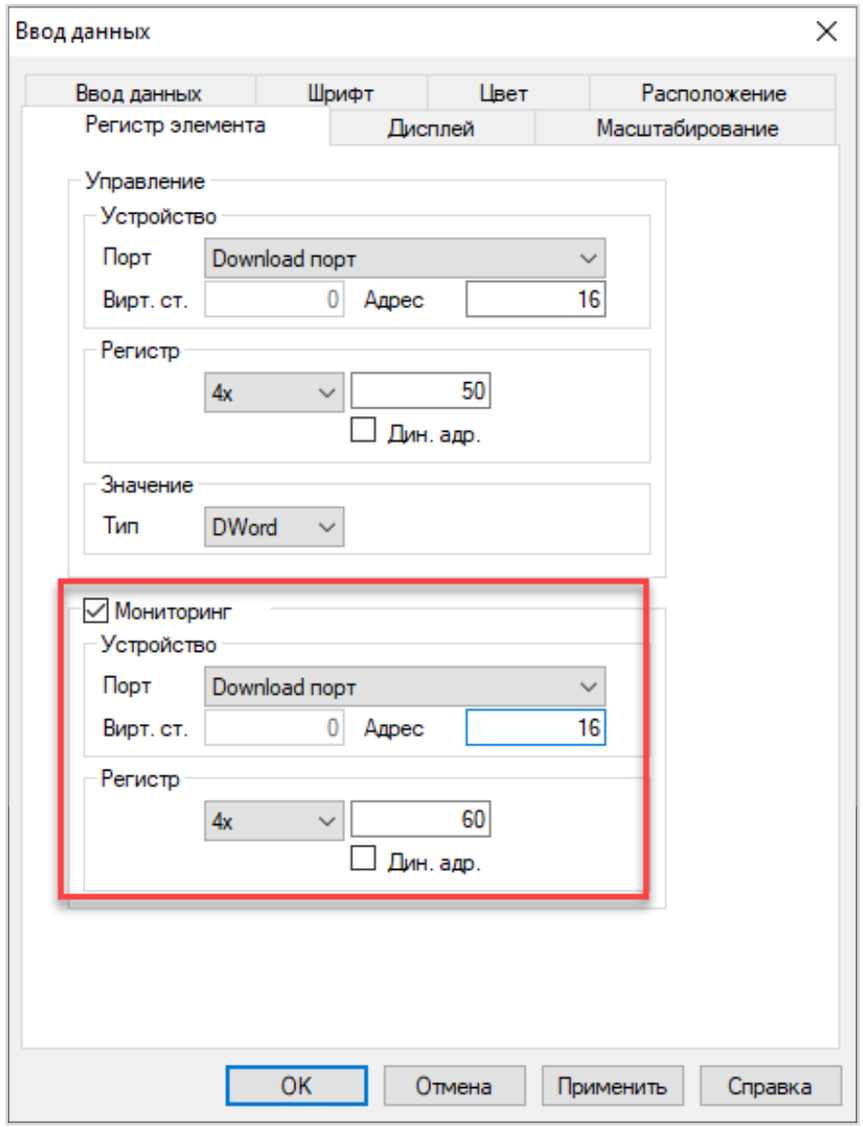
Пример: на экране расположено 6 переключателей с привязанными битами slave–устройства – 0x0, 0x21, 0x27, 0x28, 0x30 и 0x36. Панель сформирует два запроса к slave–устройству: первый – на чтение 8 бит с начиная с с адреса 0x0, второй – на чтение 24 бит начиная с адреса с 0x16.

Опция Мониторинг

По умолчанию элемент записи (например, **Цифровой ввод** или **Переключатель с индикацией**) отображают и производят запись одного и того же регистра (или бита). В некоторых случаях требуется, чтобы элемент отображал значение одного регистра, а запись производил в другой. Для этого в настройках элемента нужно установить галочку **Мониторинг**.

Настройки, расположенные под этой галочкой, относятся к параметру, который будет считываться и отображаться элементом.

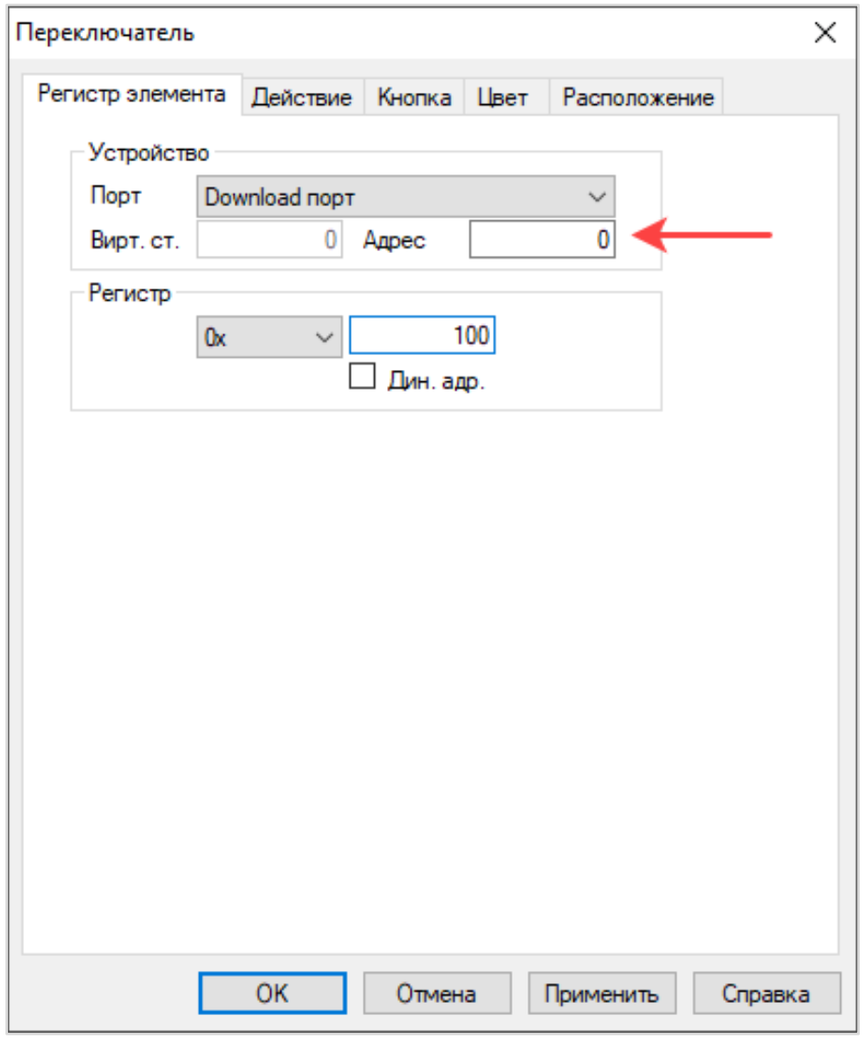
Настройки, расположенные выше (в области **Управление**), относятся к параметру, который будет записываться элементом.



Широковещательный запрос (broadcast) для Modbus RTU / ASCII

Для настройки широковещательного запроса достаточно указать адрес опрашиваемого slave–устройства = 0.

Следует учесть, что этот адрес разумно устанавливать только в элементах, не формирующих запросы чтения (например, в элементе **Переключатель** с действием **ВКЛ, пока нажат**) – потому что широковещательные запросы используются только для функций записи (вспомните урок 2.12).



Видеоуроки по настройке опроса через элементы

https://vkvideo.ru/video–68314714_456239062

https://vkvideo.ru/video–68314714_456239064

https://vkvideo.ru/video–68314714_456239065

Ограничения опроса через элементы

Опрос через элементы имеет следующие ограничения:

- нельзя настроить период опроса;
- нельзя настроить циклическую запись данных;
- отсутствует возможность детальной настройки групповых запросов;
- отсутствует возможность использовать считанные данные в проекте панели.

Остановимся подробнее на последнем пункте.

В ряде случаев требуется обработать считанные данные в макросе или сделать их доступными для master–устройства, подключённого к другому интерфейсу панели (например, панель в режиме **Modbus RTU Master** опрашивает модули ввода–вывода, и должна предоставить доступ к этим данным контроллеру, для которого она работает в режиме **Modbus TCP Slave**).

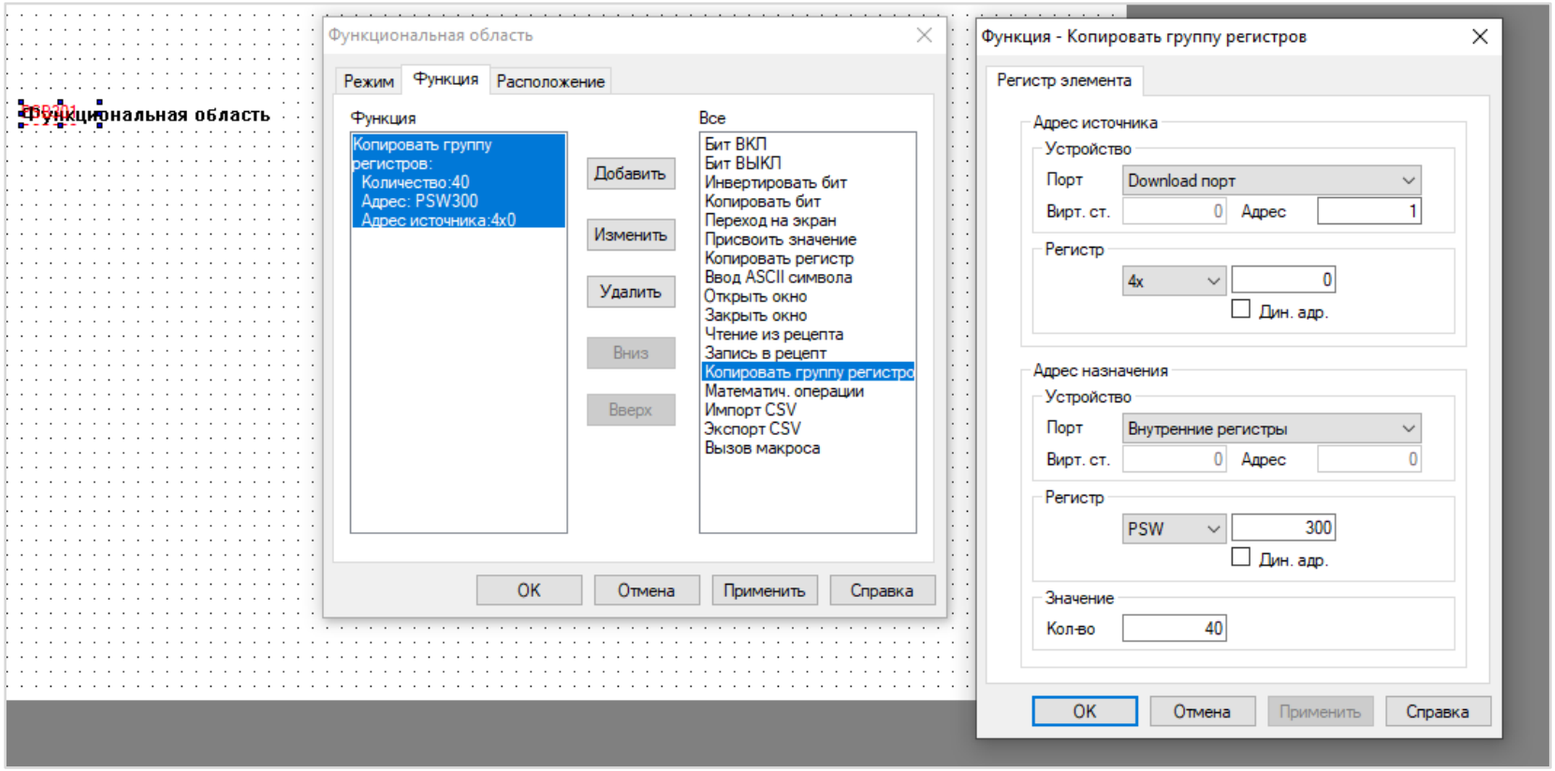
Некоторые из перечисленных выше ограничений можно обойти, если настроить опрос через функциональную область.

Как это сделать – мы рассмотрим в следующем шаге.

Функциональная область. Часть 1

Функциональная область – это невизуализируемый элемент, который используется для выполнения заданных действий (например, записи значения в регистр, перехода на другой экран и т. д.) при наступлении заданных событий (например, после включения панели, по фронту заданного бита или с заданной периодичностью).

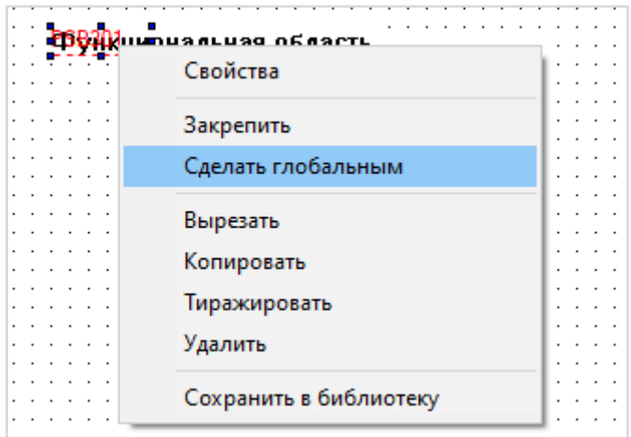
С точки зрения настройки обмена наибольший интерес представляет действие **Копировать группу регистров**.



Это действие копирует заданное количество регистров с адреса источника по адресу назначения. Возможны следующие варианты:

- адрес источника – регистры slave–устройства, адрес назначения – регистры панели.
  Выполняемая операция: чтение регистров slave–устройства.
- адрес источника – регистры панели, адрес назначения – регистры slave–устройства.
  Выполняемая операция: запись регистров slave–устройства.
- адрес источника – регистры первого slave–устройства, адрес назначения – регистры второго slave–устройства.
  Выполняемая операция: чтение регистров первого slave–устройства и их запись в регистры второго slave–устройства.
- адрес источника – регистры панели, адрес назначения – регистры панели.
  Выполняемая операция: копирование значений из одних регистров панели в другие, без выполнения Modbus–запросов.

Довольно часто требуется, чтобы действия, настроенные в функциональной области, выполнялись независимо от того, какой экран в данный момент отображается на панели. В этом случае нужно сделать функциональную область **глобальной** – после этого она «пропечатается» на всех экранах проекта:



> ***Примечание**: функциональная область не содержит действий для чтения и записи группы бит. Для чтения и записи **отдельных** бит можно использовать действия **Бит ВКЛ**, **Бит ВЫКЛ** и **Инвертировать бит**. Действие **Копировать бит** предназначено для копирования одного бита, но формируемый им Modbus–запрос производит чтение и/или запись **8 бит.***

Функциональная область. Часть 2

В настройках функциональной области на вкладке **Режим** указывается условие выполнения настроенных действий.

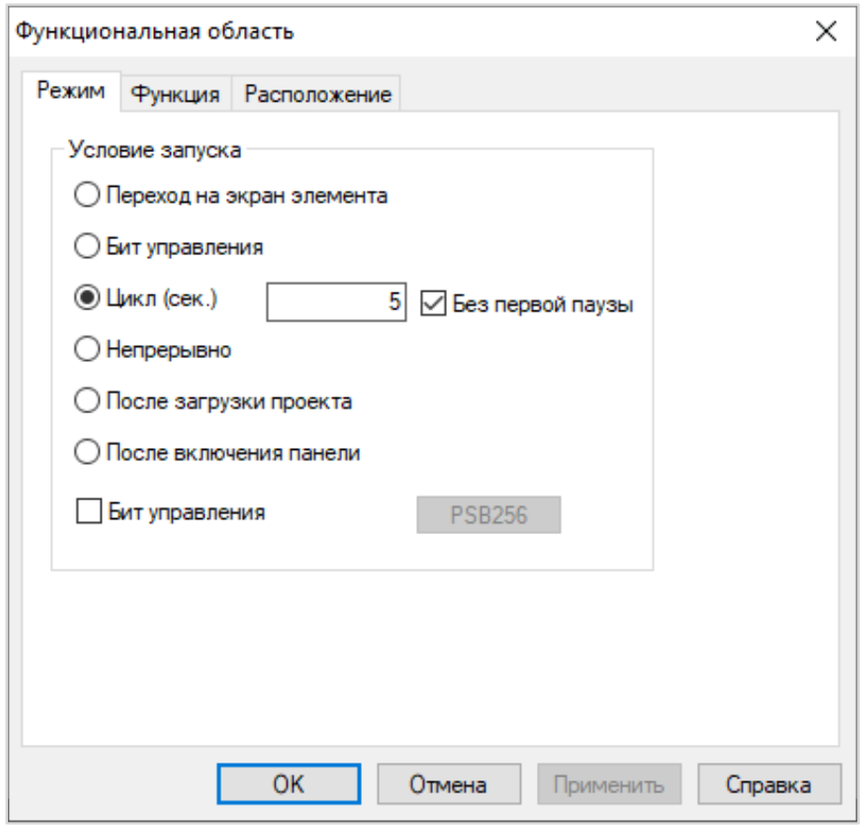
Но если в рамках действия считываются регистры slave–устройства (т. е. это действие формирует Modbus–запрос с функцией чтения) – то оно выполняется циклически независимо от настроенного на вкладке **Режим** условия. При этом бит управления тоже не влияет на действие – то есть его выполнение невозможно приостановить.

У пользователя нет возможности повлиять на период выполнения такого действия.

Для оценки (в рамках проекта с одной функциональной областью, которая формирует один запрос чтения):

- последовательный порт, скорость обмена = 115200, задержка отправки = 0: период опроса составляет 50 мс;
- Ethernet: период опроса составляет 15 мс.

Что касается действий, формирующих Modbus–запросы с функцией записи – они выполняются согласно условию, настроенному на вкладке **Режим**.



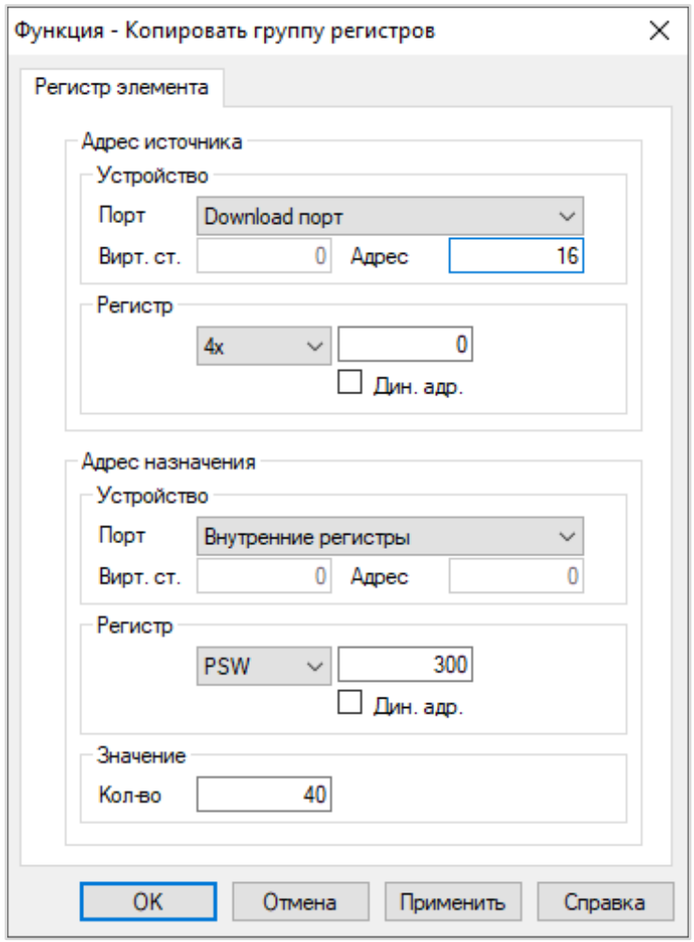
Функциональная область. Часть 3

В настройках действия **Копировать группу регистров** пользователь указывает количество считываемых (или записываемых) регистров. На основании этого значения панель формирует один или несколько групповых запросов, в каждом из которых выполняется чтение (или запись) не более 16 регистров.

Соответственно, при чтении **40** регистров начиная с регистра **0** будет сформировано три запроса:

- на чтение регистров 0–15;
- на чтение регистров 16–31;
- на чтение регистров 32–40.

Полученные значения будут размещены в локальных регистрах панели **PSW300...339**. Далее их можно будет обработать в макросе, предоставить для опроса другому устройству, которое будет выполнять по отношению к панели роль **Modbus Master** и т. д. Кроме того, можно будет привязать к индикаторам и другим элементам отдельные биты этих регистров.



Фрагмент лога обмена со slave–устройством (в HEX):

```
27-06-2025 07:40:19.063 Node1::Device4:(COM2) Tx: [0021] 10 03 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 1D 66
27-06-2025 07:40:19.063 Node1::Device4:(COM2) Rx: [0008] 10 03 00 20 00 08 46 87
27-06-2025 07:40:19.016 Node1::Device4:(COM2) Tx: [0037] 10 03 20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 6F 78
27-06-2025 07:40:19.016 Node1::Device4:(COM2) Rx: [0008] 10 03 00 10 00 10 46 82
27-06-2025 07:40:18.969 Node1::Device4:(COM2) Tx: [0037] 10 03 20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 6F 78
27-06-2025 07:40:18.969 Node1::Device4:(COM2) Rx: [0008] 10 03 00 00 00 10 47 47
```

Функциональная область. Часть 4

Вспомним, что в настройках интерфейсов присутствует галочка **Изменить порядок регистров**. При использовании функциональной области – состояние этой галочки не учитывается. Поменять порядок регистров можно только с помощью дополнительных действий, настроенных в функциональной области (например, путём вызова макроса).

> ***Примечание**: некоторые slave–устройства не поддерживают функцию **0x06**. В этом случае в панели можно активировать использование функции **0x10** для записи значения типа **Word**. Для этого нужно установить бит соответствующего системного регистра:*
>
> - *Download–порт: PFW26.2*
> - *PLC–порт: PFW36.2.*
>
> *Эти биты влияют в том числе и на команды записи, настроенные в функциональной области.*

Ограничения опроса через функциональную область

Вспомним, что **опрос через элементы** имеет следующие ограничения:

- ***нельзя настроить период опроса***;
- **нельзя настроить циклическую запись данных;**
- отсутствует возможность детальной настройки групповых запросов;
- **отсутствует возможность использовать считанные данные в проекте панели.**

Настройка опроса через функциональную область частично позволяет обойти первое ограничение (только для запросов записи) и полностью обойти второе и четвёртое.

Чтобы обойти оставшиеся ограничения – потребуется настроить обмен через макросы. Как это сделать – мы рассмотрим в следующем уроке.

Видеоурок по настройке обмена с помощью функциональной области

https://vkvideo.ru/video–68314714_456239066

4.3 Настройка опроса через макросы

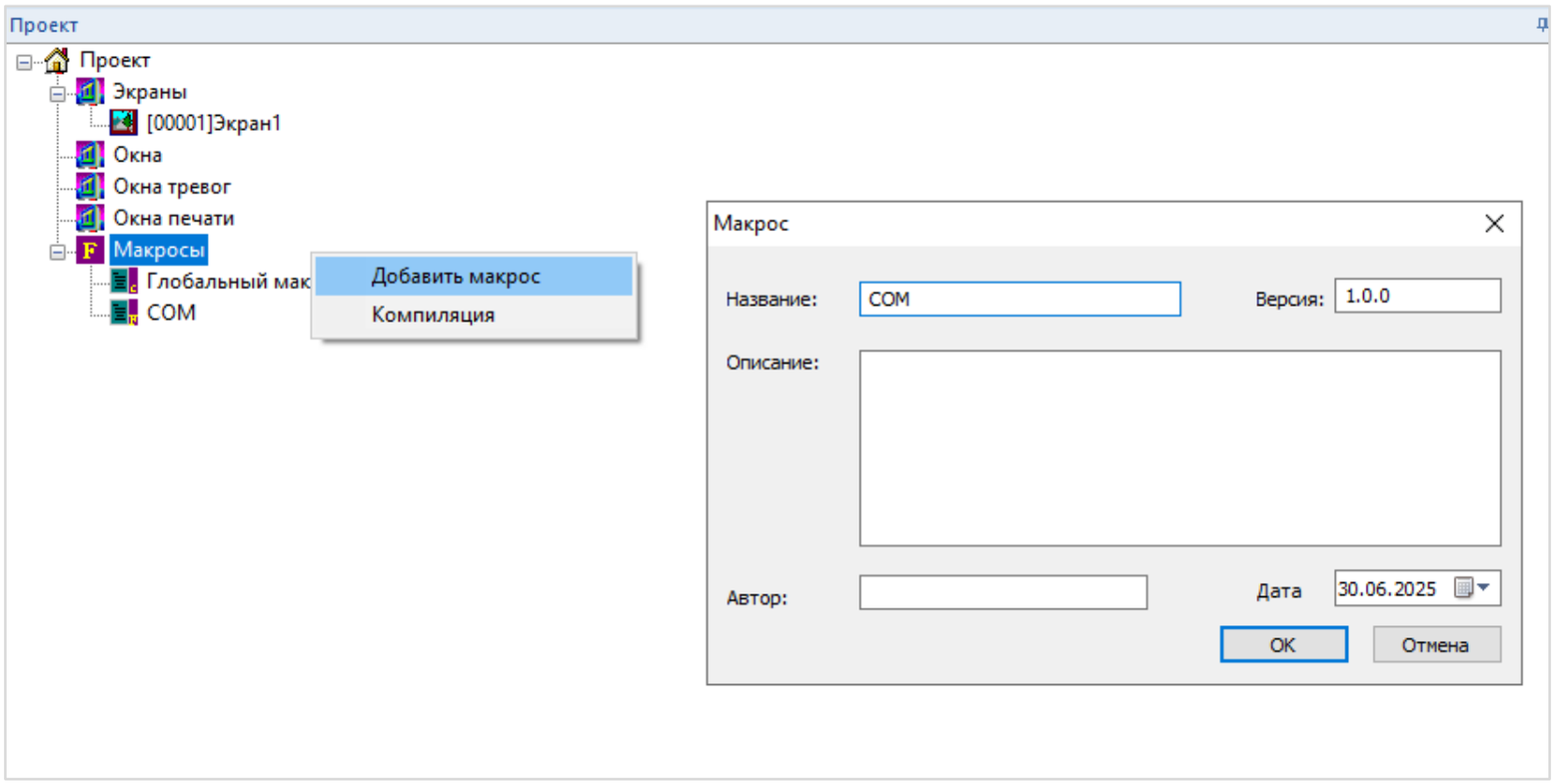
Основная информация

Макросы – это программы на языке С, которые позволяют реализовать в панели обработку данных и выполнение алгоритмов.

> *Используемая версия стандарта – C99.*

В дереве проекта отображается вкладка **Макросы**, которая по умолчанию содержит только один элемент – **Глобальный макрос**. В глобальном макросе можно создавать объекты, которые в дальнейшем будут использоваться в пользовательских макросах: функции, перечисления, структуры, глобальные переменные.

Если вы планируете использовать макросы – то вам нужен как минимум один пользовательский макрос. Для его создания нужно нажать правой кнопкой мыши на узел **Макросы** и использовать команду **Добавить макрос**. В рамках нашего урока мы создадим макрос с названием **COM**.



Нажмите два раза левой кнопкой мыши на этот макрос в дереве проекта, чтобы открыть редактор кода.

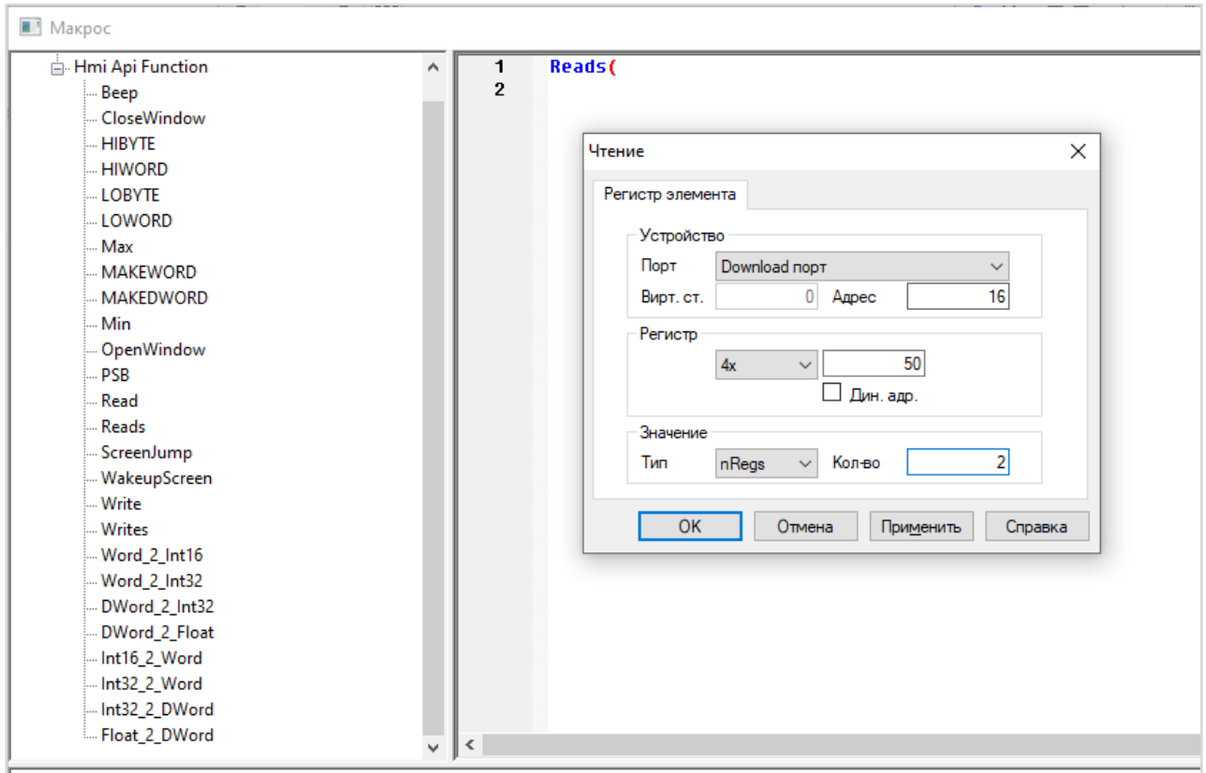
В левой части редактора отображается список «системных» функций. Четыре из них используются для отправки Modbus–запросов:

- функции **Read/Write** позволяют организовать чтение/запись одного параметра;
- функции **Reads/Writes** позволяют организовать чтение/запись одного или нескольких параметров.

В случае работы с регистрами – можно всегда использовать только функции **Reads/Writes**. Но они не поддерживают битовые функции Modbus – и тут пригодятся **Read/Write**.

Повторим пример из предыдущего урока – считаем значение с плавающей точкой (типа **Float**), которое хранится в **holding**–регистрах **50–51** slave–устройства с адресом **16**, которое подключено к **Download–порту** панели.

Для этого введём название функции **Reads** и поставим открывающую скобку – после чего откроется окно настройки параметров функции. Присвоим им нужные нам значения.




```
/*******************************************************************************
Функция: Reads

PortName: используемый порт (DOWNLOAD, PLC, NET_0 … NET_5, HMI_LOCAL_MCH)

SlaveID: адрес slave устройства

DeviceTypeReg: тип регистров slave устройства
(Modbus_TCP_REG_3X, Modbus_TCP_REG_4X, Modbus_RTU_REG_3X, Modbus_RTU_REG_4X)

Adr: адрес первого считываемого регистра

Length: количество считываемых регистров

ArrVar: имя массива, в который записываются считанные данные

Функция возвращает:  TRUE /  FALSE (чтение произведено успешно / чтение не удалось)
```

После нажатия на кнопку **ОК** будет сформирована следующая строка:

```
Reads(DOWNLOAD, 16, MODBUS_RTU_REGS_4X, 50, 2, ***);
```

Вместо *** нужно ввести адрес объекта памяти, в котором будут размещены считанные данные.

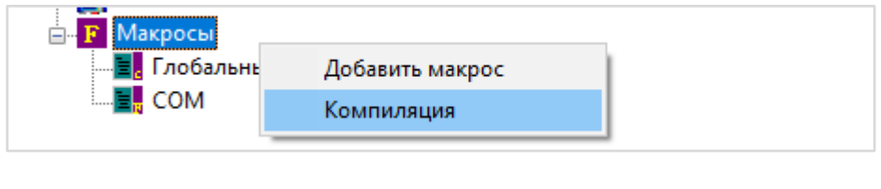
В языке С для получения адреса объекта используется оператор &.

Разместим считанное значение в локальных регистрах PSW300...301:

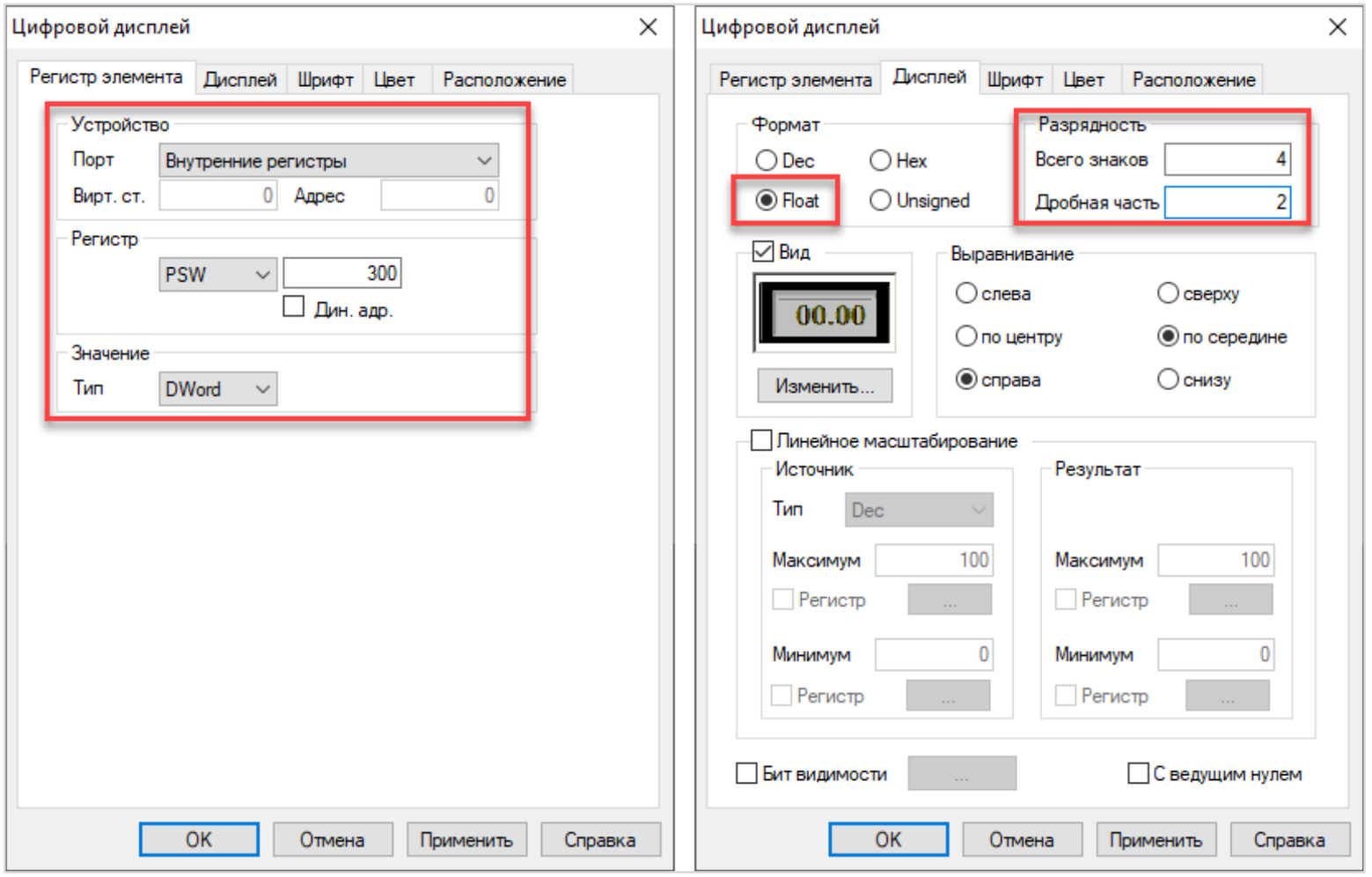
```
Reads(DOWNLOAD, 16, MODBUS_RTU_REGS_4X, 50, 2, &PSW[300] );
```

> ***Примечание**: при размещении в локальных регистрах панели параметров, размер которых превышает один регистр, начальный адрес обязательно должен быть **чётным** (т. е. нельзя было записать наше значение типа **Float** в локальные регистры **PSW301–302**). Это связано с особенностями выравнивания памяти панели.*

Для проверки корректности кода макросов нужно нажать на узел **Макросы** правой кнопкой мыши и выполнить команду **Компиляция**. Если в коде макроса допущены ошибки – то появится окно с их перечнем.



Добавим на экран элемент **Цифровой дисплей**, чтобы отобразить считанное в макросе значение:



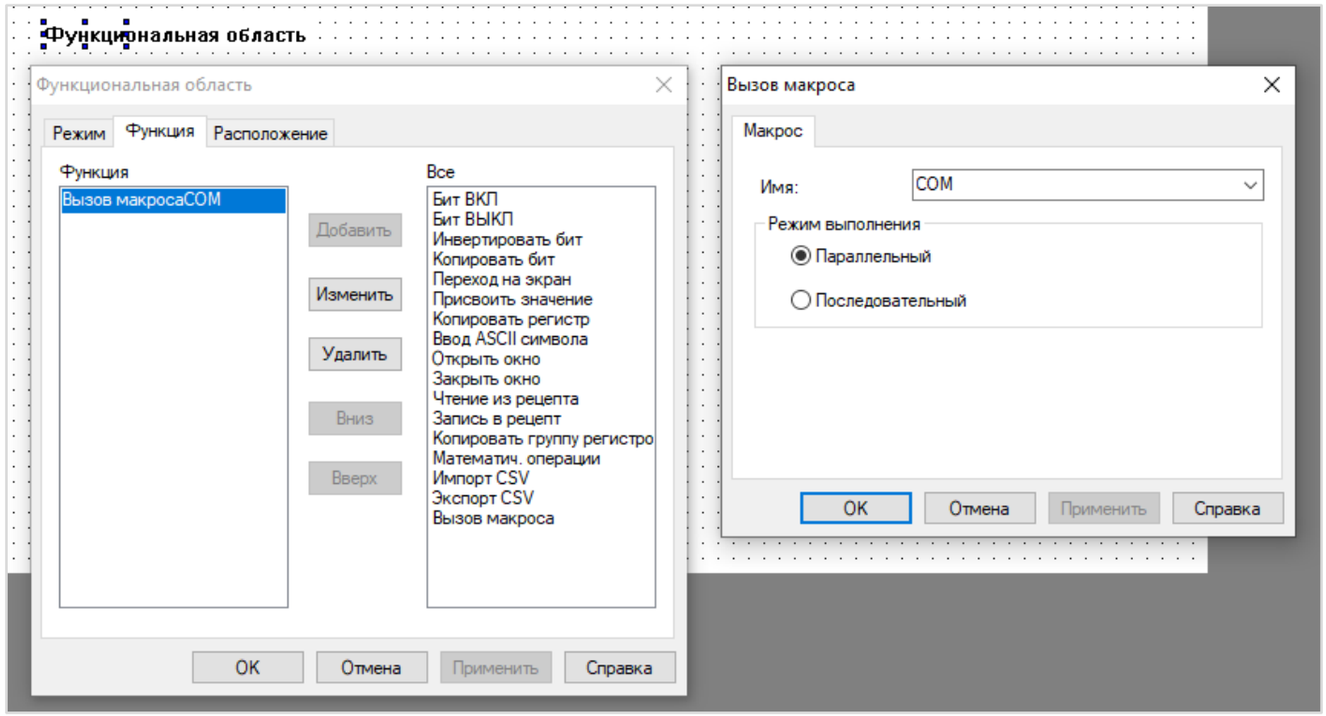
Мы создали макрос **COM**, но он пока что нигде в проекте не вызывается.

Чтобы его вызвать – добавим на экран функциональную область и настроим в ней действие **Вызов макроса**, в котором укажем наш макрос **COM**.

Режим выполнения определяет, как будет выполняться макрос по отношению к другим макросам и задачам панели (например, задаче отрисовки элементов):

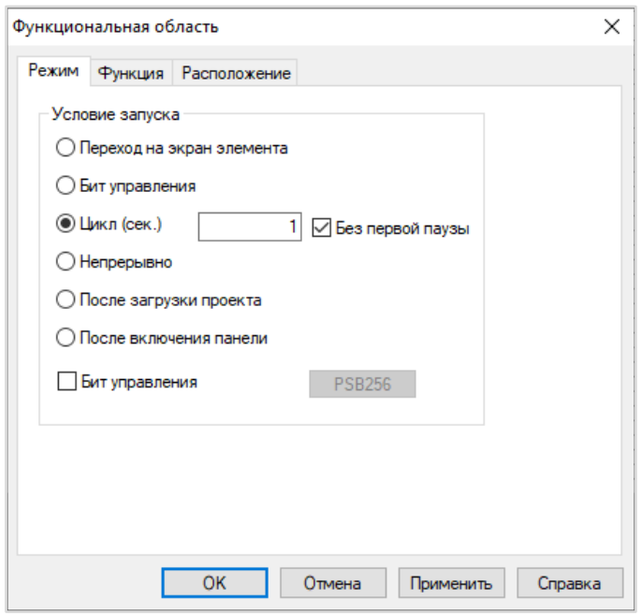
- в последовательном режиме все остальные макросы и задачи будут блокироваться на время выполнения данного макроса;
- в параллельном режиме данный макрос будет выполняться параллельно (асинхронно) по отношению к другим макросам и задачам.

Рекомендуется весь код, связанный с обменом, вынести в один макрос, выполняемый параллельно.

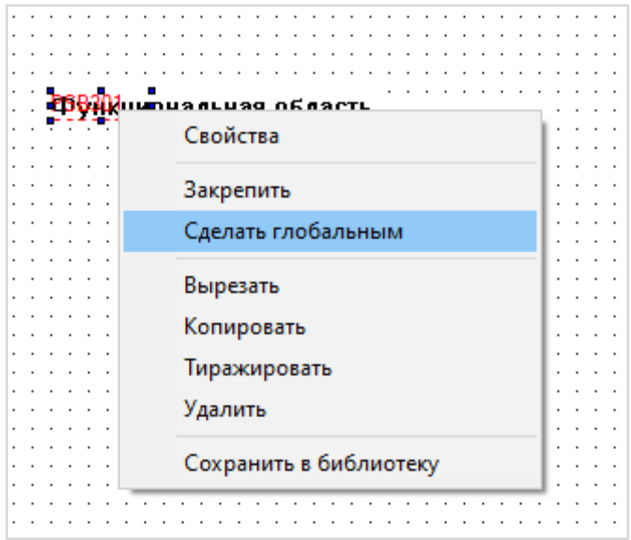


Важно отметить, что макрос выполняется только при выполнении условия, заданного в настройках режима функциональной области.

То есть в отличие от действия **Копировать группу регистров** – в данном случае у нас есть контроль над периодом опроса и мы можем останавливать/возобновлять его по биту управления.



Довольно часто требуется, чтобы макросы, вызываемые в функциональной области, выполнялись независимо от того, какой экран в данный момент отображается на панели. В этом случае нужно сделать функциональную область глобальной – после этого она «пропечатается» на всех экранах проекта:



Особенности обработки макросов

Вспомним, что в настройках интерфейсов присутствует галочка **Изменить порядок регистров**. При использовании функциональной области – состояние этой галочки не учитывается. Это распространяется и на макросы. При необходимости – можно переставить регистры считанного значения прямо в коде макроса:

> *Для указания адреса массива не используется оператор &, потому что в языке C имя массива соответствует начальному адресу памяти, по которому размещены его данные. Впрочем, вы могли бы в примере ниже использовать запись типа &rawData – это не повлияло бы на его работу.*

```
1   WORD rawData[2];
2   WORD SwapData[2];
3
4   Reads(DOWNLOAD, 16, MODBUS_RTU_REGS_4X, 50, 2, rawData);
5
6
7   SwapData[0] = rawData[1];
8   SwapData[1] = rawData[0];
9
10  *(float*)(PSW+300) = *(float*)(SwapData);
11
```

Наибольший интерес представляет последняя строка – в ней демонстрируется, как использовать преобразование типов (type casting) в коде макроса. В данном случае мы считываем значение в массив из двух WORD–переменных (каждая из которых соответствует одному из регистров Modbus), потом копируем их в другой массив с перестановкой, а затем записываем в локальные регистры панели – и в этот момент нам нужно указать компилятору, что наш массив WORD'ов на самом деле является значением типа Float.

***

Синтаксис функции **Writes** не отличается от **Reads**. В качестве последнего аргумента передаётся адрес памяти, в которой размещаются записываемые данные.

Некоторые slave–устройства не поддерживают функцию **0x06 (Write Single Register)**. Функции **Write/Writes** всегда используют эту функцию, если количество записываемых регистров = 1. Системные биты **PFW26.2** и **PFW36.2** не влияют на это поведение. Если требуется организовать в макросе запись одного регистра с помощью функции **0x10 (Write Multiple Registers)** – то потребуется реализовать формирование Modbus–запросов и разбор ответов; как это сделать – описано в данной статье и примере.

Групповые запросы в макросах

В отличие от опроса через элементы и действие **Копировать группу регистров** – для каждого вызова функции **Reads/Writes** будет сформирован один групповой Modbus–запрос на чтение/запись указанного количества регистров.

Но если указанное количество регистров превысит **120** – то будет сформировано несколько Modbus–запросов, в рамках каждого их которых будет считано/записано не более 120 регистров.

Обработка ошибок и управление обменом в макросах

Макросы предоставляют пользователю полный контроль над обменом:

- можно изменять значения аргументов функций **Reads** и **Writes** в процессе работы проекта – например, предоставить оператору возможность изменения адреса опрашиваемого slave–устройства;
- можно определять, успешно ли завершился каждый сеанс обмена. Если в ответ на запрос был получен корректный ответ (без кода ошибки Modbus) – то функция вернёт значение **TRUE**. Если ответ содержит код ошибки или вообще не был получен – то функция вернёт значение **FALSE**;
- можно отключать/включать вызов определённой функции (т. е. отправку определённого Modbus–запроса) – например, остановить опрос slave–устройства, выведенного в ремонт. Если требуется остановить обмен со всеми slave–устройствами – то проще воспользоваться битом управления функциональной области, в которой вызывается макрос.

Приведём пример, в котором демонстрируется диагностика и управление обменом:

```
1   if (PSW[400] == 0)
2   {
3
4       if (Reads(DOWNLOAD, 16, MODBUS_RTU_REGS_4X, 50, 2, &PSW[300] ) )
5       {
6           PSW[401] = PSW[401] + 1;
7       }
8       else
9       {
10          PSW[402] = PSW[402] + 1;
11      }
12
13  }
```

Функция **Reads** вызывается только в том случае, если локальной регистр **PSW400** имеет значение **0**. Если он имеет какое–то иное значение – то опрос прекращается. Например, можно добавить на экран **Переключатель с индикацией** с привязанным битом **PSW400.0** – чтобы с помощью него оператор мог остановить и возобновить отправку данного запроса.

Вызов функции **Reads** заключён в оператор **if**.

Если в ответ на запрос будет получен корректный ответ – то значение локального регистра **PSW401** будет увеличено на 1. Если ответ не будет получен или будет получен ответ с кодом ошибки Modbus – то значение локального регистра **PSW402** будет увеличено на 1.

Таким образом, **PSW401** представляет собой счётчик «успешных» запросов, а **PSW402** – счётчик «неуспешных».

Эти счётчики подвержены переполнению – после достижения значения 65535 счёт начнётся с 0. В случае необходимости – можно реализовать счётчики с типом DWORD с диапазоном значений 0...4294967295.

Заключение

Мы рассмотрели основы обмена через макросы. Этот способ несколько сложнее, чем настройка опроса через элементы или функциональную область с действием **Копировать группу регистров**. Но с другой стороны он позволяет:

- контролировать период опроса (в том числе, для операций чтения);
- детально настраивать групповые запросы;
- полностью контролировать обмен.

На этом мы заканчиваем рассматривать настройку панелей оператора СП3хх в режиме **Modbus Master**.

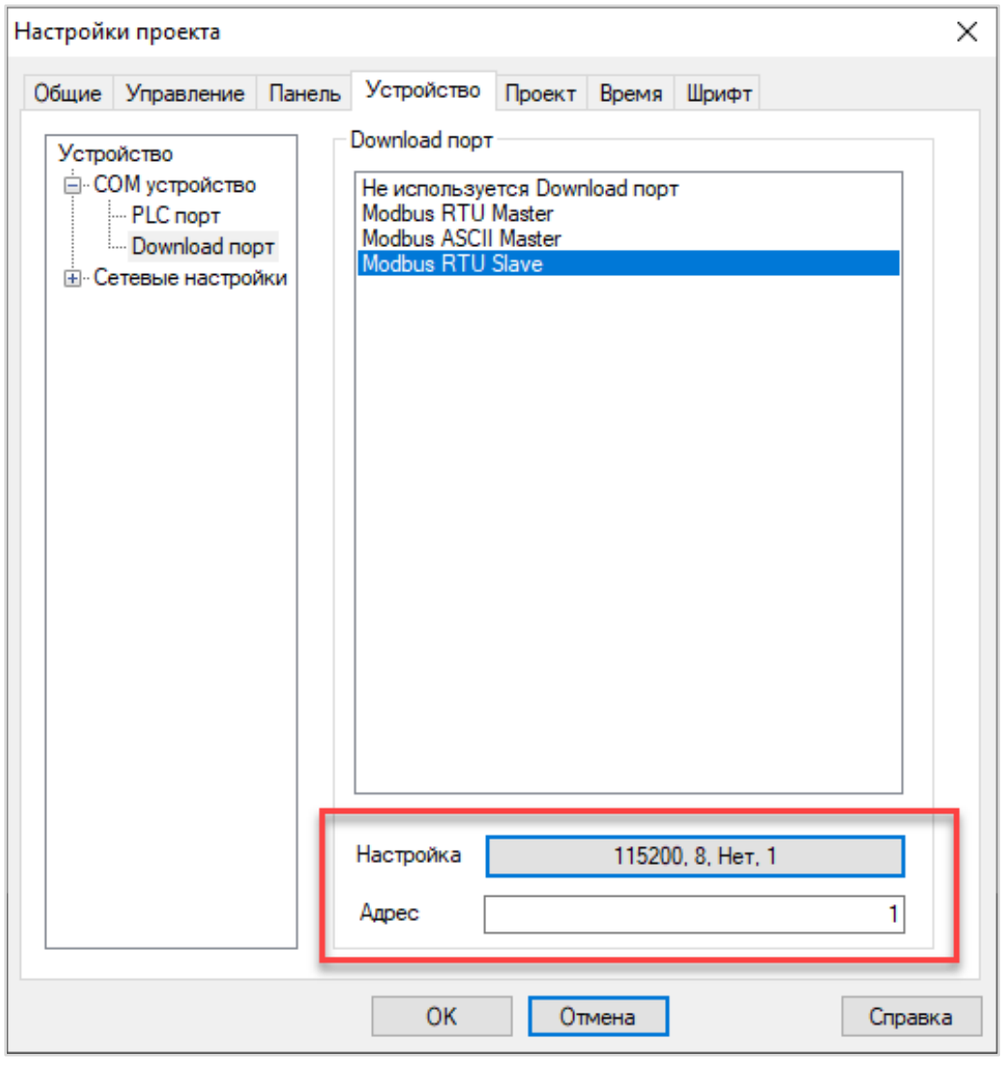
В следующем уроке настроим СП3хх в режиме **Modbus Slave**.

4.4 Настройка панели в режиме Modbus Slave

Для настройки панели в режиме **Modbus Slave** – нужно выбрать соответствующий пункт в настройках используемого последовательного интерфейса в меню **Файл – Настройки проекта – Устройство**. Доступные настройки интерфейса мы рассмотрели в уроке 4.1.

У расширенных модификаций панелей, имеющих интерфейс **Ethernet**, всегда активирован режим **Modbus TCP Slave** (используется порт **502**, поле **Unit ID** в запросе не валидируется, поддерживается до 10 одновременных клиентских подключений).

Сетевые настройки панели задаются в одноименной вкладке, которую видно на скриншоте ниже.



Области памяти

Панель имеет три области памяти:

Название	**Тип**	**Область Modbus**	**Диапазон**
PSB	оперативные биты	coils	256*...1023
PSW	оперативные регистры	holding–регистры	256...4095
PFW	энергонезависимые регистры	holding–регистры	256...55535**

Все эти области доступны по Modbus – к ним можно обратиться по любому из интерфейсов панели.

Оперативные биты и регистры не сохраняют свои значения после перезагрузки панели (или загрузки в неё нового проекта), а энергонезависимые – сохраняют.

*Биты и регистры с адресами < 256 являются системными – считать или записать их значения по Modbus не получится (в ответе будет указан код ошибки **0x02 (ILLEGAL DATA ADDRESS)**).

Все биты панели (**PSB**) с точки зрения Modbus относятся к области **coils**. Для их чтения используется функция **0x01 (Read Coils)**, для записи – **0x05 (Write Single Coil)** или **0x0F (Write Multiple Coils)**. В рамках одного группового Modbus–запроса можно прочитать или записать все доступные биты панели. Адресация бит является прямой – т. е. локальный бит **PSB256** является coil'ом с адресом **256**.

Все регистры панели с точки зрения Modbus относятся к области **holding**–регистров. Для их чтения используется функция **0x03 (Read Holding Registers)**, для записи – **0x06 (Write Single Register)** или **0x10 (Write Multiple Registers)**. В рамках одного Modbus–запроса можно прочитать или записать не более 120 регистров.

Для оперативных регистров (**PSW**) адресация регистров является прямой – т. е. локальный регистр **PSW256** является holding–регистром с адресом **256**.

Для энергонезависимых регистров (PFW**) адресация регистров в Modbus смещена на **+10000** – т. е. локальный регистр **PFW256** является holding–регистром с адресом **10256**, а локальный регистр **PFW55535** – holding–регистром с адресом **65535**. Так как спецификация Modbus указывает, что **65535** – это максимально возможный адрес регистра, то получить доступ по Modbus к локальным регистрам панели с адресом **PFW55536** и выше не получится (хотя фактически они имеются).

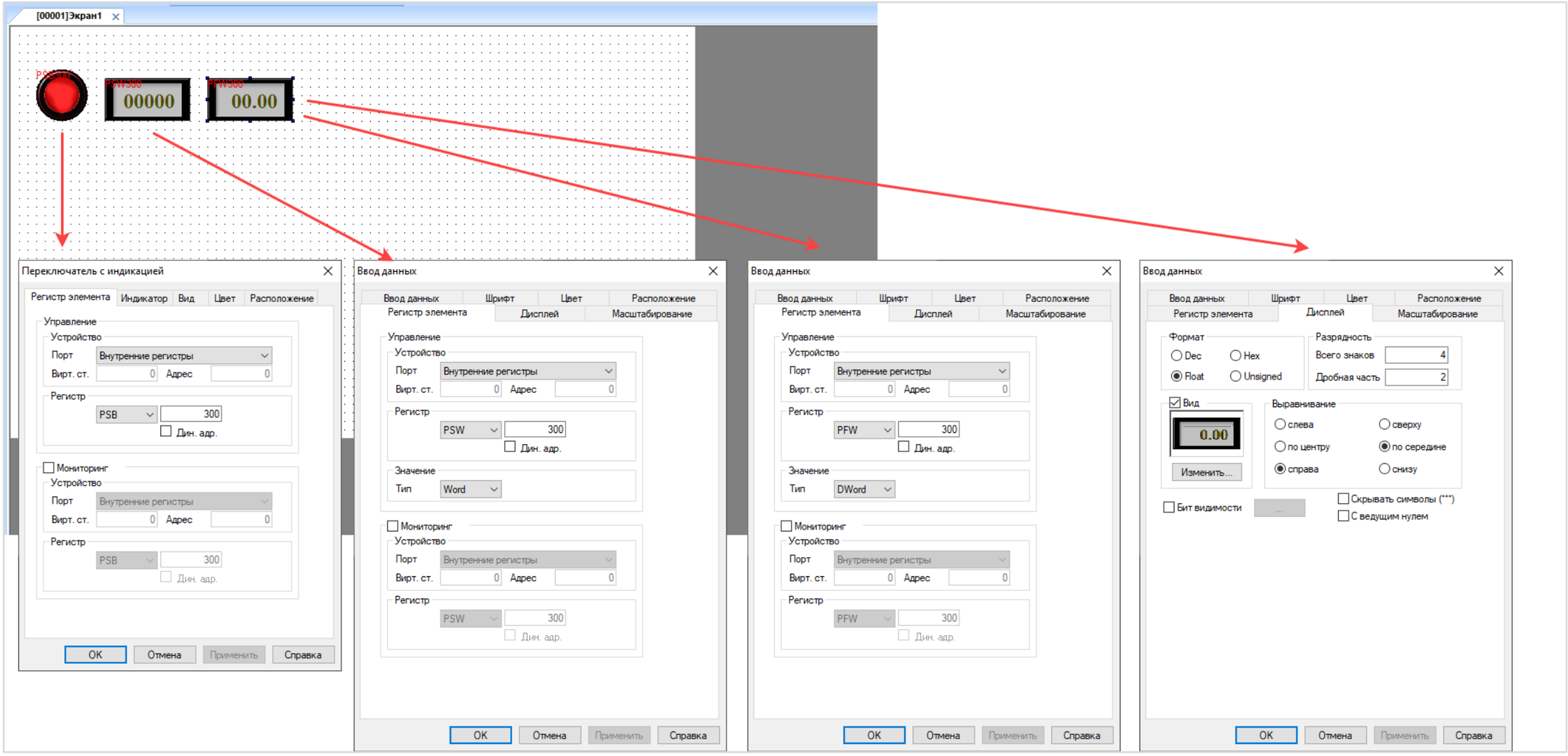
Для чтения и записи локальных битов и регистров необязательно, чтобы они были привязаны к элементам экрана; например, их значения могут формироваться в макросе.

Пример

Пусть наша панель работает в режиме **Modbus TCP Slave**.

Разместим на экране 3 элемента:

- переключатель с индикацией с привязанным локальным битом **PSB300**;
- цифровой ввод с привязанным локальным регистром **PSW300** (тип **Word**, формат **Unsigned**);
- цифровой ввод с привязанным локальным регистром **PFW300** (тип **Dword**, формат **Float**).



В качестве **Modbus Master** используем уже известный нам **MasterOPC Universal Modbus Server**. Настроим теги следующим образом:

Тег <<COILS>> : PSB300

Общие настройки		
Комментарий		
Включен в работу		True
Адрес	(0x012C)	300
Тип данных в устройстве		bool
Тип данных в сервере		bool
Тип доступа		ReadWrite
Скрипт		
Разрешение выполнения скрипта после чтения		False
Разрешение выполнения скрипта перед записью		False
Дополнительно		
Наличие отдельного регистра записи		False
Чтение сразу после записи		False
Сброс команды записи		True
HDA		
HDA доступ		False

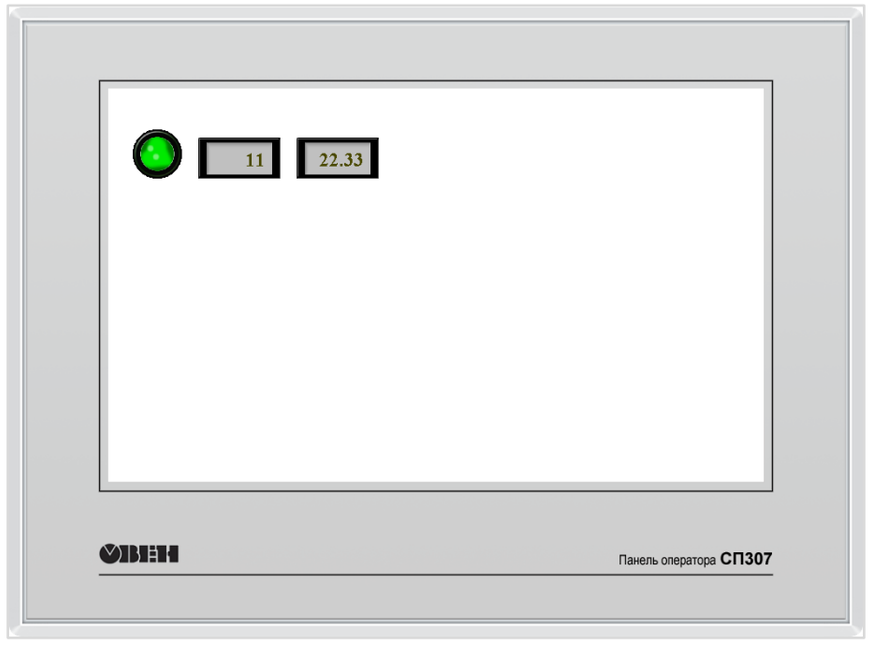
Тег <<HOLDING_REGISTERS>> : PSW300

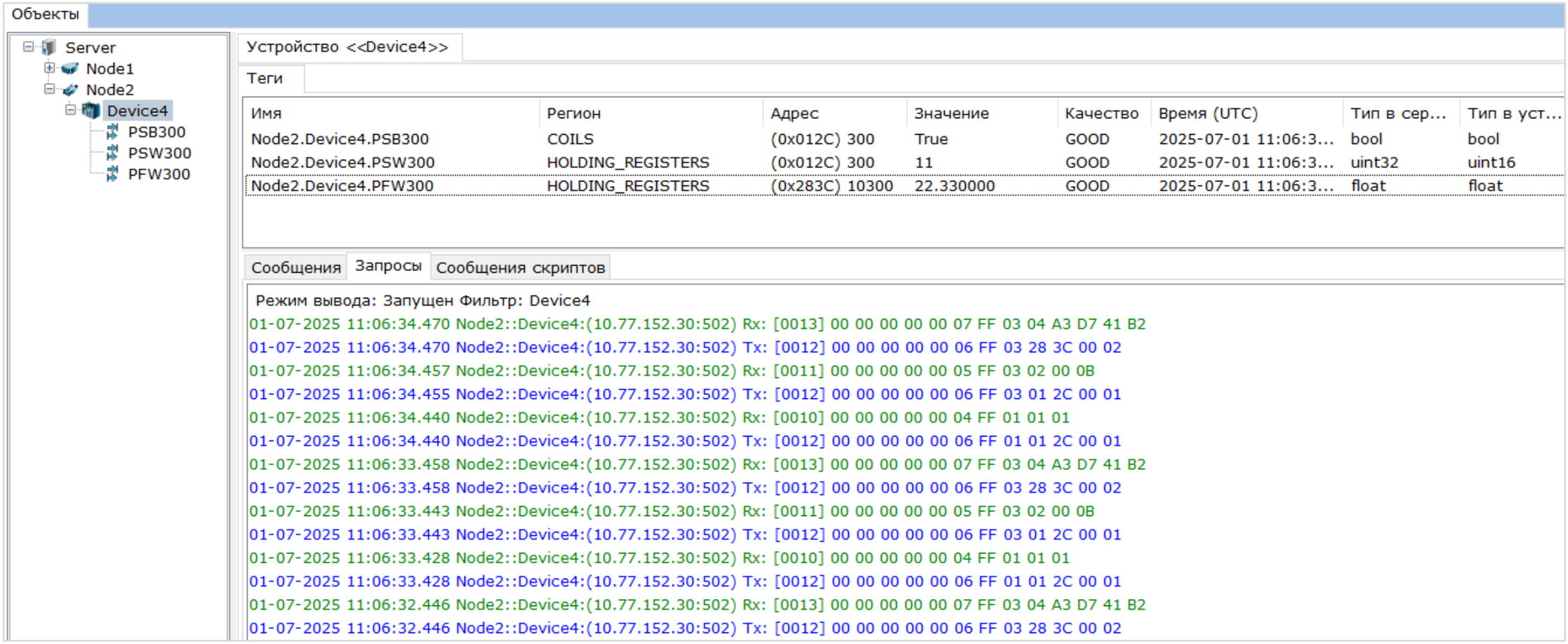
Общие настройки		
Комментарий		
Включен в работу		True
Адрес	(0x012C)	300
Тип данных в устройстве		uint16
Тип данных в сервере		uint32
Тип доступа		ReadWrite
Использовать перестановку байтов устройства		True
Последний тег в групповом запросе		False
Пересчет (A*X + B)		False
Скрипт		
Разрешение выполнения скрипта после чтения		False
Разрешение выполнения скрипта перед записью		False
Дополнительно		
Извлечение бита из данных		False
Наличие отдельного регистра записи		False
Чтение сразу после записи		False
Сброс команды записи		True
Принудительная запись командой 6		False
HDA		
HDA доступ		False

Тег <<HOLDING_REGISTERS>> : PFW300

Общие настройки		
Комментарий		
Включен в работу		True
Адрес	(0x283C)	10300
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
Скрипт		
Разрешение выполнения скрипта после чтения		False
Разрешение выполнения скрипта перед записью		False
Дополнительно		
Извлечение бита из данных		False
Наличие отдельного регистра записи		False
Чтение сразу после записи		False
Сброс команды записи		True
Принудительная запись командой 6		False
HDA		
HDA доступ		False

Демонстрация успешного обмена:





Видеоурок по настройке панели в режиме Modbus Slave

https://vkvideo.ru/video–68314714_456239063

4.5 Диагностика и управление обменом

Если панель работает в режиме **Modbus Master,** то в случае обрыва связи хотя бы с одним из slave–устройств период обновления экрана и время отклика существенно увеличиваются. При этом если опрос slave–устройства настроен через элементы – то в этих элементах будут отображаться последние успешно считанные значения.

Для диагностики связи можно воспользоваться системными окнами **60013** и **60014**. Для возможности открытия окна потребуется добавить в проект элемент **Кнопка открытия окна** или **Функциональная кнопка**.

Окно **60013** используется для расширенных модификаций панелей и содержит индикаторы, отображающие:

- статус связи в рамках Download–порта;
- статус связи в рамках PLC–порта;
- статусы связи для Modbus TCP Slave–устройств.

Окно **60014** используется для базовых модификаций панелей и содержит только индикаторы Download– и PLC–порта.

Зелёный цвет характеризует отсутствие ошибок обмена или же неиспользование данного порта/устройства в проекте. Красный цвет характеризует отсутствие связи с одним из устройств, подключенных к COM–порту, или же с одним из Modbus TCP Slave устройств.

Индикатор мигает (попеременно меняет цвет с зеленого на красный) в случае попытки восстановления связи.

Системные биты и регистры диагностики связи

К упомянутым в прошлом шаге индикаторам диагностики обмена привязаны системные биты:

- **PSB50** – индикатор диагностики Download–порта;
- **PSB51** – индикатор диагностики PLC–порта;
- **PSB54...59** – индикаторы диагностики для Modbus TCP Slave–устройств 1...6.

Пользователь может разместить их на экране панели, чтобы, например, видеть диагностическую информацию без необходимости открытия дополнительного окна.

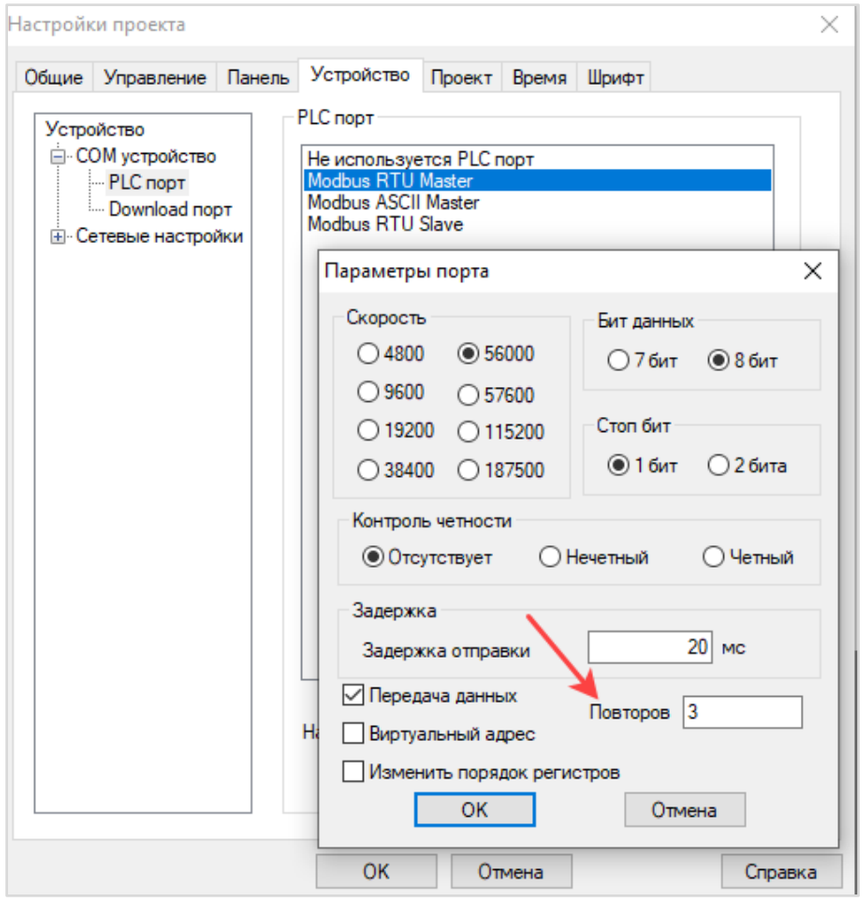
Кроме того, панель имеет ряд системных регистров, позволяющих получить более детализированную диагностическую информацию:

Dowload–порт	**PLC–порт**	**Описание**
PFW28	PFW38	Таймаут ожидания ответа, в миллисекундах
PSW60	PSW70	Количество Modbus–запросов, на которые был получен корректный ответ
PSW61	PSW71	Количество неуспешных «сеансов связи»
PSW62	PSW72	Количество Modbus–запросов, на которые не был получен ответ за время таймаута
PSW63	PSW73	Количество Modbus–запросов, на которые был получен ответ с кодом ошибки Modbus
PSW66	PSW76	Адрес последнего slave–устройства, с которым произошла ошибка обмена

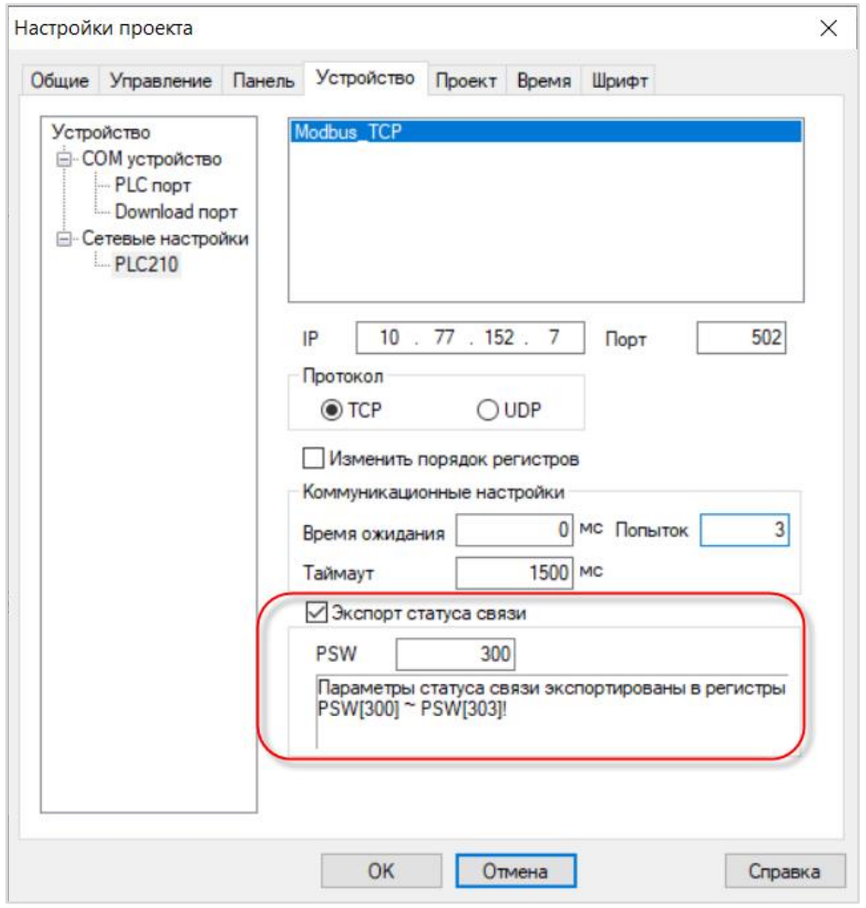
Давайте рассмотрим работу системных битов и регистров на конкретном примере – пусть slave–устройство, подключенное к Download–порту, перестало функционировать. Тогда:

- панель отправляет запрос и не получает ответ в течение таймаута (который определяется значением регистра **PFW28**) – после чего детектируется отсутствие ответа и значение регистра **PSW62** увеличивается на 1;
- панель отправляет повторный запрос. Если на X последовательных запросов (где X = значение параметра **Количество повторов,** заданного в настройках последовательного интерфейса) нет ответов, то бит **PSB50** устанавливается в **TRUE** на некоторое (зависящее от ряда факторов, в т. ч. значения таймаута) время, а значение регистра **PSW61** увеличивается на 1 («сеанс связи» не был проведён). Фактически, если за всё время работы панели связи с этим устройством не было, то **PSW61** = **PSW62 / Количество повторов**;
- в регистр **PSW66** записывается адрес slave–устройства, с которым не состоялся «сеанс связи».

После этого начинается следующий «сеанс связи»: бит **PSB50** сбрасывается в **FALSE** и происходит отправка следующего запроса.



Для режима **Modbus TCP Master** – соответствующие 4 счётчика запросов (аналоги **PSW60...63** и **PSW70...73**) могут быть экспортированы в выбранные локальные регистры панели с помощью установки галочки **Экспорт статуса связи** в настройках Modbus TCP Slave–устройства.



Все счётчики имеют диапазон 0...65535 и подвержены переполнению.

Системные регистры для изменения настроек COM–портов

В редких конкретных случаях требуется обеспечить возможность изменения настроек COM–порта с экрана панели оператора.

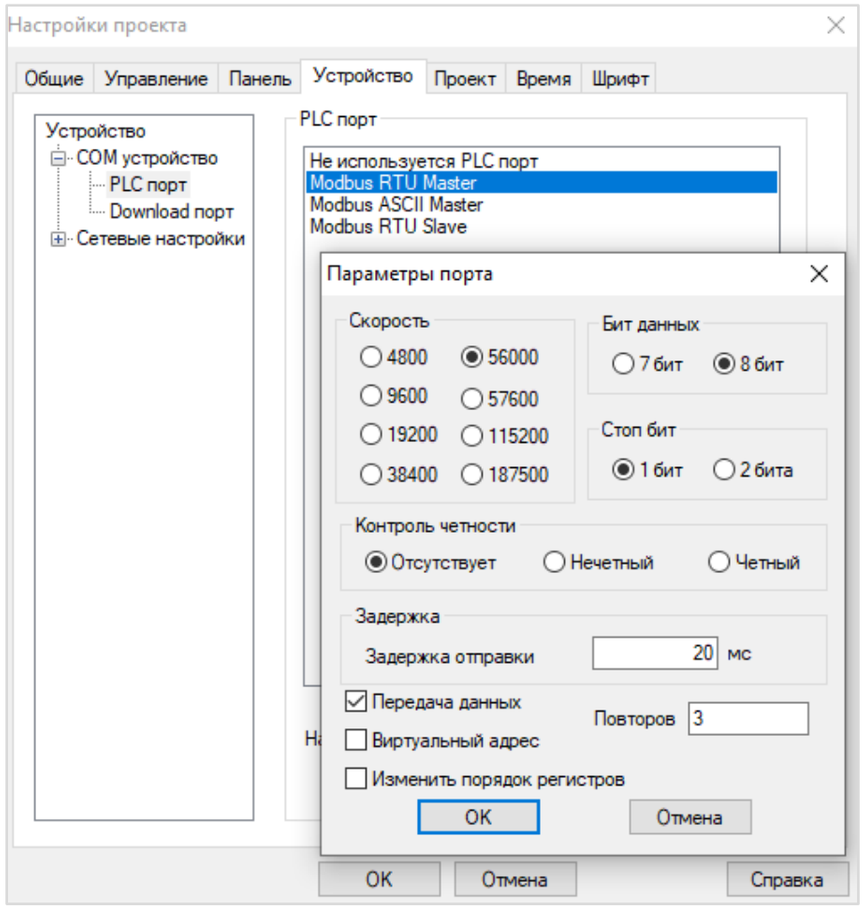
Для этого используются соответствующие системные регистры:

Download–порт	**PLC–порт**	**Описание**
PFW28	PFW38	Таймаут ожидания ответа, в миллисекундах
PFW20	PFW30	Скорость обмена. Возможные значения: **48** – 4800 бит/с **96** – 9600 бит/с **192** – 19200 бит/с **384** – 38400 бит/с **576** – 57600 бит/с **1152** – 115200 – бит/с
PFW21	PFW31	Количество бит данных. Возможные значения: **0** – 8 бит **1** – 7 бит
PFW22	PFW32	Количество стоп бит. Возможные значения: **0** – 2 стоповых бита **2** – 1 стоповый бит
PFW23	PFW33	Режим контроля чётности. Возможные значения:

		0 – отсутствует (NONE) **1** – проверка на нечётность (ODD) **2** – проверка на чётность (EVEN)
PFW24	PFW34	Адрес панели в режиме **Modbus RTU Slave**
PFW25	PFW35	Задержка отправки, в миллисекундах
PFW26.2	PFW36.2	Выбор функции Modbus, используемой для записи одного регистра (значения типа **WORD**). Не влияет на обмен через макросы. **FALSE** – 0x06 (Write Single Register) **TRUE** – 0x10 (Write Multiple Registers)

После изменения значений системных регистров требуется перезагрузить панель для применения новых настроек.

Большая часть системных регистров соответствует настройкам интерфейса, задаваемым в меню **Настройки проекта – Устройство**. Исключение составляют таймаут ожидания ответа и бит выбора функции записи – их настроить в этом меню не получится.



Системные регистры **PSW64–65** (для Download–порта) и **PSW74–75** (для PLC–порта) содержат информацию о текущем выбранном режиме обмена:

- PSW64 = 5, PSW65 = 16 – Modbus RTU Master;
- PSW64 = 22, PSW65 = 16 – Modbus ASCII Master;
- PSW64 = 23, PSW65 = 6 – Modbus RTU Slave;
- PSW64 = 0, PSW65 = 0 – интерфейс не используется.

(аналогично для регистров **PSW74–75**)

Это регистры недоступны для записи; изменить режим работы интерфейса можно только в меню **Настройки проекта – Устройство**.

Системный регистр для управления обменом

В некоторых случаях требуется в процессе работы панели останавливать/возобновлять обмен с определёнными slave–устройствами – например, в случае их вывода в ремонт.

Это можно реализовать с помощью системного регистра **PFW130**. Подробное описание механизма управления обменом с помощью этого регистра приведено в п. 9.11 руководства пользователя.

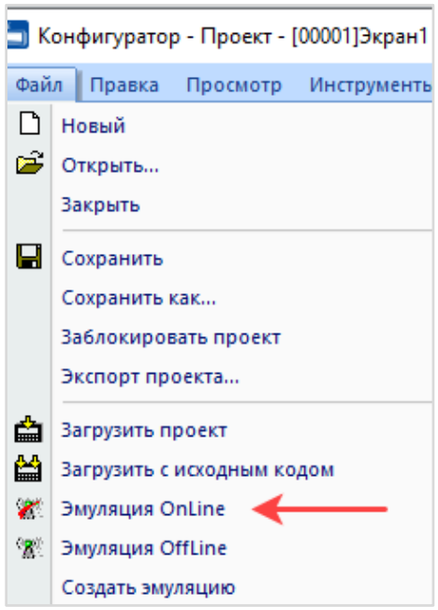
К сожалению, данный функционал имеет определённые недоработки (например – отключение обмена с одним конкретным slave–устройством может привести к остановке обмена с другими slave–устройствами), поэтому вместо него предпочтительнее перенести обмен в макрос, в рамках которого реализовать управление обменом не составляет труда; мы рассмотрели, как это сделать, в уроке 4.3.

4.6 Проверка обмена в режиме эмуляции

В случае отсутствия у вас реальной панели – вы можете проверить работу проекта, запустив его на вашем ПК в режиме онлайн–эмуляции. Период непрерывной работы эмулятора ограничен 30 минутами, после чего его можно перезапустить для продолжения использования.

В режиме онлайн–эмуляции (в отличие от оффлайн–эмуляции) можно проверить часть функционала, связанного с обменом по Modbus.

Для запуска онлайн–эмуляции нужно использовать команду **Файл – Эмуляция OnLine**.



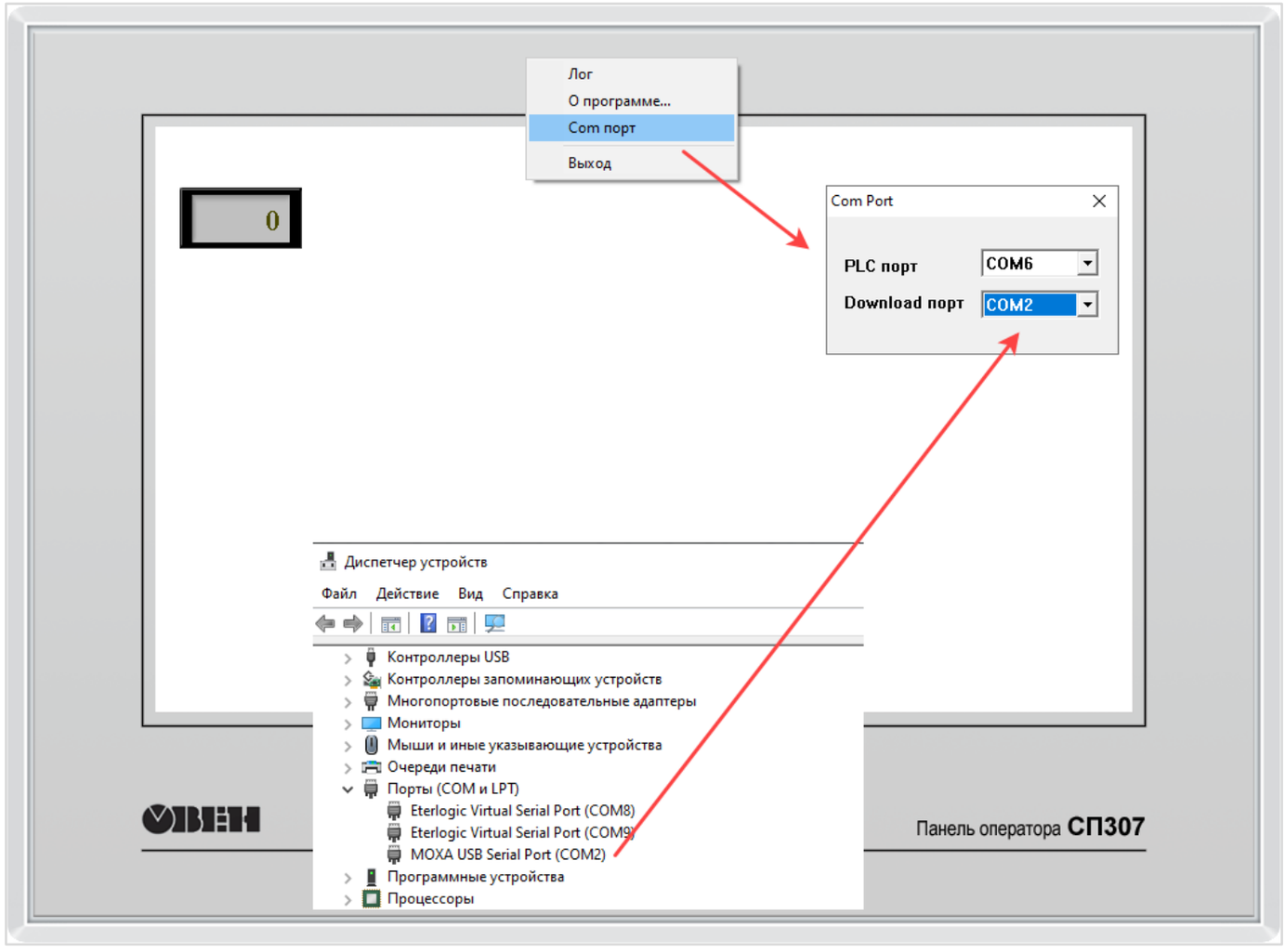
Онлайн–эмуляция имеет ряд ограничений:

- поддерживаются только режимы **Modbus RTU Master** и **Modbus TCP Master**.
  Режимы **Modbus ASCII Master**, **Modbus RTU Slave** и **Modbus TCP Slave** не поддерживаются;
- не поддерживаются макросы;
- в режиме **Master** не поддерживается доступ к отдельным битам регистров (в стиле **4x0.0**);
- в режиме **Slave** не поддерживается доступ к регистрам **PFW** с адресом **4096** и выше;
- не поддерживаются системные регистры для изменения настроек COM–портов и управления обменом.

Эмулятор панели может опрашивать настоящие slave–устройства (модули ввода–вывода и т. д.).

В случае использования протокола Modbus TCP – эти slave–устройства должны находиться в одной локальной сети с ПК, на котором запускается онлайн–эмуляция. IP–адреса этих устройств должны быть прописаны в меню **Настройки проекта – Управление**. Сетевые настройки, заданные для самой панели, в режиме онлайн–эмуляции не имеют значения.

В случае использования протокола Modbus RTU – эти slave–устройства должны быть подключены к ПК через преобразователь интерфейсов RS–485/USB или RS–232/USB. Для указания номеров создаваемых преобразователями виртуальных COM–портов (которые вы можете посмотреть в **Диспетчере устройств Windows**) нужно после запуска эмуляции нажать правой кнопкой мыши на появившемся экране панели и выбрать пункт **Com порт**.



Slave–устройства также могут быть проэмулированы – например, с помощью программ **MasterOPC Universal Modbus Server** и **VSPE**, которые мы рассматривали в модуле 2.

https://vkvideo.ru/video–68314714_456240091

4.7 Настройка обмена по Modbus в других панелях оператора

В прошлых уроках мы рассмотрели, как настроить обмен по Modbus для панелей оператора СП3хх от компании ОВЕН.

Основные принципы будут справедливы и для большинства других панелей оператора. Продемонстрируем это на примере панелей оператора компании **Weintek**, конфигурируемых в ПО **EasyBuilder Pro**. Мы не будем детально рассматривать каждый из аспектов настройки обмена – вместо этого постараемся показать, насколько они близки к уже изученному нами материалу.

https://www.weintek.com/globalw/Default.aspx

Настройки интерфейса и выбор протокола

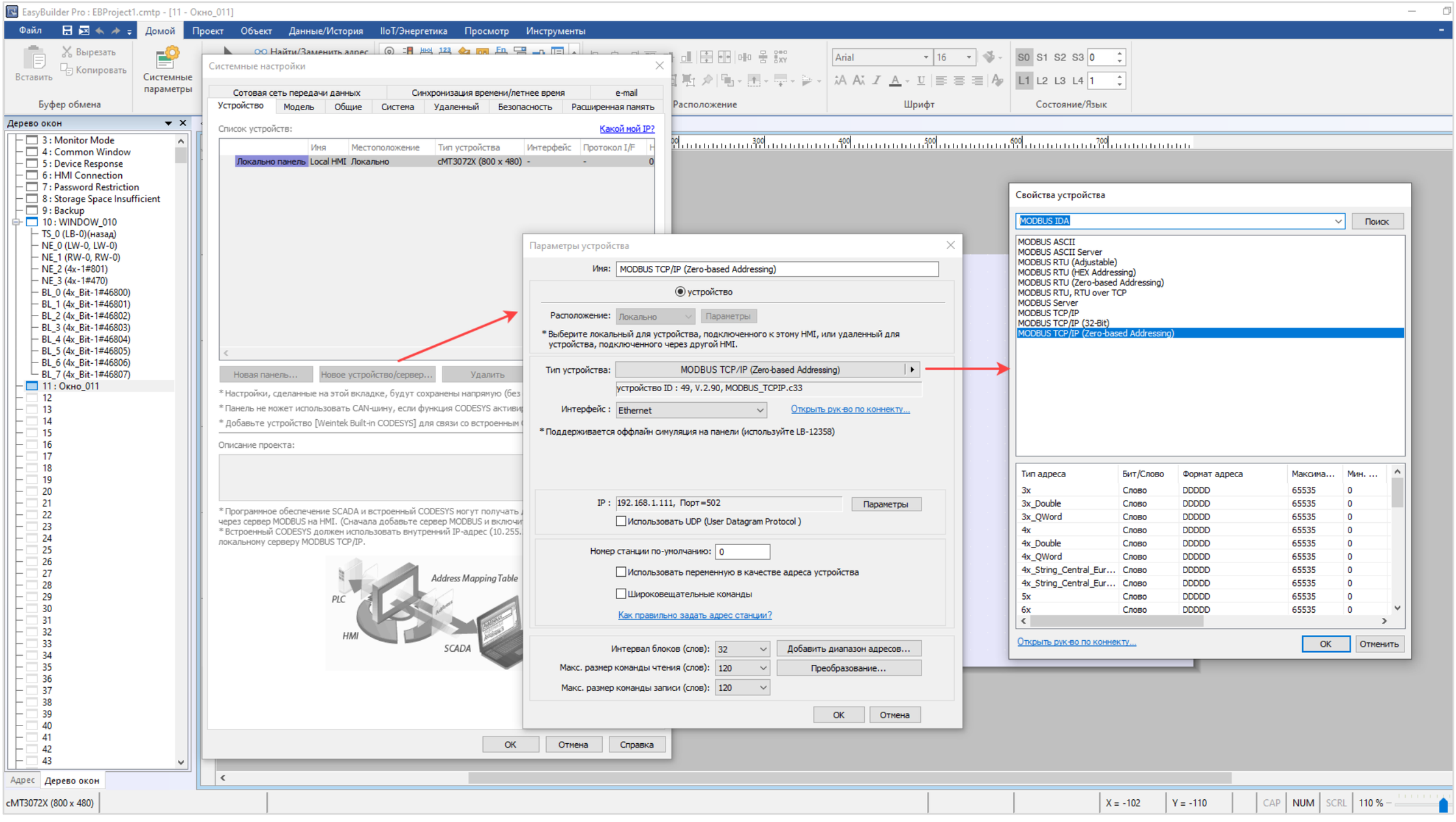
Настройка сетевых интерфейсов и выбор протокола обмена осуществляется в меню **Системные параметры – Устройство** с помощью кнопки **Новое устройство/сервер**.

В появившемся окне выбирается протокол обмена и режим работы (с помощью кнопки **Тип устройства**). В большинстве случаев интерфейс определяется автоматически на основании выбранного протокола и не может быть изменен пользователем. Исключение составляют варианты **Modbus Server** и **Modbus RTU, RTU over TCP**.

- **Modbus Server** позволяет настроить работу панели в режиме **Slave** по протоколу **Modbus RTU** или **Modbus TCP**;
- **Modbus ASCII** Server позволяет настроить работу панели в режиме **Slave** по протоколу **Modbus ASCII**.

Все остальные варианты связаны с настройкой обмена в режиме **Modbus RTU Master** и **Modbus TCP Master**. Отличия между ними незначительны и касаются, например, способа расчёта адресов регистров. Из личного опыта мы рекомендуем варианты **Modbus RTU (Zero–based Addressing)** и **Modbus TCP (Zero–based Addressing)** – в этом случае адресация регистров ведётся с **0**, что является естественным для протокола Modbus.

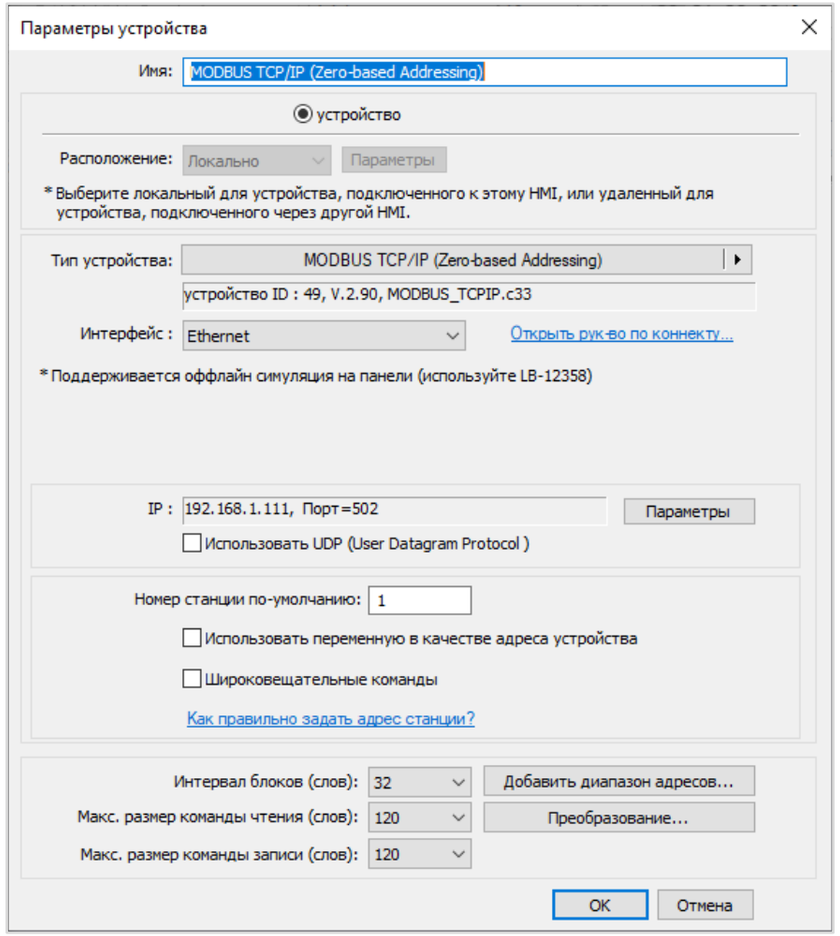
Вариант **Modbus RTU, RTU Over TCP** используется при работе с конвертерами интерфейсов Ethernet/COM. Мы поговорим о них в модуле 5.


EasyBuilder Pro : EBProject1.cmtp - [11 - Окно_011]
Файл
Домой
Проект
Объект
Данные/История
IIoT/Энергетика
Просмотр
Инструменты
Вставить
Вырезать
Копировать
Буфер обмена
Системные параметры
Найти/Заменить адрес
Расположение
Arial
16
Шрифт
Состояние/Язык
Дерево окон
3 : Monitor Mode
4 : Common Window
5 : Device Response
6 : HMI Connection
7 : Password Restriction
8 : Storage Space Insufficient
9 : Backup
10 : WINDOW_010
TS_0 (LB-0)(назад)
NE_0 (LW-0, LW-0)
NE_1 (RW-0, RW-0)
NE_2 (4x-1#801)
NE_3 (4x-1#470)
BL_0 (4x_Bit-1#46800)
BL_1 (4x_Bit-1#46801)
BL_2 (4x_Bit-1#46802)
BL_3 (4x_Bit-1#46803)
BL_4 (4x_Bit-1#46804)
BL_5 (4x_Bit-1#46805)
BL_6 (4x_Bit-1#46806)
BL_7 (4x_Bit-1#46807)
11 : Окно_011
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
Адрес
Дерево окон
Системные настройки
Сотовая сеть передачи данных
Синхронизация времени/летнее время
e-mail
Устройство
Модель
Общие
Система
Удаленный
Безопасность
Расширенная память
Список устройств:
Какой мой IP?
Имя
Местоположение
Тип устройства
Интерфейс
Протокол I/F
Локально панель
Local HMI
Локально
cMT3072X (800 x 480)
Новая панель...
Новое устройство/сервер...
Удалить
* Настройки, сделанные на этой вкладке, будут сохранены напрямую (без
* Панель не может использовать CAN-шину, если функция CODESYS активи
* Добавьте устройство [Weintek Built-in CODESYS] для связи со встроенным
Описание проекта:
* Программное обеспечение SCADA и встроенный CODESYS могут получать
через сервер MODBUS на HMI. (Сначала добавьте сервер MODBUS и включи
* Встроенный CODESYS должен использовать внутренний IP-адрес (10.255.
локальному серверу MODBUS TCP/IP.
Address Mapping Table
PLC
HMI
SCADA
OK
Отмена
Справка
Параметры устройства
Имя:
MODBUS TCP/IP (Zero-based Addressing)
устройство
Расположение:
Локально
Параметры
* Выберите локальный для устройства, подключенного к этому HMI, или удаленный для устройства, подключенного через другой HMI.
Тип устройства:
MODBUS TCP/IP (Zero-based Addressing)
устройство ID : 49, V.2.90, MODBUS_TCPIP.c33
Интерфейс :
Ethernet
Открыть рук-во по коннекту...
* Поддерживается оффлайн симуляция на панели (используйте LB-12358)
IP :
192.168.1.111, Порт=502
Параметры
Использовать UDP (User Datagram Protocol)
Номер станции по-умолчанию:
0
Использовать переменную в качестве адреса устройства
Широковещательные команды
Как правильно задать адрес станции?
Интервал блоков (слов):
32
Добавить диапазон адресов...
Макс. размер команды чтения (слов):
120
Преобразование...
Макс. размер команды записи (слов):
120
OK
Отмена
Свойства устройства
MODBUS IDA
Поиск
MODBUS ASCII
MODBUS ASCII Server
MODBUS RTU (Adjustable)
MODBUS RTU (HEX Addressing)
MODBUS RTU (Zero-based Addressing)
MODBUS RTU, RTU over TCP
MODBUS Server
MODBUS TCP/IP
MODBUS TCP/IP (32-Bit)
MODBUS TCP/IP (Zero-based Addressing)
Тип адреса
Бит/Слово
Формат адреса
Максима...
Мин. ...
3x
Слово
DDDDD
65535
0
3x_Double
3x_QWord
4x
4x_Double
4x_QWord
4x_String_Central_Eur...
5x
6x
Открыть рук-во по коннекту...
OK
Отменить
cMT3072X (800 x 480)
X = -102
Y = -110
CAP
NUM
SCRL
110 %

Настройки для режима Modbus Master

Большинство настроек нам уже знакомы:

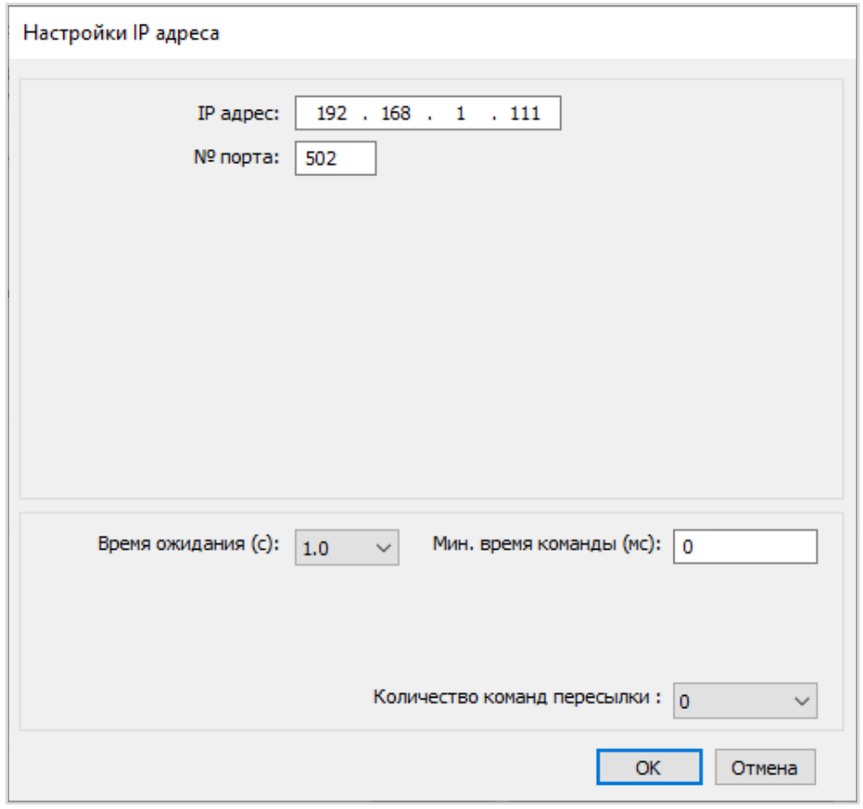
- Настройки COM–порта (для Modbus RTU) или IP–адрес и порт опрашиваемого slave–устройства (для Modbus TCP);
- галочка **Использовать UDP** позволяет использовать протокол Modbus UDP вместо Modbus TCP;
- **Номер станции по умолчанию** – адрес опрашиваемого slave–устройства. Для Modbus TCP – это Unit ID. Для Modbus RTU – это Slave ID, который используется «по умолчанию», если он не задан в явном виде в настройках элемента. Как его задать – мы увидим в следующем шаге. Т. е. для опроса нескольких slave–устройств, подключённых к последовательному интерфейсу панели – достаточно добавить одно устройство Modbus RTU;
- галочка **Использовать переменную в качестве адреса устройства** позволяет изменять описанный выше адрес через системный регистр панели;
- галочка **Широковещательные команды** позволяет указать специальный адрес. При выборе этого адреса в элементе, формирующем Modbus–запрос записи – запись будет производиться путём отправки широковещательного запроса (broadcast);
- **Интервал блоков слов** – это настройка позволяет задать максимальный «разрыв» между адресами регистров, начиная с которого они не будут автоматически объединяться в групповой запрос. В прошлых уроках мы рассматривали принцип автоформирования групповых запросов в панелях СП3хх – у Weintek он похож, но у пользователя есть возможность влиять на механизм его работы с помощью этой настройки;
- **Макс. размер команды чтения/записи** – максимальное количество регистров, которое может быть считано/записано в рамках группового запроса.



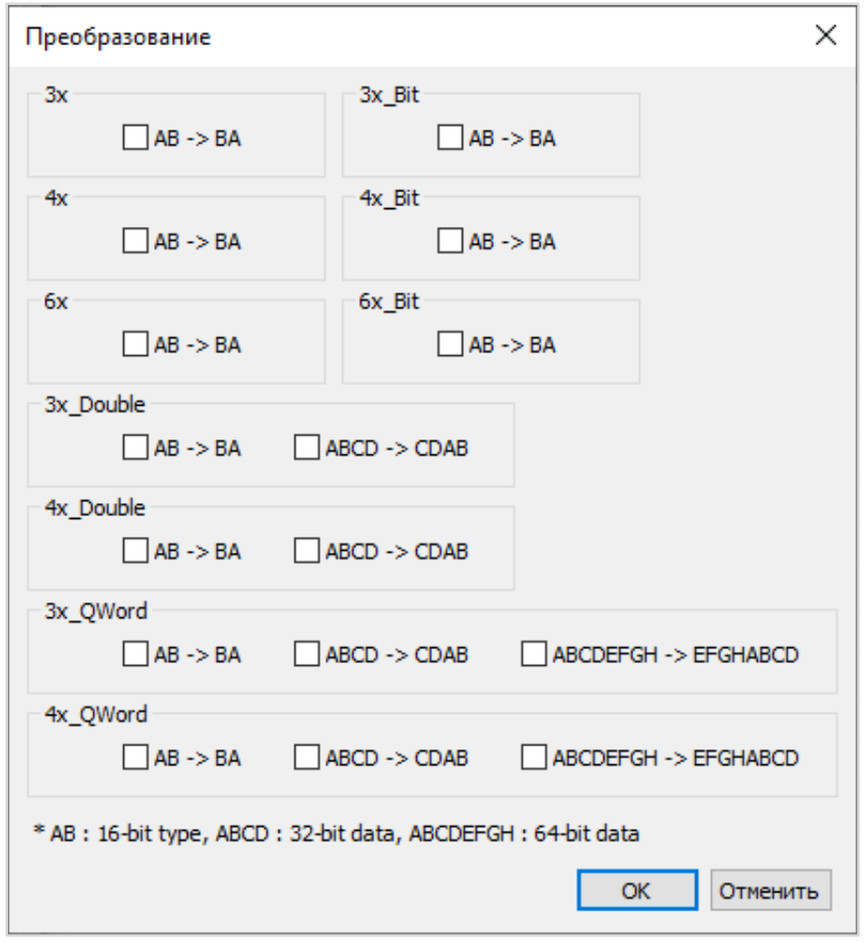
Нажатие на кнопку **Параметры** открывает следующее окно:

- **Время ожидания** – таймаут ожидания ответа от slave–устройства;
- **Количество команд пересылки** – количество повторных запросов в случае отсутствия ответа;
- **Мин. время команды –** задержка между получением ответа и отправкой следующего запроса (turnaround delay).

В случае отсутствия ответа на экране панели появится окно с информацией об отсутствии связи с данным slave–устройством.



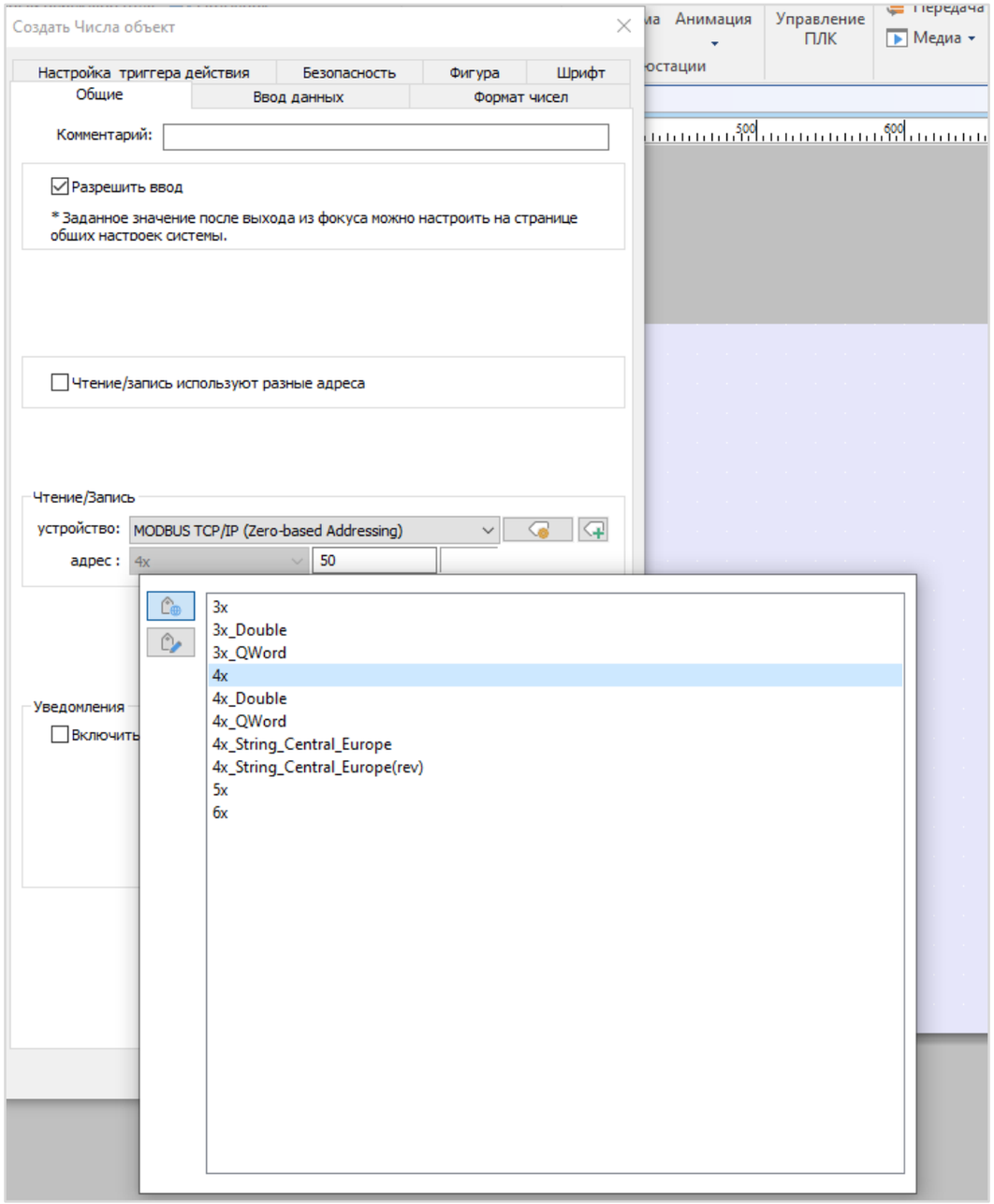
Меню **Преобразование** позволяет настроить порядок байт и регистров для различных типов данных (мы обсудим их в следующем шаге).



Меню **Добавить диапазон адресов** позволяет тонко настроить Modbus–запросы на чтение бит (coils и discrete inputs), формируемые панелью.

Настройка опроса через элементы

Настройка опроса через элементы в EasyBuilder Pro происходит примерно так же, как и в конфигураторе панелей СП3хх:



В выпадающем списке параметра **Адрес** можно увидеть множество вариантов.

3x и **4x** нам уже знакомы – это идентификаторы областей input–регистров и holding–регистров соответственно.

Все остальные пункты являются их вариациями.

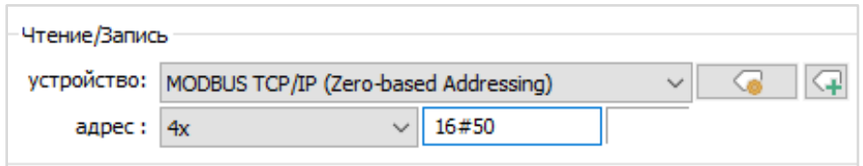
Например, для **5x** используются те же самые функции Modbus, что и для **4x**, но для параметров, занимающих 2 регистра, будет выполнена перестановка регистров.

Более подробно все возможные варианты описаны в этой статье:

https://weintek.pro/blog/tipy–modbus–adresov–v–easybuilder–pro?srsltid=AfmBOooeggKmZqAYpyIm6Aj6gNERC6n5FQbZUBA48HL835mfjYnbhvpI

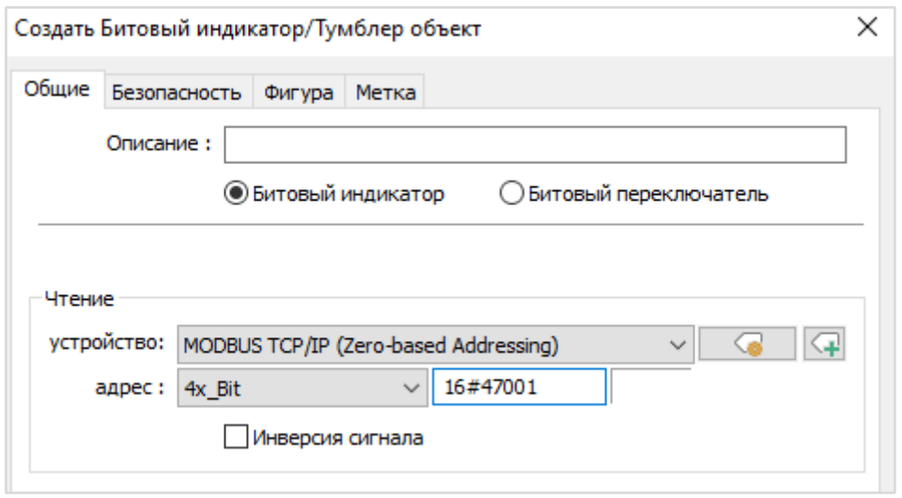
В поле ввода, расположенном справа от этого выпадающего списка, задаётся начальный адрес регистра опрашиваемого параметра. Для протоколов с постфиксом **Zero–based Addressing** вводимый адрес регистра должен совпадать с адресом регистра slave–устройства (без смещений на 1 и т. д.). Т. е. **4x50** соответствует **holding**–регистру с адресом **50**.

Обратите внимание, что в настройках элемента нет отдельного поля для ввода адреса опрашиваемого slave–устройства. По умолчанию в качестве адреса используется значение параметра **Номер станции по умолчанию**, который мы видели на предыдущем шаге. Если требуется, чтобы элемент формировал запрос к конкретному slave–устройству – то нужно сделать это следующим образом:



Значение перед символом # определяет адрес slave–устройства.

Если же вы хотите привязать к битовому элементу (индикатору или переключателю) бит регистра – то нужно сделать это так:



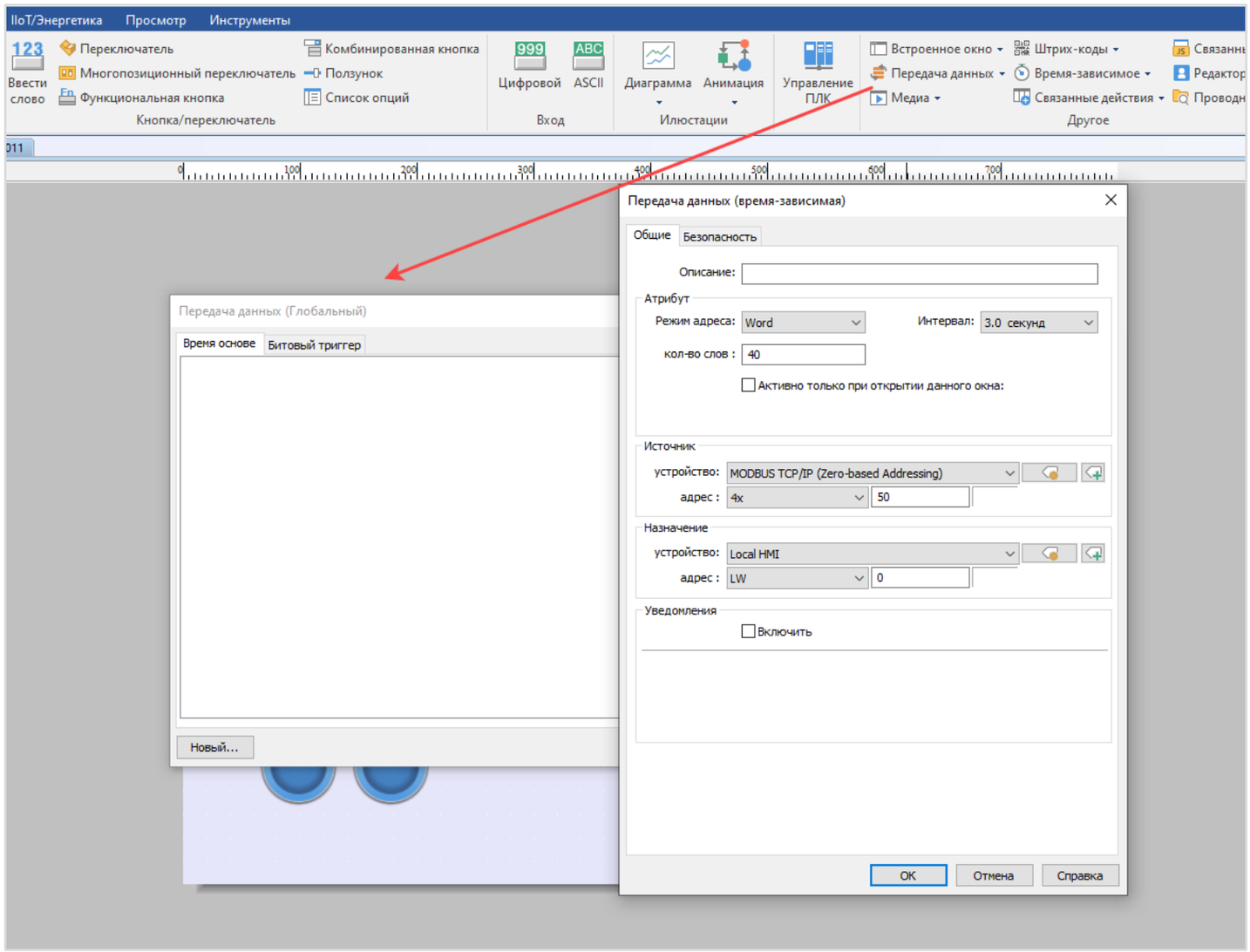
Для области **4x_bit** (и аналогичных) последние две цифры адреса соответствую номеру бита регистра с ведущим нулём.

Т.е. 47000 соответствует биту 0 регистра 470, 47001 – биту 1 регистра 470 и т. д.

Настройка опроса через «передачу данных»

Аналогом функциональной области с действием **Копировать группу регистров** в EasyBuilder Pro является невизуализируемый элемент **Передача данных**. Как и функциональная область, он может быть глобальным – то есть выполняться независимо от открытого в данный момент окна.

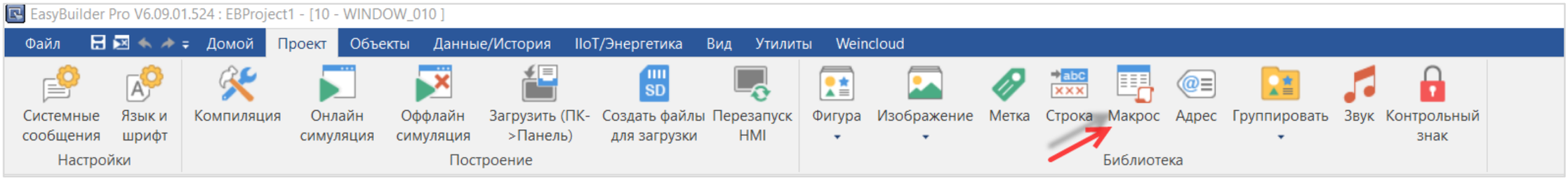
В отличие от СП3хх – в рамках этого элемента можно настроить период отправки запроса (и эта настройка в том числе будет влиять на запросы чтения), а также формировать групповой запрос для чтения/записи набора бит.



Настройка опроса через макросы

EasyBuilder Pro поддерживает написание макросов на C–подобном языке.

https://www.rusavtomatika.com/articles/spravochnik–po–makrosam–v–easybuilder–pro–na–russkom/



Для обмена по Modbus (и другим протоколам) используются встроенные функции **GetData/GetDataEx** и **SetData/SetDataEx**.

Разница между ними заключается в том, что функции **GetData** и **SetData** прекращают выполнение всего макроса в случае возникновения ошибки при чтении/записи, а функции **GetDataEx** и **SetDataEx** продолжают работу макроса несмотря на ошибку. Для определения факта возникновения ошибки может использоваться функция **GetError**.

Аргументы функций:

- переменная, содержащая записываемое значение (для **SetData/SetDataEx**) или переменная, в которой будет размещено считанное значение (для **GetData/GetDataEx**)**;**
- имя устройства, которому будет отправлен запрос (и которое настроено в меню **Системные параметры – Устройство**);
- идентификатор области памяти;
- адрес начального регистра или бита. В нём может быть закодирован адрес slave–устройства и номер бита регистра – мы рассматривали это на позапрошлом шаге;
- количество объектов. Для «одиночных» переменных одно должно быть равно 1, для массивов – определяет число используемых элементов массива. Значение этого аргумента влияет на количество бит/регистров, которое будет использовано в запросе.

Условие вызова макроса (например, период выполнения) можно настроить непосредственно в редакторе макроса. В этом случае никаких дополнительных настроек для его выполнения не требуется. Если в макросе не задано никаких условий выполнения – то требуется организовать его вызов в проекте (например, в невизуализируемом элементе **Управление ПЛК** или настройках конкретного окна).

Редактор макросов

ID макроса: 0 Имя макроса: macro_0

Безопасность

Условие выполнения

Период. вып. Интервал времени (0~864000) : 10 x 100ms

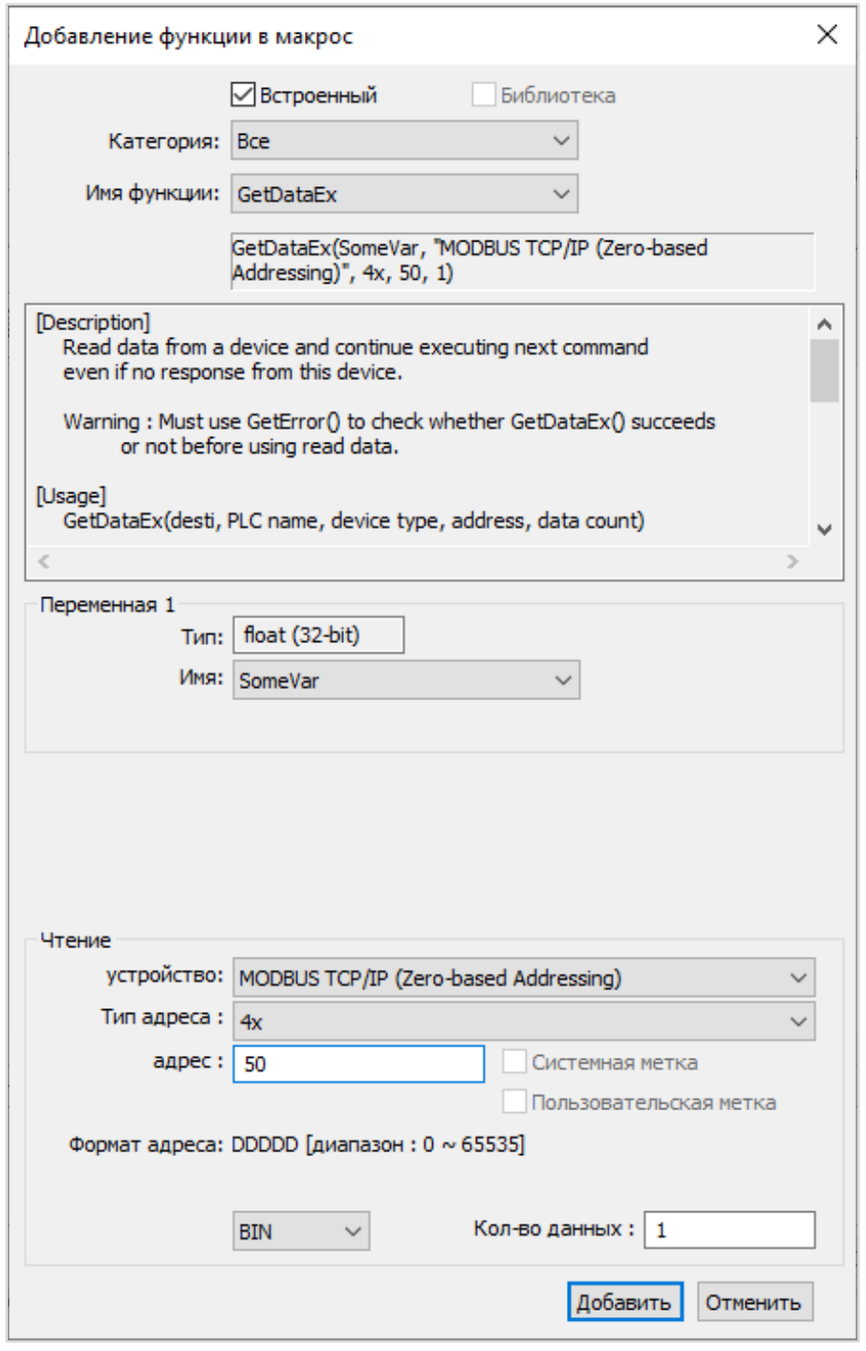
Однократно выполнить при старте панели

Защитить паролем... [Запретить]

```
1
2  macro_command main()
3
4      float SomeVar = 0
5      GetDataEx(SomeVar, "MODBUS TCP/IP (Zero-based Addressing)", 4x, 50, 1)
6
7  end macro_command
```

*Нажмите правую кнопку мыши для вызова контекстного меню редактора

Функции... Сохран. и Скомпилир. Автономное моделирование Онлайн симуляция Выход Помощь


Добавление функции в макрос
Встроенный
Библиотека
Категория:
Все
Имя функции:
GetDataEx
GetDataEx(SomeVar, "MODBUS TCP/IP (Zero-based Addressing)", 4x, 50, 1)
[Description]
Read data from a device and continue executing next command
even if no response from this device.
Warning : Must use GetError() to check whether GetDataEx() succeeds
or not before using read data.
[Usage]
GetDataEx(desti, PLC name, device type, address, data count)
Переменная 1
Тип:
float (32-bit)
Имя:
SomeVar
Чтение
устройство:
MODBUS TCP/IP (Zero-based Addressing)
Тип адреса :
4x
адрес :
50
Системная метка
Пользовательская метка
Формат адреса: DDDDD [диапазон : 0 ~ 65535]
BIN
Кол-во данных :
1
Добавить
Отменить

Настройка в режиме Modbus Slave

Панели Weintek имеют три основные области памяти:

Область памяти	**Аналог в СП3хх**	**Тип**	**Область Modbus**	**Диапазон**
LB	PSB	оперативные биты	coils/discrete inputs	0...12895
LW	PSW	оперативные регистры	input–регистры holding–регистры	0...9998
RW	PFW	энергонезависимые регистры	input–регистры holding–регистры	0...55536*

Если в меню **Системные параметры – Устройство** добавлены устройства, соответствующие работе панели в режиме Modbus Slave (т. е. **Server** или **ASCII Server**) – то эти области памяти будут доступны по Modbus.

Чтение битов области **LB** может осуществляться функциями **0x01 (Read Coils)** и **0x02 (Read Discrete Inputs)**, а запись – функциями **0x05 (Write Single Coil)** и **0x0F (Write Multiple Coils)**. Адресация бит является прямой – т. е. локальный бит **LB0** является coil'ом (и одновременно discrete input'ом) с адресом **0**.

Чтение регистров областей **LW** и **RW** может осуществляться функциями **0x03 (Read Holding Registers)** и **0x04 (Read Input Registers)**, а запись – функциями **0x06 (Write Single Register)** и **0x10 (Write Multiple Registers)**.

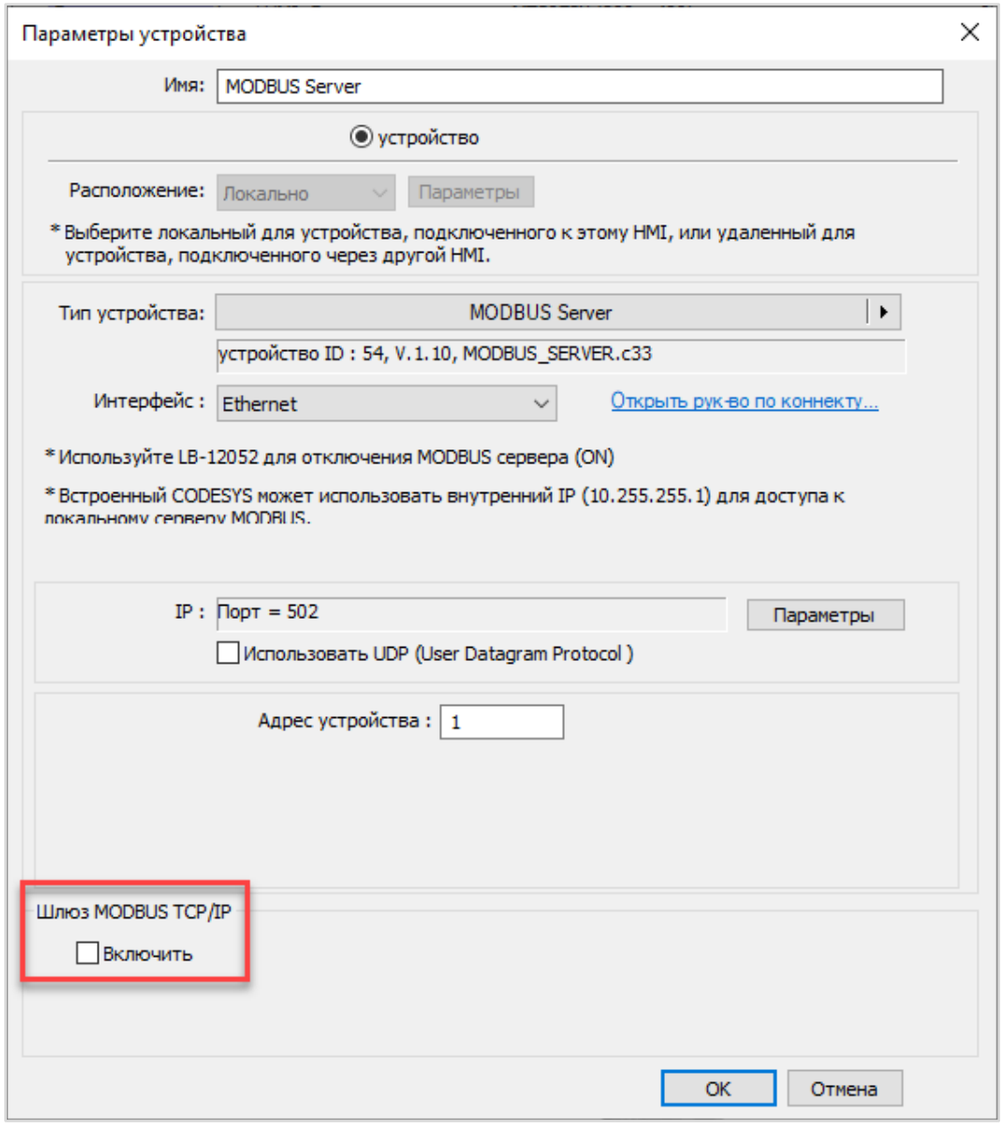
Для оперативных регистров (**LW**) адресация регистров является прямой – т. е. локальный регистр **LW0** является holding–регистром (и одновременно input–регистром) с адресом **0**.

Для энергонезависимых регистров (**RW**) адресация регистров в Modbus смещена на **+9999** – т. е. локальный регистр **RW0** является holding–регистром (и одновременно input–регистром) с адресом **9999**, а локальный регистр **RW55536** – регистром с адресом **65535**. Так как спецификация Modbus указывает, что **65535** – это максимально возможный адрес регистра, то получить доступ по Modbus к локальным регистрам панели с адресом **RW55537** и выше не получится (хотя фактически они имеются).

В режиме Modbus TCP Slave поддерживается до 64 одновременных клиентских подключений. Поле Unit ID валидируется – панель отвечает на запрос только в том случае, если в нём используется Unit ID, заданный в настройках интерфейса.

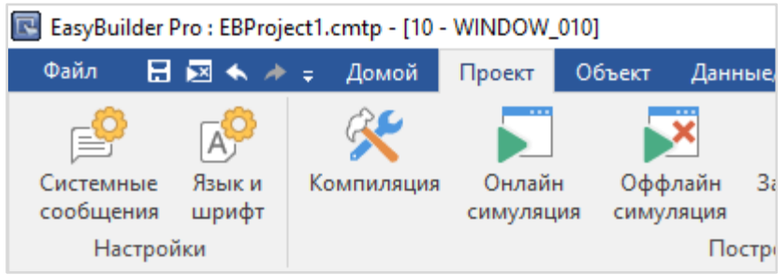
Также в режиме Slave панели Weintek поддерживают работу в качестве ретранслятора; в этом случае запросы от master–устройства пересылаются другим slave–устройствам, а ответы slave–устройств пересылаются в обратном направлении. Более подробно работа в этом режиме описана в документации:

https://www.weintekusa.com/globalw/Datasheet/download/datasheets/Weintek%20HMI%20with%20Built–In%20Modbus%20TCP%20Server.pdf


Параметры устройства
Имя:
MODBUS Server
устройство
Расположение:
Локально
Параметры
* Выберите локальный для устройства, подключенного к этому HMI, или удаленный для устройства, подключенного через другой HMI.
Тип устройства:
MODBUS Server
устройство ID : 54, V.1.10, MODBUS_SERVER.c33
Интерфейс :
Ethernet
Открыть рук-во по коннекту...
* Используйте LB-12052 для отключения MODBUS сервера (ON)
* Встроенный CODESYS может использовать внутренний IP (10.255.255.1) для доступа к
IP :
Порт = 502
Параметры
Использовать UDP (User Datagram Protocol)
Адрес устройства :
1
Шлюз MODBUS TCP/IP
Включить
OK
Отмена

Проверка обмена в режиме эмуляции

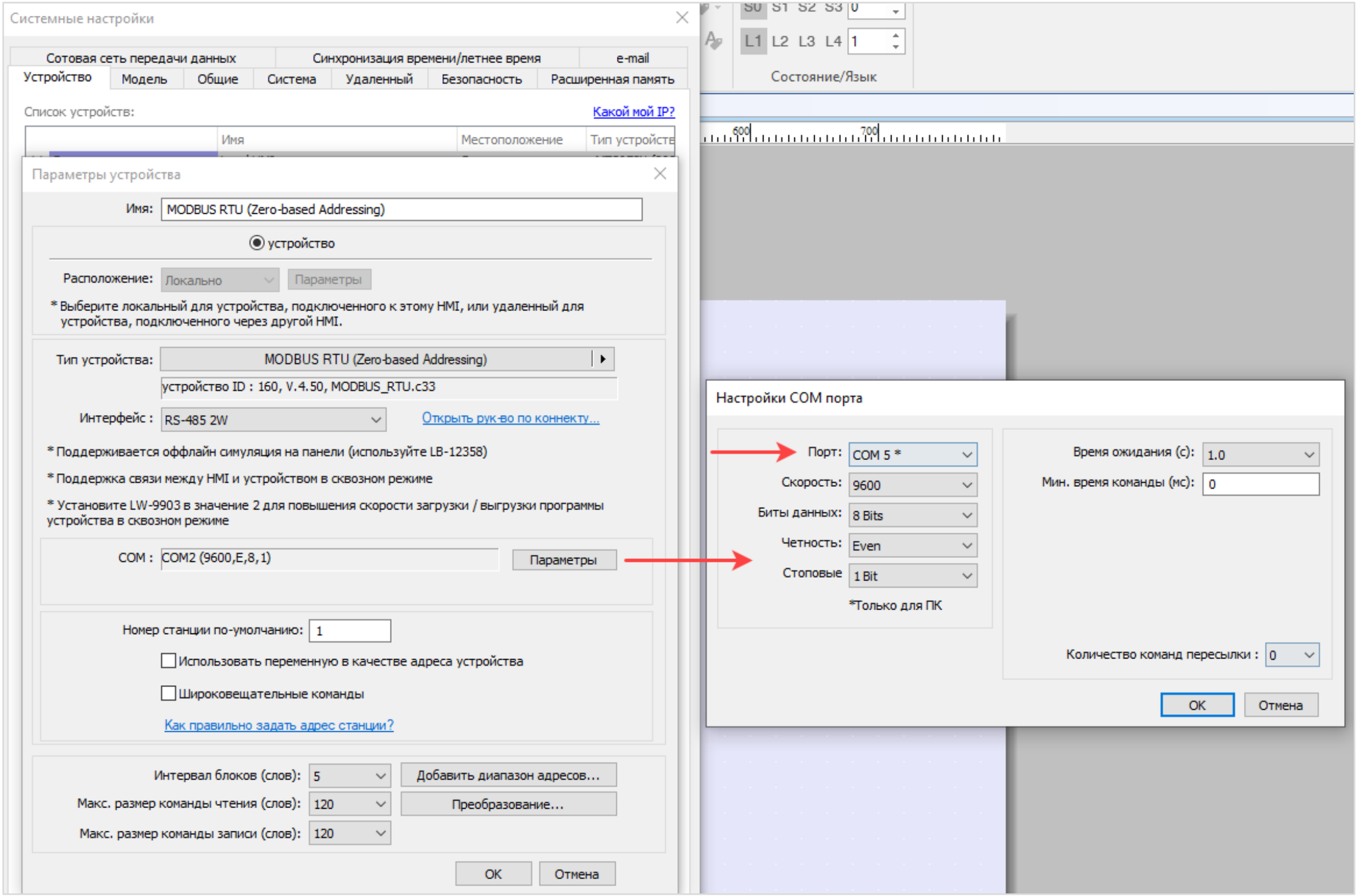
Как и Конфигуратор СП300, EasyBuilder Pro поддерживает проверку проекта в режиме онлайн–эмуляции. Период непрерывной работы эмулятора ограничен 60 минутами, после чего его можно перезапустить для продолжения использования. Для запуска эмулятора используется кнопка **Онлайн симуляция**.



С точки зрения обмена по Modbus – режим эмуляции не имеет никаких ограничений. Вы можете проверить работу обмена как в режиме Master, так и в режиме Slave. Эмулятор поддерживает исполнение макросов, побитовый доступ к регистрам и весь остальной функционал, рассмотренный в предыдущих шагах.

В случае использования протоколов Modbus RTU и ASCII – в настройках устройства должен быть выбран соответствующий виртуальный COM–порт ПК.

В случае использования режима Modbus TCP Slave – в настройках master–устройства должен быть указан действительный адрес сетевого интерфейса ПК, даже если программа–мастер запускается на том же ПК, что и EasyBuilder Pro (т. е. вариант **127.0.0.1** не подходит).


Системные настройки
Сотовая сеть передачи данных
Синхронизация времени/летнее время
e-mail
Устройство
Модель
Общие
Система
Удаленный
Безопасность
Расширенная память
Список устройств:
Какой мой IP?
Имя
Местоположение
Тип устройств
Параметры устройства
Имя: MODBUS RTU (Zero-based Addressing)
устройство
Расположение: Локально
Параметры
* Выберите локальный для устройства, подключенного к этому HMI, или удаленный для устройства, подключенного через другой HMI.
Тип устройства: MODBUS RTU (Zero-based Addressing)
устройство ID : 160, V.4.50, MODBUS_RTU.c33
Интерфейс : RS-485 2W
Открыть рук-во по коннекту...
* Поддерживается оффлайн симуляция на панели (используйте LB-12358)
* Поддержка связи между HMI и устройством в сквозном режиме
* Установите LW-9903 в значение 2 для повышения скорости загрузки / выгрузки программы устройства в сквозном режиме
COM : COM2 (9600,E,8,1)
Параметры
Номер станции по-умолчанию: 1
Использовать переменную в качестве адреса устройства
Широковещательные команды
Как правильно задать адрес станции?
Интервал блоков (слов): 5
Добавить диапазон адресов...
Макс. размер команды чтения (слов): 120
Преобразование...
Макс. размер команды записи (слов): 120
OK
Отмена
S0 S1 S2 S3 0
L1 L2 L3 L4 1
Состояние/Язык
600
700
Настройки COM порта
Порт: COM 5 *
Скорость: 9600
Биты данных: 8 Bits
Четность: Even
Стоповые 1 Bit
*Только для ПК
Время ожидания (с): 1.0
Мин. время команды (мс): 0
Количество команд пересылки : 0
OK
Отмена

Диагностика и управление обменом

Для диагностики и управления обменом используется системные биты и регистры.

См. п. 22.3.12 – 22.3.15 в документации EasyBuilder Pro:

https://www.rusavtomatika.com/upload_files/documents/weintek/EBPro/UserManual/eng/UserManual_separate_chapter/Chapter–22–System–Registers–UserManual–eng.pdf

4.8 Тестовые задания

Ответы приведены в п. 9.

1. Какие режимы работы по Modbus поддерживают панели оператора СП3xx?
   - Modbus RTU Master
   - Modbus RTU Slave
   - Modbus ASCII Master
   - Modbus TCP Slave
   - Modbus TCP Master
   - Modbus RTU over TCP Master
   - Modbus ASCII Slave

2. Какой из вариантов обмена, доступных в панелях оператора СП3xx, поддерживает настройку периода опроса как для запросов чтения, так и для запросов записи?
   - Опрос через элементы
   - Опрос через функциональную область
   - Опрос через макросы

3. Сопоставьте варианты настройки обмена, доступные в панелях оператора СП3xx, и их описания.

Опрос через элементы	Наиболее прост в настройке, не позволяет детально настраивать групповые запросы и организовать циклическую отправку запросов записи
Опрос через функциональную область	Компромиссный вариант, позволяющий ограниченно настраиваить групповые запросы и организовать циклическую отправку запросов записи
Опрос через макросы	Наиболее сложен в настройке, позволяет полностью управлять обменом, детально настраивать групповые запросы и организовать циклическую отправку запросов записи

4. Панель оператора СП3xx работает в режиме Modbus Slave. Какой адрес регистра нужно ввести в Modbus Master, чтобы обратиться к локальному регистру панели **PFW40001**?

5. Шлюзы Modbus

5.1 Основная информация

В рамках системы автоматизации могут использоваться устройства с разными интерфейсами и протоколами – и тогда возникает вопрос, как настроить между ними обмен.

Иногда это является следствием модернизации, когда к уже существующему оборудованию добавляется новое, не поддерживающее ранее используемый в системе протокол. Иногда это решение принимается на стадии проектирования – например, если подходящее по всем критериям оборудование использует интерфейс или протокол, несовместимый с другими устройствами системы.

Причины могут быть разными, но они приводят к общей задаче. Одним из возможных решений является использование специализированных устройств – шлюзов (gateway) протоколов и интерфейсов.

На рынке множество шлюзов для разных интерфейсов и протоколов – например, для преобразования CANopen в EtherCAT, Profibus в Profinet, EtherNet/IP в BACnet/IP и т. д.

Есть, конечно, и шлюзы для преобразования большинства промышленных протоколов в Modbus.

Но в рамках этого модуля мы остановимся на двух конкретных типах шлюзов:

- преобразователях интерфейсов Ethernet / COM;
- преобразователях протоколов Modbus TCP / Modbus Serial.

Сразу отметим, что некоторые шлюзы могут совмещать обе эти функции.

5.2 Преобразователи интерфейсов

Преобразователи интерфейсов Ethernet/COM. Часть 1

Давайте мысленно представим одну из вариаций системы мониторинга и управления инженерными системами здания – вентиляцией, отоплением, освещением, водоснабжанием и т. д.

В состав этой системы входят «полевые» устройства (модули ввода–вывода, преобразователи частоты и т. д.) и один или несколько компьютеров с запущенной SCADA–системой, которая производит опрос и управление этими устройствами. Поскольку в здании уже есть локальная сеть, к которой подключены эти компьютеры, то довольно удобно применить «полевые» устройства с поддержкой интерфейса Ethernet и протокола Modbus TCP, чтобы воспользоваться уже имеющейся инфраструктурой.

По мере развития системы требуется добавлять в неё новые устройства – и в какой–то момент оказывается, что у очередного набора приборов нет исполнений с интерфейсом Ethernet, а есть только с RS–485. При этом каждое устройство поддерживает свой протокол обмена – у одного это Modbus RTU, у другого – DCON, у третьего – специфический протокол, разработанный производителем данного прибора. Конечно, можно попробовать найти аналоги этих устройств с поддержкой Ethernet и Modbus TCP, но давайте предположим, что такой возможности нет – оборудование уже закуплено или имеющиеся аналоги не подходят по функциональным характеристикам.

Предположим, что SCADA–система поддерживает все упомянутые протоколы. Можно попробовать использовать преобразователи RS–485/USB – но тогда придётся прокладывать через здание кабель для связи между устройствами и компьютером; довольно часто это является неприемлемым решением.

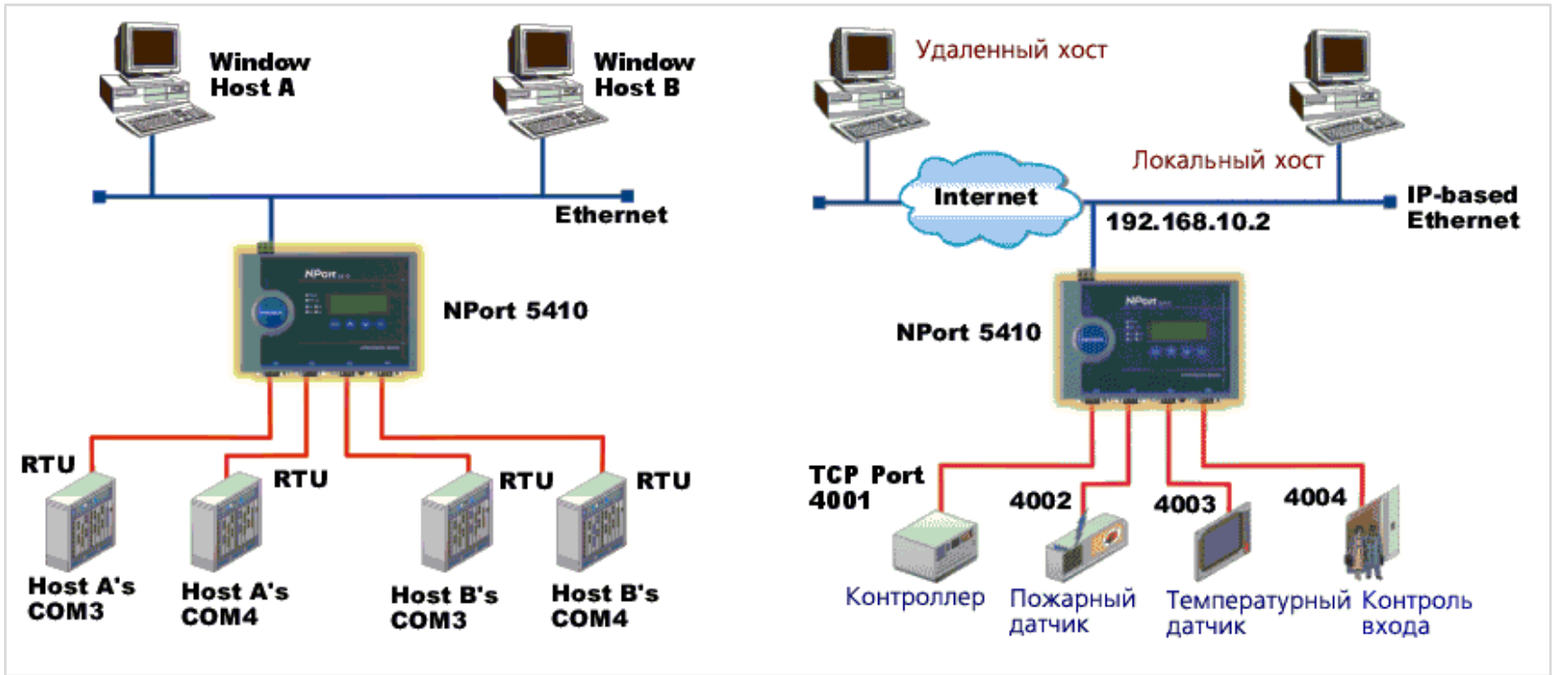
В этой ситуации можно воспользоваться преобразователем интерфейсов – устройством, которое имеет на борту интерфейсы Ethernet и RS–485 (или, например, RS–232) и обеспечивает передачу данных между ними без какого–либо преобразования. По этой причине такие устройства иногда называют «прозрачными шлюзами».

В качестве примеров подобных устройств можно назвать серию преобразователей NPort от компании Моха, линейку шлюзов tDS от компании ICP DAS, устройства компании USR IOT и т. д.

Преобразователи интерфейсов Ethernet/COM. Часть 2

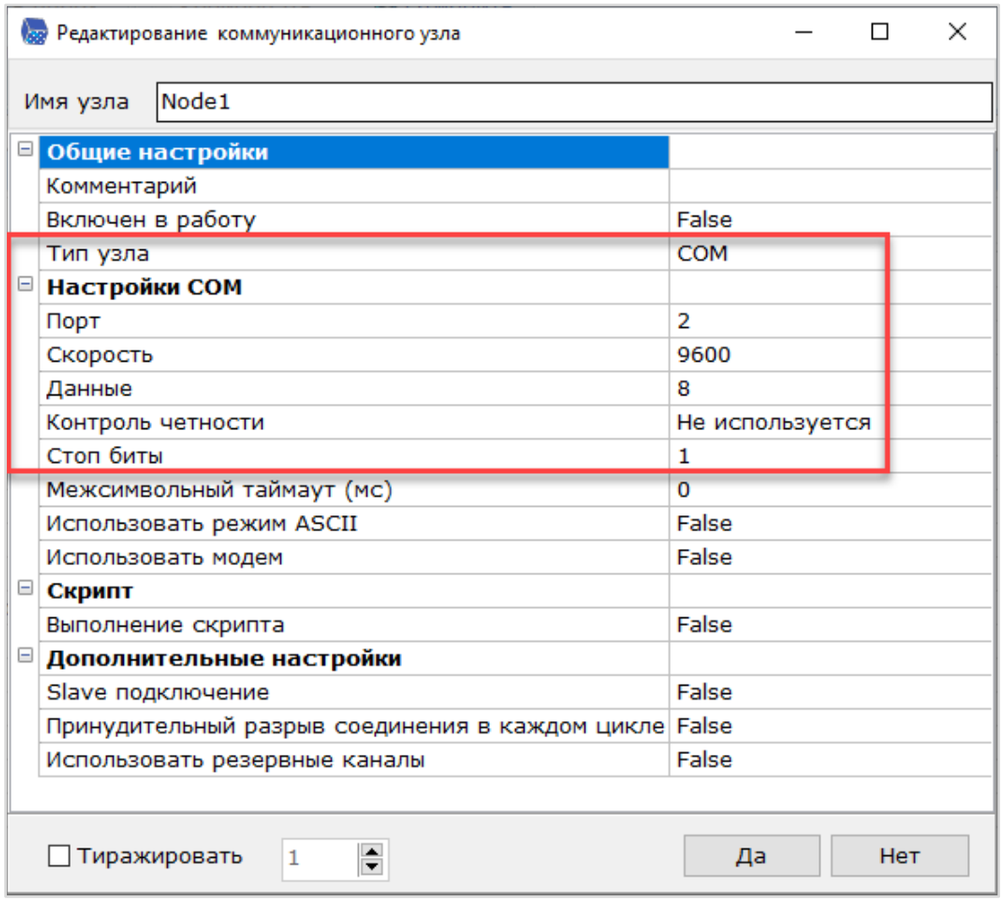
На стороне ПК со SCADA–системой есть два основных варианта опроса устройств, подключённых к преобразователю протоколов:

- через виртуальный COM–порт. SCADA–система открывает и использует этот COM–порт так, как будто он является физическим портом компьютера, а специальное промежуточное ПО ретранслирует все поступающие в этот виртуальный COM–порт запросы на заданный IP–адрес и сетевой порт преобразователя; поступившие ответы отправляются обратно в этот виртуальный порт. В качестве промежуточного ПО может использоваться драйвер, который предоставляет производитель шлюза (и тогда, вероятно, он поддерживает только работу с этим конкретным шлюзом; надо отметить, что не все производители предоставляют такие драйверы), либо универсальная программа – например, упомянутые в уроке 2.17 VSPE, Tibbo Device Server Toolkit и socat;
- через TCP–соединение (или, реже, UDP–пакеты). SCADA–система устанавливает соединение со шлюзом по заданному IP–адресу и номеру порта и отправляет через него запросы в формате протокола slave–устройства.

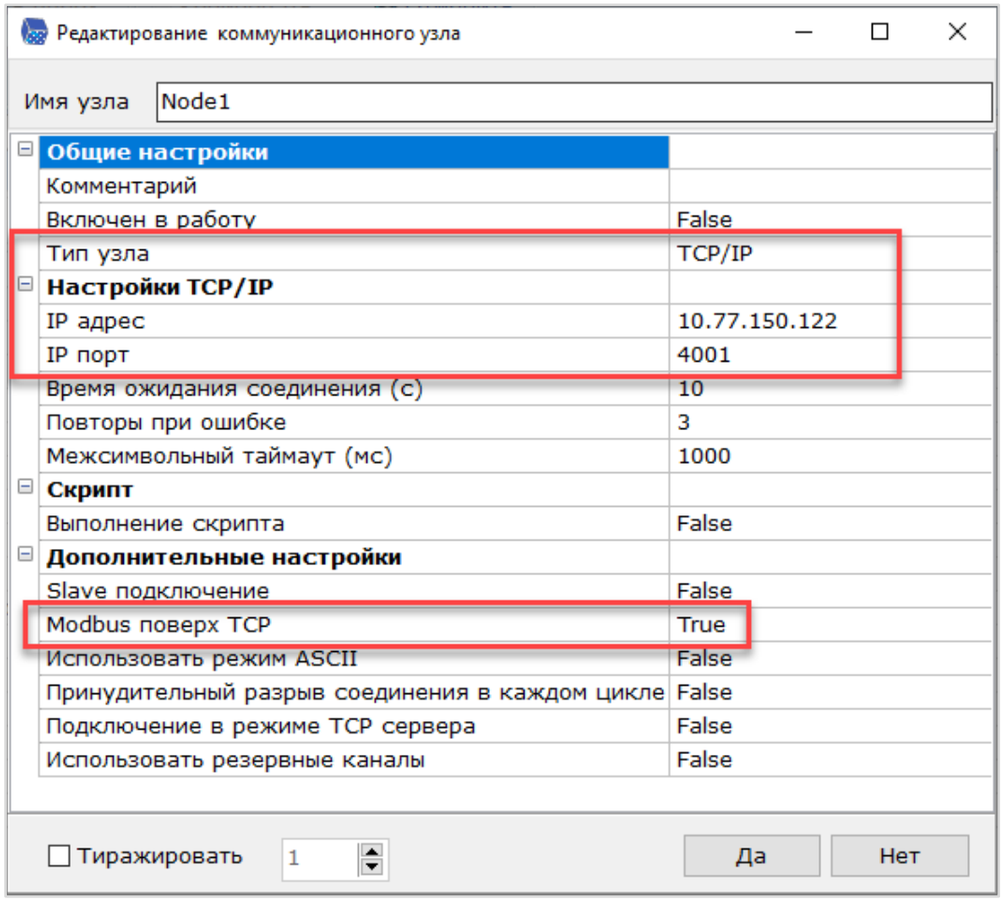


Основным отличием этих двух вариантов является интерфейс, выбираемый в настройках SCADA–системы. На примере уже неоднократно использованного нами **MasterOPC Universal Modbus Server**:

- для варианта 1 (с виртуальным COM–портом) настройка узла будет выглядеть следующим образом:



- для варианта 2 (с TCP–соединением) настройка узла будет выглядеть так:



Установка параметра **Modbus поверх TCP** в значение **TRUE** приводит к тому, что OPC–сервер будет отправлять не пакеты протокола **Modbus TCP**, а пакеты протокола **Modbus RTU**. Такой режим обмена обычно называют **Modbus RTU over TCP**.

Если slave–устройство использует протокол **Modbus ASCII** – то дополнительно требуется установить параметру **Использовать режим ASCII** значение **TRUE**. Впрочем, режим **Modbus ASCII over TCP** используется крайне редко.

Преобразователи интерфейсов Ethernet/COM. Часть 3

В прошлых шагах мы рассмотрели сценарий, в рамках которого SCADA–система (или OPC–сервер) выполняет функцию Modbus Master, используя протокол Modbus RTU over TCP. В данном случае master–устройство подключено к Ethernet–интерфейсу шлюза.

В более редких и специфических ситуациях master–устройство подключается к интерфейсу RS–485 или RS–232, а Ethernet используется для подключения slave–устройств.

***

Преобразователи интерфейсов могут иметь несколько COM–портов. В этом случае при использовании драйвера виртуального порта для каждого COM–порта конвертера на ПК создаётся собственный виртуальный COM–порт. В случае использования TCP–соединения – обычно для каждого COM–порта выделяется собственный TCP–порт.

***

Когда преобразователь получает TCP– или UDP–пакет, то пересылает его данные (payload) в последовательный интерфейс; так как такой пакет является цельным сообщением, то преобразователь начинает его передачу в COM–порт по факту его получения. При этом заголовок TCP/UDP–пакета и другая его служебная информация в COM–порт не отправляются – устройства на последовательной линии связи всё равно никак не смогли бы ей воспользоваться.

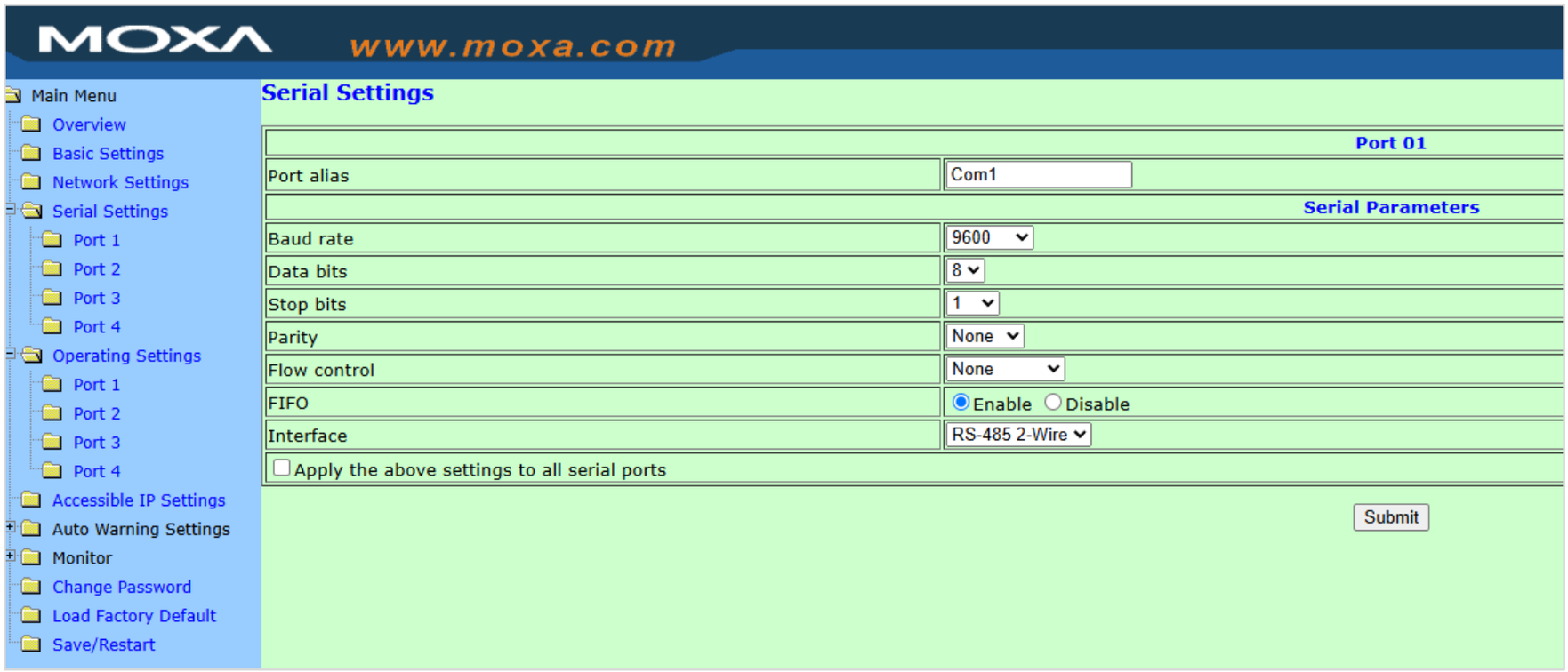
Данные, полученные по последовательному интерфейсу, преобразователь должен отправить в рамках TCP–соединения или UDP–пакета. Но эти данные передаются в виде последовательности байтов, а так как преобразователь не знает об используемом протоколе обмена – то у него нет возможности определить конец пакета (по его размеру, стоп–символу или иным способом). По этой причине в большинстве преобразователей присутствует настройка, которая устанавливает таймаут конца пакета. Если в течение заданного времени в COM–порт преобразователя не поступило новых байтов – то все накопленные в буфере байты отправляются по TCP или UDP.

***

Настройка преобразователя заключается в выборе режима его работы и настройке интерфейсов. Она выполняется через утилиту, предоставляемую производителем, или web–интерфейс (если он имеется).

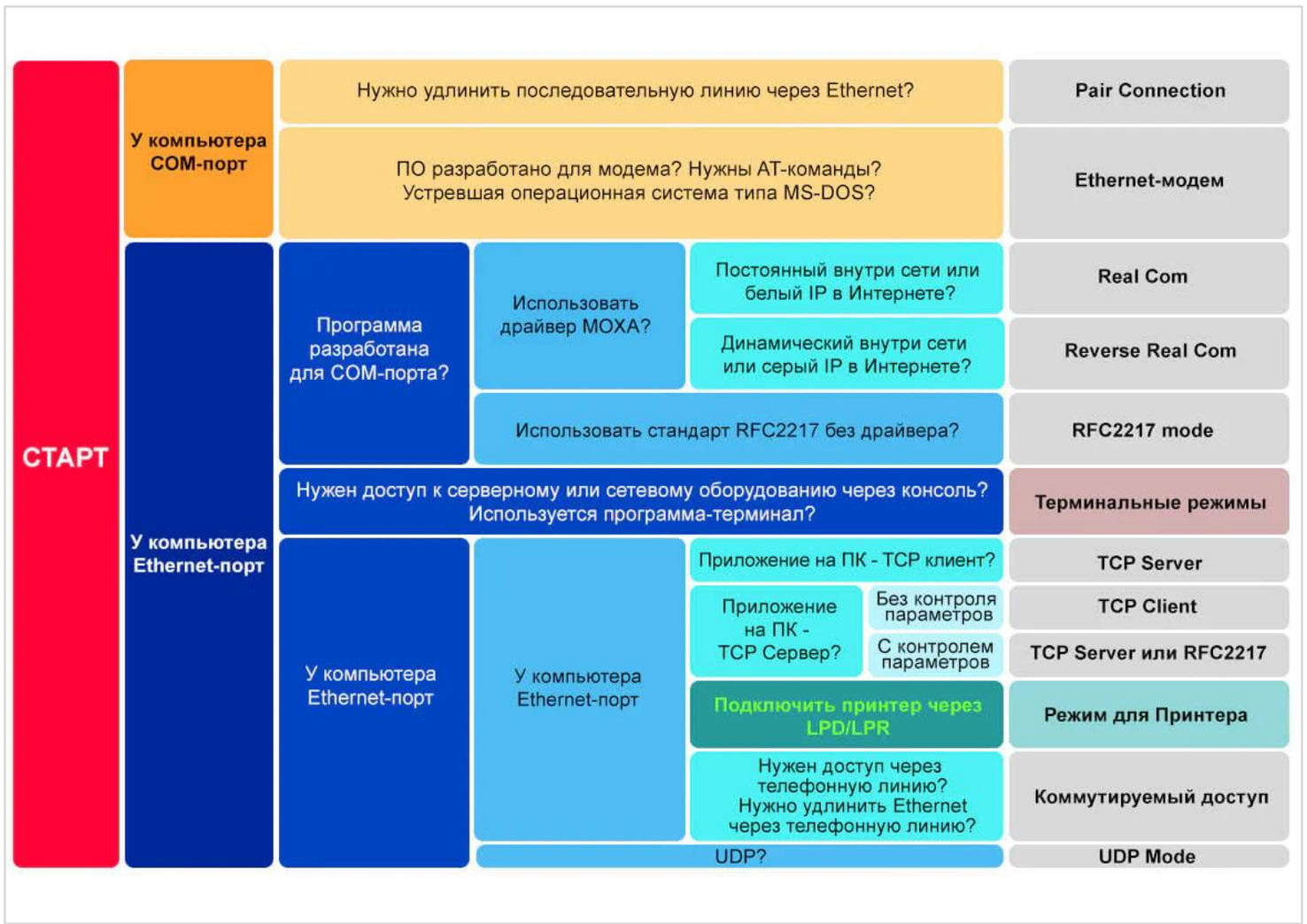
Ниже приведён скриншот web–интерфейса конвертера **Moxa NPort**. Его основными вкладками являются:

- **Network settings** (настройки интерфейса Ethernet);
- **Serial settings** (настройки COM–портов);
- **Operating settings** (режимы работы для каждого из COM–портов).



***

Продвинутые преобразователи интерфейсов могут поддерживать разные режимы работы и иметь множество настроек. На примере упомянутого Моха NPort:



Информация о них приведена в документации на преобразователь и данной статье:

https://moxa.ru.com/blogs/training/nport–land

***

Помимо режимов работы важной характеристикой преобразователя интерфейсов является количество одновременных TCP–подключений. Некоторые преобразователи поддерживают только одно подключение, некоторые – несколько. Это имеет значение, если вам требуется обеспечить одновременный опрос slave–устройств, подключённых к COM–портам преобразователя, с помощью нескольких master–устройств (например, с помощью ПЛК и SCADA–системы); поскольку COM–порт является последовательным интерфейсом связи, то преобразователь должен «распределять по времени» запросы, поступающие от master–устройств. Обычно это реализуется с помощью размещения поступающих запросов в очереди (и очевидно, что эта очередь должна иметь некоторый конечный размер).

Информация о количестве одновременных TCP–подключений и принципе их обработки должна быть приведена документации на преобразователь.

5.3 Преобразователи протоколов Modbus TCP/Modbus Serial

Преобразователи протоколов Modbus TCP/Modbus Serial. Часть 1

В прошлом уроке мы рассмотрели сценарий, в рамках которого:

- требуется подключить к SCADA–системе устройства с разными протоколами;
- SCADA–система поддерживает использование этих протоколов через TCP–соединение.

Теперь представим другую задачу.

Пусть в системе используется ПЛК, у которого уже задействованы все COM–порты – но возникает необходимость подключения ещё нескольких slave–устройств. Причём подключить их к уже имеющимся последовательным линиям связи не получается – например, они используют жёстко заданные настройки COM–порта без возможности изменения (предположим, 19200–8E1), которые не согласуются с настройками других устройств шины. Или эти устройства размещены на расстоянии от ПЛК, и проложить новую линию связи до них будет затруднительно – но, как и в прошлом примере, на объекте имеется локальная сеть.

Кажется, что можно было бы повторить решение из предыдущего урока – воспользоваться преобразователем интерфейсов. Но дело в том, что значительная часть ПЛК и других устройств (например, панелей оператора) не поддерживает режим **Modbus RTU over TCP**; они способны работать только по протоколу **Modbus TCP**.

Соответственно, от преобразователя теперь требуется не просто обеспечить передачу данных из одного интерфейса в другой, но и при этом осуществить преобразование пакетов протокола **Modbus TCP** в **Modbus RTU** (или **Modbus ASCII**) и в обратном направлении. Некоторые устройства могут совмещать эти функции и, в зависимости от настроек, работать как в режиме преобразователя протоколов, так и в режиме прозрачного шлюза.

В качестве примеров преобразователей протоколов Modbus TCP/Modbus Serial можно назвать серию преобразователей MGate MB от компании Моха, линейку шлюзов tGW от компании ICP DAS, устройства компании USR IOT и преобразователь МКОН от компании ОВЕН.

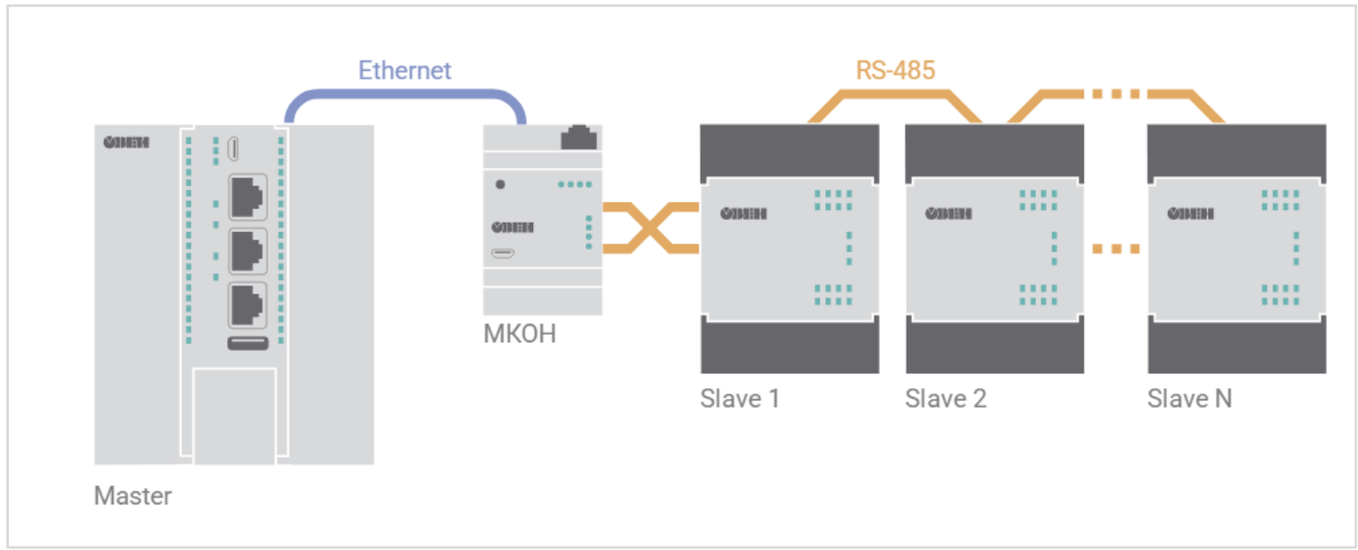
Преобразователи протоколов Modbus TCP/Modbus Serial. Часть 2

Типичный сценарий использования преобразователя Modbus TCP/Modbus Serial выглядит следующим образом:

Master–устройство (ПЛК, панель оператора, ПК со SCADA–системой и т. п.) использует протокол Modbus TCP и отправляет запросы преобразователю. С его точки зрения преобразователь является Modbus TCP Slave–устройством.

Преобразователь получает запрос и преобразует его в формат протокола Modbus RTU (или Modbus ASCII), при этом поле Unit ID из пакета Modbus TCP становится полем Slave ID пакета Modbus RTU/ASCII. Именно для такого использования поле Unit ID и было сохранено в спецификации Modbus TCP.

Преобразованный пакет отправляется в COM–порт. Ответ от slave–устройства преобразуется в пакет Modbus TCP и отправляется master–устройству.



Некоторые преобразователи предоставляют пользователю возможность тонкого конфигурирования – например, не «прямое» преобразование Unit ID в Slave ID, а на основании заданной таблицы маршрутизации (в стиле «Unit ID 1 —> Slave ID 12», «Unit ID 2 —> Slave ID 36» и т. д.).

Многопортовые преобразователи могут позволить указать, в какой именно COM–порт должен быть отправлен запрос с конкретным Unit ID. Или, например, для работы с конкретным COM–портом может использоваться конкретный TCP–порт преобразователя.

Как и в случае преобразователей интерфейса – важной характеристикой преобразователей протокола является количество одновременных TCP–подключений и принцип «распараллеливания» доступа к COM–порту для нескольких master–устройств.

Разумно спроектированное master–устройство должно учитывать сценарий работы со шлюзом. В этом случае требуется установить одно TCP–соединение, в рамках которого будет производиться отправка запросов, содержащих разные Unit ID.

Например, в среде CODESYS V3.5 к компоненту Modbus TCP Slave можно добавить «дочерние» устройства, олицетворяющие slave–устройства, подключённые к COM–порту преобразователя. В компоненте Modbus TCP Slave задаётся IP–адрес и TCP–порт шлюза, а в дочерних компонентах – адреса slave–устройств, которые будут использованы в качестве Unit ID при формировании запросов.

Файл Правка Вид Проект Компиляция Онлайн Отладка Инструменты Окно Справка

Application [Device: Plc Logic]

Устройства

- Без имени44
  - Device (PLC210)
    - Plc Logic
      - **Application**
        - TargetVars
        - ImagePool
        - Менеджер библиотек
        - PLC_PRG (PRG)
        - Конфигурация задач
        - Менеджер визуализации
        - Visualization
    - Ethernet (Ethernet)
      - Modbus_TCP_Master (Modbus TCP Master)
        - Modbus_TCP_Slave (Modbus TCP Slave)
          - MU110_32R (MU110-32R)
          - MV110_8A (MV110-8A)
          - PChV3_M01 (PChV3_M01)
          - TRM202 (TRM202)
          - SomeDevice (Modbus Slave, COM Port)
    - PLC210_11 (PLC210-11)
    - OwenRTC (OwenRTC)
    - OwenCloud (OwenCloud)
    - Buzzer (Buzzer)
    - Drives (Drives)
    - Debug (Debug)
    - Info (Info)
    - Watchdog (Watchdog)

Modbus_TCP_Slave

Общее
Канал Modbus Slave
Modbus Slave Init
ModbusTCPSlave Конфигурация
ModbusTCPSlave МЭК-объектов
Состояние
Информация

Modbus TCP

IP-адрес слейва: 10 . 77 . 150 . 20
Таймаут ответа (мс): 1000
Порт: 502

SomeDevice

Общее
Канал Modbus Slave
Modbus Slave Init
ModbusGenericSerialSlave МЭК-объектов
Состояние
Информация

Modbus-RTU/ASCII

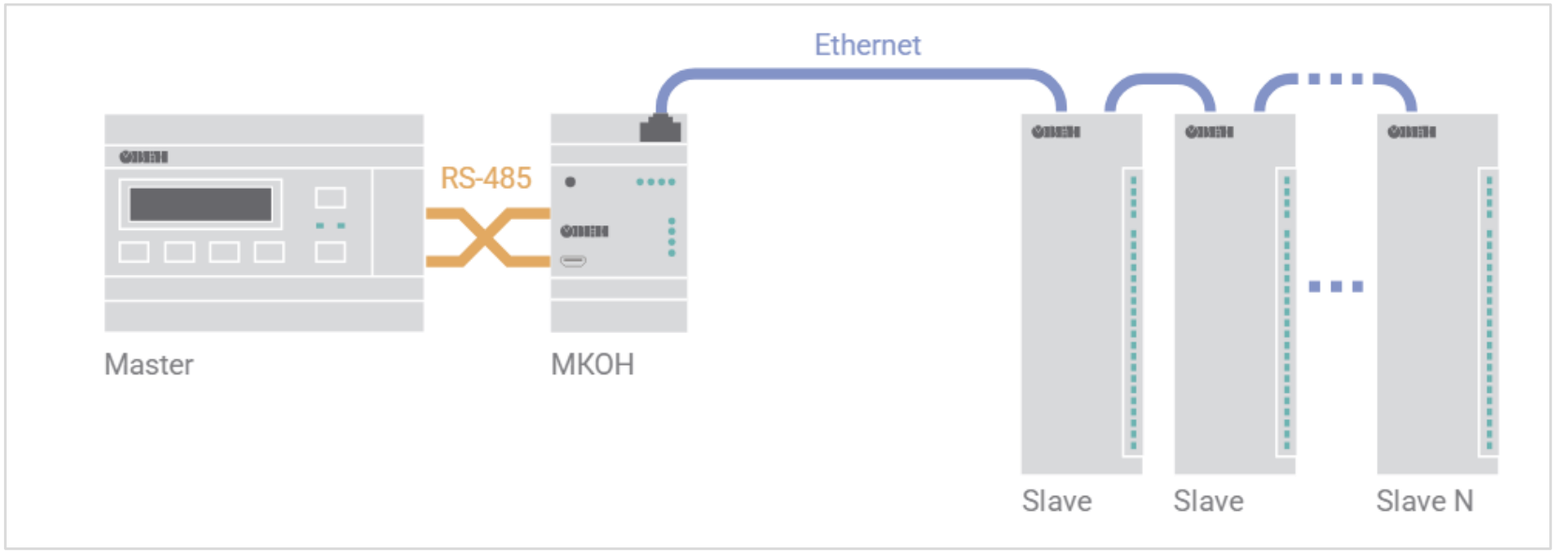
MODBUS

Адрес слейва [1..247]: 1
Таймаут ответа (мс): 1000

Преобразователи протоколов Modbus TCP/Modbus Serial. Часть 3

В существенно более редких случаях требуется обратный вариант работы преобразователя – когда master–устройство использует протокол Modbus RTU (или Modbus ASCII), а slave–устройства поддерживают протокол Modbus TCP.

Например, такая ситуация может возникнуть при модернизации системы управления, в которую требуется добавить ещё некоторое количество slave–устройств, но найти подходящие приборы с наличием COM–порта не получается.



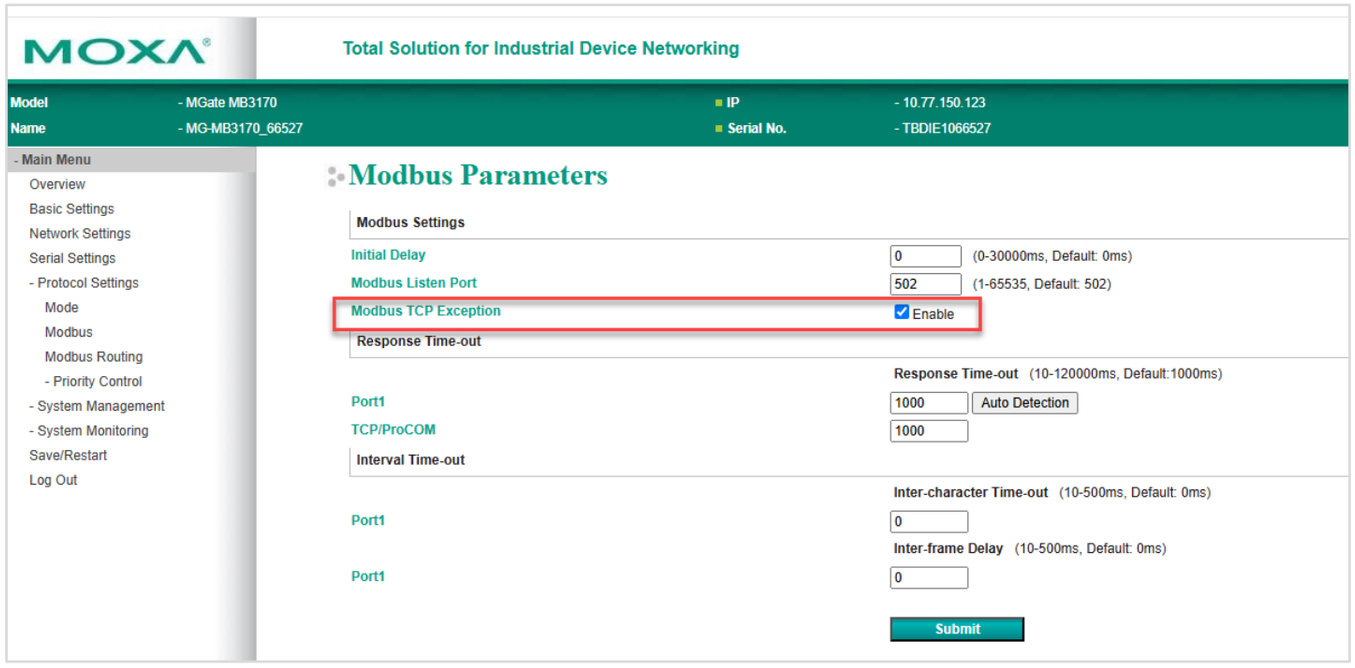
Преобразователи протоколов Modbus TCP/Modbus Serial. Часть 4

Давайте вернёмся к первому сценарию – Modbus TCP Master отправляет преобразователю запрос, который конвертируется в формат Modbus RTU (или Modbus ASCII) и отправляется в COM–порт. Предположим, что slave–устройство, которому предназначался этот запрос, вышло из строя – соответственно, ответа от него не последует. Что должен делать преобразователь в этом случае?

В принципе, он может не делать ничего – спустя время, определяемое настройкой таймаута, master–устройство диагностирует отсутствие ответа и отправит следующий запрос. Но возникает неудобство – получается, что с точки зрения master–устройства не будет возможности отличить отсутствие связи со slave–устройством от отсутствия связи с самим преобразователем.

Для предотвращения такой ситуации в спецификации Modbus предусмотрен специальный код ошибки – **0x0B (GATEWAY TARGET DEVICE FAILED TO RESPOND)**. Если преобразователь позволяет настроить собственный таймаут ожидания ответа – то по его истечении он может отправить master–устройству ответ с этим кодом. Это позволит master–устройству определить, что преобразователь продолжает функционировать, а вот slave–устройство, которому предназначался запрос – нет.

В некоторых преобразователях для отправки этой ошибки требуется установить специальную настройку. Например, у Моха MGate она называется **Modbus TCP Exception**.



Второй и последней специфической ошибкой протокола Modbus, которую может отправить преобразователь, является **0x0A (GATEWAY PATH UNAVAILABLE)**. Согласно спецификации – она может быть отправлена master–устройству в двух ситуациях:

- если полученный запрос не удаётся маршрутизировать (например, его Unit ID не был задан в настройках преобразователя и поэтому нельзя определить, в какой COM–порт он должен быть отправлен);
- если очередь запросов преобразователя переполнена (например, ещё одно master–устройство ведёт интенсивный обмен, «полностью заняв под себя» COM–порт).

Преобразователи протоколов Modbus TCP/Modbus Serial. Часть 5

Продвинутые преобразователи протоколов обычно предоставляют пользователю дополнительный функционал. Рассмотрим его на примере преобразователя **Moxa MGate MB3660**.

https://moxa.ru/shop/modbus/mgate_mb3660/mgate–mb3660–16–2ac/

В дополнение к «классическому» режиму работы, в котором шлюз последовательно пересылает запросы от master–устройства slave–устройствам и возвращает ответы в обратном направлении, MGate поддерживает ещё два режима – **режим агента** и **интеллектуальный режим**.

В **режиме агента** преобразователь не ждёт запросов от master–устройства. Вместо этого он самостоятельно производит опрос slave–устройств согласно заданным пользователем настройкам и размещает их данные в своей памяти по заданным адресам битов и регистров. Master–устройство обращается к этой памяти, производя чтение и запись значений. Это особенно эффективно для многопортовых шлюзов, которые могут параллельно отправлять запросы по каждому из своих COM–портов. Кроме того, количество подключённых к шлюзу master–устройств не влияет на производительность, потому что все они получают доступ к уже «собранным» данным.

Режим агента требует от пользователя настройки каждого запроса, который будет отправляться шлюзом. В некоторых случаях таких запросов может быть много, и их подготовка займёт значительное время. Сэкономить его позволяет так называемый **интеллектуальный режим** – в нём преобразователь сначала работает «классическим» образом, но параллельно «запоминает» поступающие на него запросы от master–устройств и вскоре начинает производить их самостоятельную отправку. Когда master–устройство отправит уже «изученный» запрос – то шлюз не будет пересылать его в другой интерфейс, а мгновенно отправит в ответ данные, которые были считаны им самим при последнем сеансе опроса.

Список всех «изученных» запросов можно посмотреть в web–конфигураторе или конфигурационной утилите MGate Manager.

https://moxa.ru.com/blogs/articles/vmeste–s–mgate–mb3660–vy–poluchite–dannye–ot–modbus–rtu–ustroystv–v–10–raz–bystree

***

Когда master–устройство присылает запрос многопортовому преобразователю, то возникает вопрос – в какой из COM–портов его нужно переслать? Мы уже обсуждали, что технически можно закрепить за каждым из COM–портов отдельный TCP–порт. Но, предположим, этого не было сделано.

В этом случае окажется полезной функция **Auto Device Routing**. Она работает следующим образом: впервые получив запрос с новым Unit ID, шлюз последовательно отправляет его через каждый из своих COM–портов, дожидаясь ответа. При получении ответа данный Unit «связывается» с этим COM–портом, и все последующие запросы с этим Unit ID будут отправляться именно в него.

***

Представим, что шлюз работает в «классическом» режиме и его одновременно опрашивают несколько master–устройств, которым нужны данные slave–устройств, подключённых к одному конкретному COM–порту шлюза. Обсуждая преобразователи интерфейсов, мы упоминали, что обычно это реализуется с помощью размещения поступающих запросов в очереди.

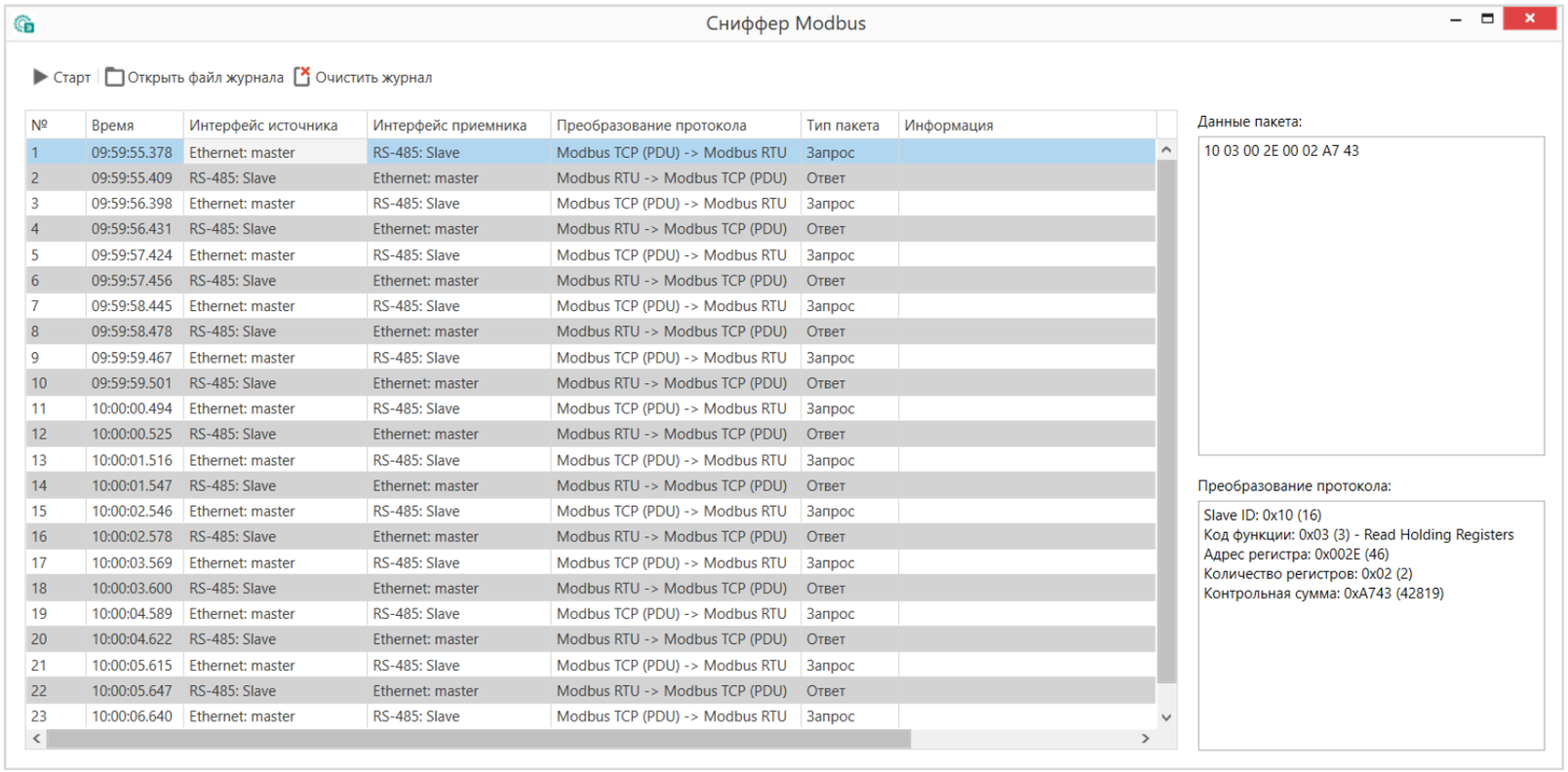
В некоторых случаях может быть полезной возможность приоритезации запросов, хранящихся в этой очереди. Например, нам требуется, чтобы ПЛК, подключённый к шлюзу, получал от него данные как можно чаще, а запросы SCADA–системы обрабатывались по остаточному принципу.

MGate имеет три режима приоритезации:

- на уровне TCP–портов шлюза (например, чтобы запросы, поступающие на порт 502, обрабатывались в первую очередь);
- на уровне IP–адреса master–устройства;
- на уровне параметров запроса (Unit ID, кода функции и т. д.).

***

При работе с уже настроенным шлюзом может возникнуть потребность в изучении запросов, передаваемых по каждому из его интерфейсов. Конечно, можно было бы для этого воспользоваться утилитами, изученными нами в уроке 2.17, но если в конфигурационном ПО шлюза есть встроенный сниффер – то проще воспользоваться им. Например, такой сниффер есть в Moxa MGate Manager, а также в утилите OWEN Configurator, используемой для настройки преобразователя ОВЕН МКОН – ниже приведён скриншот из неё:



№	Время	Интерфейс источника	Интерфейс приемника	Преобразование протокола	Тип пакета	Информация
1	09:59:55.378	Ethernet: master	RS-485: Slave	Modbus TCP (PDU) -> Modbus RTU	Запрос	
2	09:59:55.409	RS-485: Slave	Ethernet: master	Modbus RTU -> Modbus TCP (PDU)	Ответ	
3	09:59:56.398	Ethernet: master	RS-485: Slave	Modbus TCP (PDU) -> Modbus RTU	Запрос	
4	09:59:56.431	RS-485: Slave	Ethernet: master	Modbus RTU -> Modbus TCP (PDU)	Ответ	
5	09:59:57.424	Ethernet: master	RS-485: Slave	Modbus TCP (PDU) -> Modbus RTU	Запрос	
6	09:59:57.456	RS-485: Slave	Ethernet: master	Modbus RTU -> Modbus TCP (PDU)	Ответ	
7	09:59:58.445	Ethernet: master	RS-485: Slave	Modbus TCP (PDU) -> Modbus RTU	Запрос	
8	09:59:58.478	RS-485: Slave	Ethernet: master	Modbus RTU -> Modbus TCP (PDU)	Ответ	
9	09:59:59.467	Ethernet: master	RS-485: Slave	Modbus TCP (PDU) -> Modbus RTU	Запрос	
10	09:59:59.501	RS-485: Slave	Ethernet: master	Modbus RTU -> Modbus TCP (PDU)	Ответ	
11	10:00:00.494	Ethernet: master	RS-485: Slave	Modbus TCP (PDU) -> Modbus RTU	Запрос	
12	10:00:00.525	RS-485: Slave	Ethernet: master	Modbus RTU -> Modbus TCP (PDU)	Ответ	
13	10:00:01.516	Ethernet: master	RS-485: Slave	Modbus TCP (PDU) -> Modbus RTU	Запрос	
14	10:00:01.547	RS-485: Slave	Ethernet: master	Modbus RTU -> Modbus TCP (PDU)	Ответ	
15	10:00:02.546	Ethernet: master	RS-485: Slave	Modbus TCP (PDU) -> Modbus RTU	Запрос	
16	10:00:02.578	RS-485: Slave	Ethernet: master	Modbus RTU -> Modbus TCP (PDU)	Ответ	
17	10:00:03.569	Ethernet: master	RS-485: Slave	Modbus TCP (PDU) -> Modbus RTU	Запрос	
18	10:00:03.600	RS-485: Slave	Ethernet: master	Modbus RTU -> Modbus TCP (PDU)	Ответ	
19	10:00:04.589	Ethernet: master	RS-485: Slave	Modbus TCP (PDU) -> Modbus RTU	Запрос	
20	10:00:04.622	RS-485: Slave	Ethernet: master	Modbus RTU -> Modbus TCP (PDU)	Ответ	
21	10:00:05.615	Ethernet: master	RS-485: Slave	Modbus TCP (PDU) -> Modbus RTU	Запрос	
22	10:00:05.647	RS-485: Slave	Ethernet: master	Modbus RTU -> Modbus TCP (PDU)	Ответ	
23	10:00:06.640	Ethernet: master	RS-485: Slave	Modbus TCP (PDU) -> Modbus RTU	Запрос	

5.4 Каплеры и арбитры

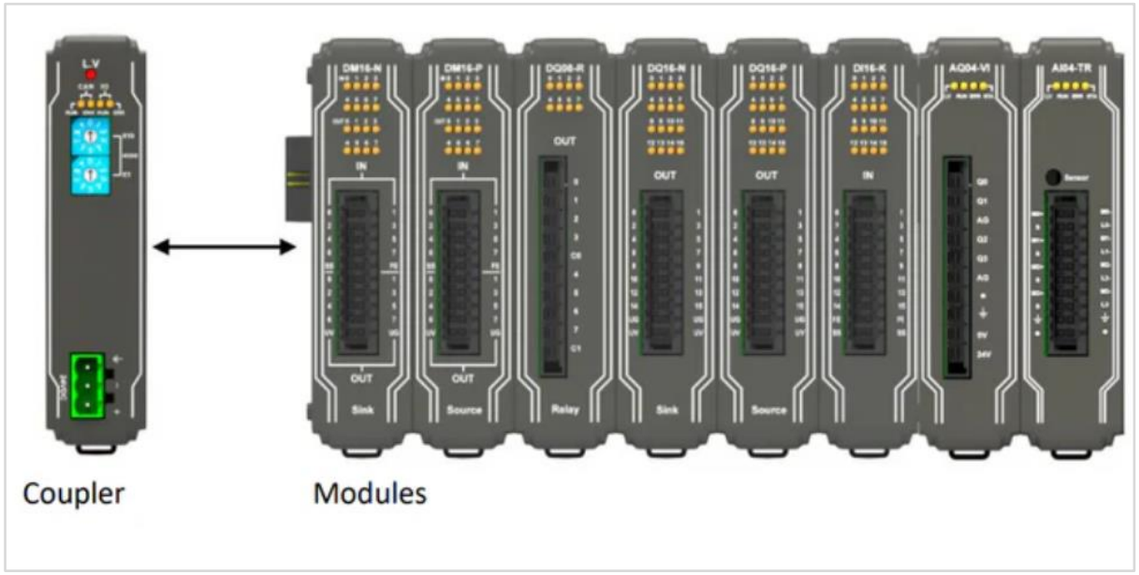
Каплеры

Каплеры – это специфический вид шлюзов, предназначенных для «активного» опроса slave–устройств. В прошлом уроке мы рассмотрели режим агента шлюзов Моха MGate. Для шлюза такой функционал является довольно редким, а для каплера – основным.

Задача каплера – производить опрос slave–устройств и предоставлять их данные master–устройствам. Набор поддерживаемых интерфейсов и протоколов зависит от конкретной модели каплера. Например, на рисунке ниже показан каплер iR–ETN от компании Weintek, который в режиме Master производит опрос модулей линейки iR по системной шине и предоставляет их данные в режиме Slave по протоколам Modbus TCP и EtherNet/IP. Для линейки iR существуют и другие модели каплеров: iR–COP с протоколом CANopen и iR–ECAT с протоколом EtherCAT.

Карта регистров каплера iR–ETN формируется автоматически и зависит от модификаций, количества и порядка подключённых модулей.

Некоторые каплеры других производителей позволяют тонко настраивать обмен – на уровне запросов, отправляемых каплером, и адресов регистров, в которых размещаются данные slave–устройств; иногда даже имеется функционал для преобразования данных (конвертация типа, масштабирование и т. д.).



https://www.weintek.com/globalw/Product/Product_speciR.aspx

Арбитры

Отдельным видом устройств, отчасти схожих с шлюзами, являются арбитры (они могут называться и по–другому: «маршрутизаторы RS–485» и т. п.).

Задача арбитра – предоставить возможность подключения нескольких master–устройств к одной последовательной линии связи. Представим, что у нас уже есть действующая система управления, в которой ПЛК в режиме Modbus RTU Master по интерфейсу RS–485 производит опрос slave–устройств, и требуется добавить в неё панель оператора.

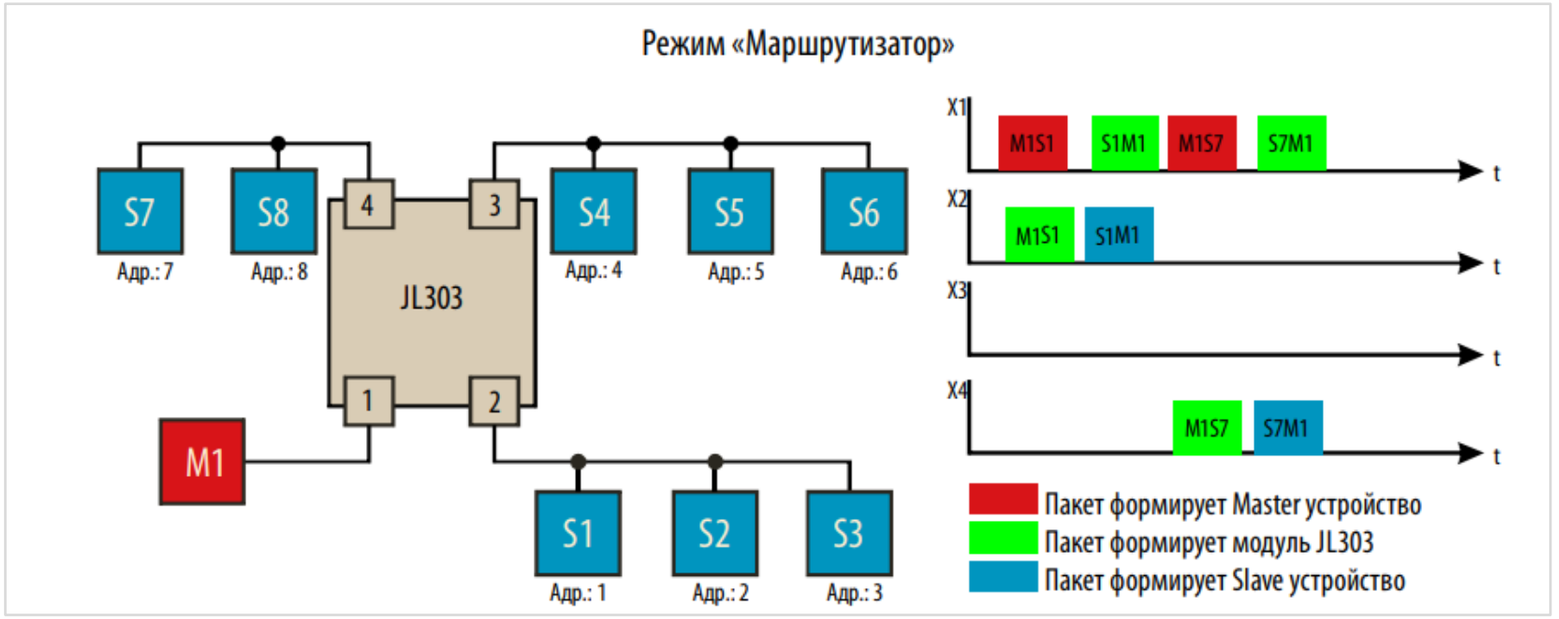
Разумным решением было бы настроить ПЛК как Modbus Slave и подключить к нему панель, которая будет работать в режиме Modbus Master. Но, предположим, такой возможности нет (например, нет исходников проекта ПЛК). Подключить два master–устройства к последовательной линии связи не получится – их запросы будут «накладываться» друг на друга. Поэтому потребуется воспользоваться специализированным промежуточным устройством.

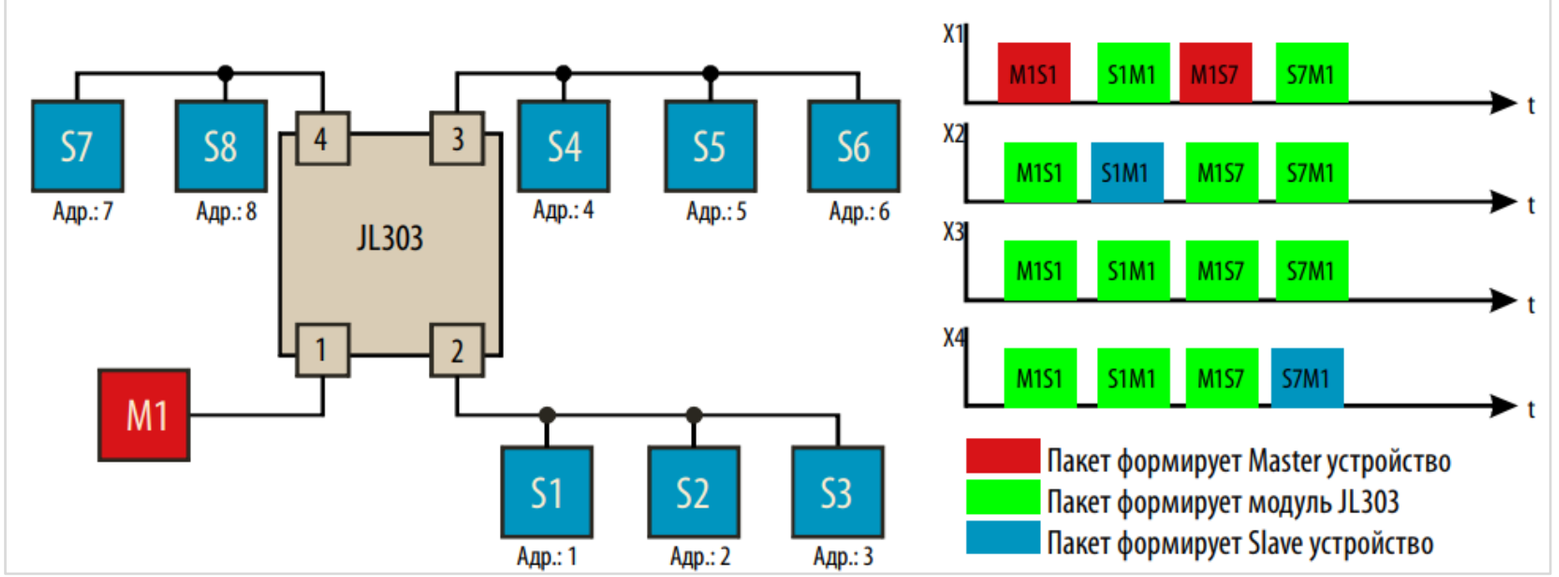
Примерами таких устройств является PL303/JL303 от компании JET LOGIC и Arbitr485–120 от компании eSkomatik.

Рассмотрим принцип их работы на примере PL303, который имеет 4 интерфейса RS–485 и может использоваться в 3 режимах:

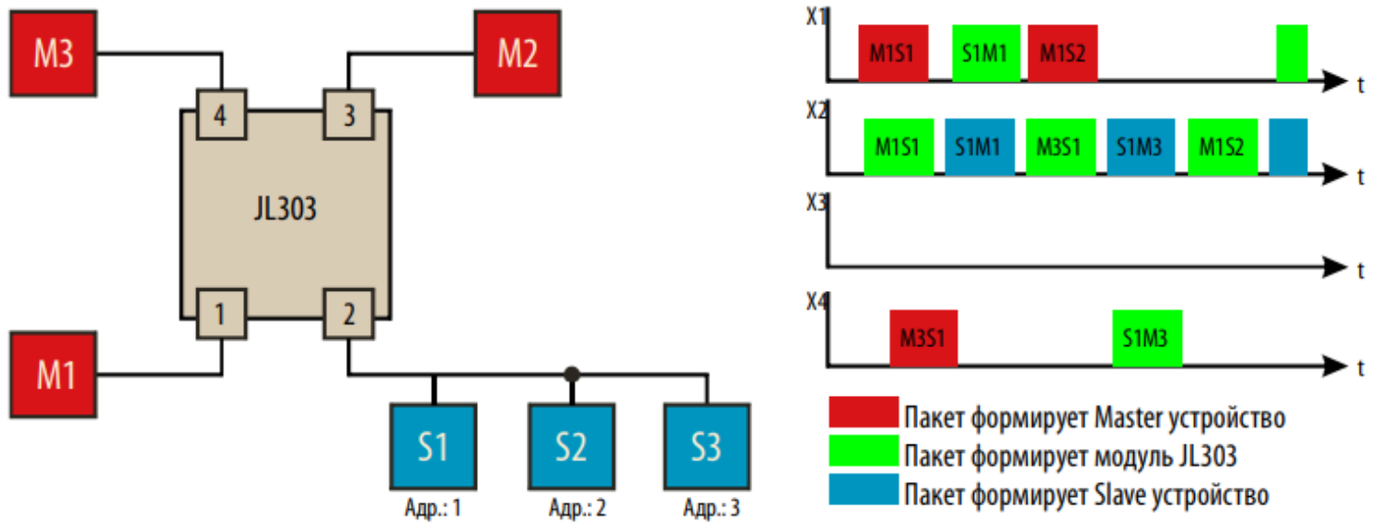
- **концентратор** – в этом режиме пакет, поступивший в один из COM–портов преобразователя, пересылается во все остальные. Таким образом, в этом случае PL303 совмещает функции повторителя и разветвителя интерфейса RS–485, которые мы рассмотрели в уроке 2.14;
- **маршрутизатор** – в этом режиме в пакете, поступившем в один из COM–портов PL303, анализируется значение поля Slave ID (на основании настроенной пользователем таблицы маршрутизации), в результате чего пакет пересылается в заданный COM–порт с подменой значения поля Slave ID на заданное. Это позволяет организовать опрос slave–устройств с одинаковыми адресами (предположим, изменить его в настройках slave–устройства по каким–то причинам нельзя);
- **маршрутизатор + мастер–арбитр** – в этом режиме к двум или трём портам преобразователя подключаются master–устройства, а к остальным двум или одному – slave–устройства. Master–устройства независимо друг от друга отправляют запросы, а PL303 **поочерёдно** пересылает их slave–устройствам с подменой поля Slave ID (если настроена таблица маршрутизации). Именно этот режим потребовался бы для приведённого в начале шага сценария с подключением панели оператора к последовательной линии связи в роли второго master–устройства.

Arbitr485–120 имеет два интерфейса RS–485 и может использоваться только в режиме арбитра.

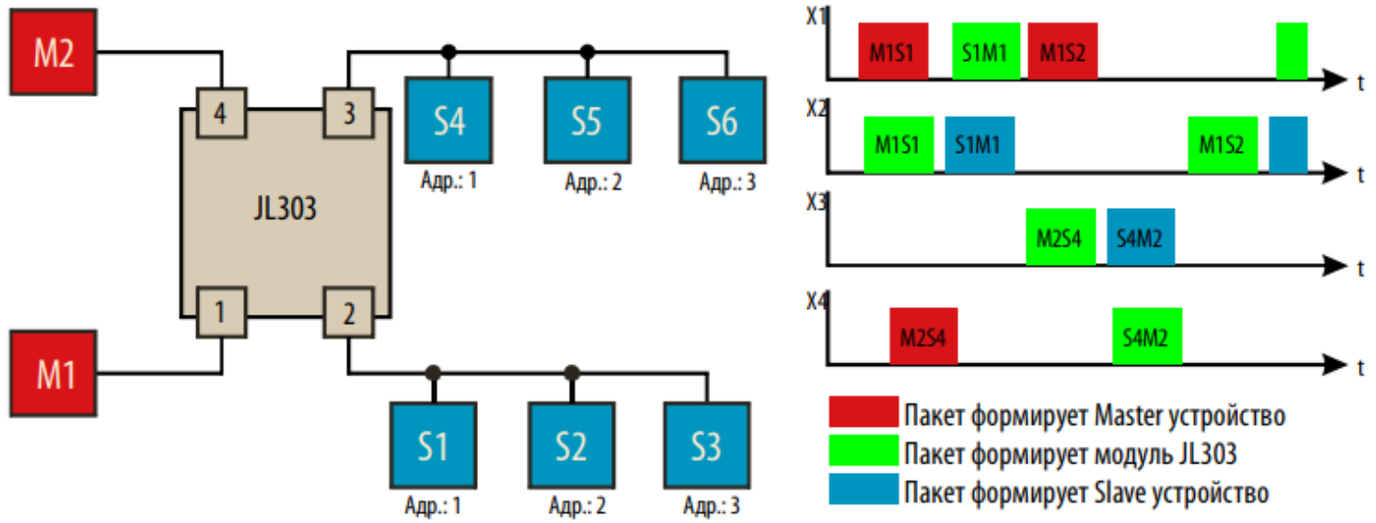




Режим «Маршрутизатор+Мастер-арбитр» с одной веткой Slave-устройств



Режим «Маршрутизатор+Мастер-арбитр» с двумя ветками Slave-устройств



5.5 Тестовые задания

Ответы приведены в п. 9.

1. Какое устройство требуется для настройки обмена между приборами, один из которых поддерживает только протокол Modbus TCP, а другой – только Modbus RTU?
   - Коммутатор
   - Конвертер интерфейсов
   - Арбитр
   - Конвертер протоколов

2. Требуется настроить опрос Modbus RTU Slave–устройства, подключённого к преобразователю интерфейсов Ethernet/COM. Какой протокол обмена следует использовать на стороне master–устройства?
   - Modbus TCP over RTU
   - Modbus TCP
   - Modbus RTU
   - Modbus RTU over TCP

3. Сопоставьте режимы работы шлюза Moxa MGate с их описанием:

Классический	Пересылает запросы от master к slave без обработки данных
Интеллектуальный	«Запоминает» поступающие запросы с дальнейшей самостоятельной отправкой
Агент	Самостоятельно опрашивает slave-устройства и кэширует результаты

4. Что такое «прозрачный шлюз»?
   - Устройство, способное преобразовывать протокол в обе стороны
   - Шлюз с маршрутизацией трафика
   - Устройство, передающее данные между интерфейсами без преобразования протокола
   - Устройство для прослушки трафика сети

6. Modbus в системах диспетчеризации

6.1 OPC–серверы и облачный сервис OwenCloud

В прошлых модулях мы рассматривали реализацию протокола Modbus в устройствах, относящихся к «среднему» уровню системы автоматизации – модулях ввода–вывода, регуляторах, контроллерах, панелях оператора и коммуникационных шлюзах.

Modbus может также применяться для передачи данных между средним и верхним уровнем АСУ. Верхний уровень обычно представлен ПК, на которых установлено специализированное ПО для отображения, обработки и сохранения информации. Характерным представителем такого ПО являются SCADA–системы.

Некоторые SCADA–системы имеют встроенные коммуникационные драйвера, и тогда настройка обмена по Modbus для них не имеет существенных отличий от, например, настройки обмена в ПЛК. Другие поддерживают только технологию OPC, и тогда для обмена по Modbus требуется ещё одна программа – OPC–сервер. Мы уже обсуждали OPC в уроке 2.1, и в ходе курса регулярно использовали **MasterOPC Universal Modbus Server**, разработанный компанией МПС софт.

Давайте обзорно рассмотрим настройки обмена по Modbus, которые присутствуют в данном ПО.

MasterOPC Universal Modbus Server. Настройки узла

> *Приведённая далее информация основана на материалах онлайн–справки **MasterOPC Universal Modbus Serve**r:*
>
> *https://support.mps–soft.ru/ModbusHelpRus/index.html*

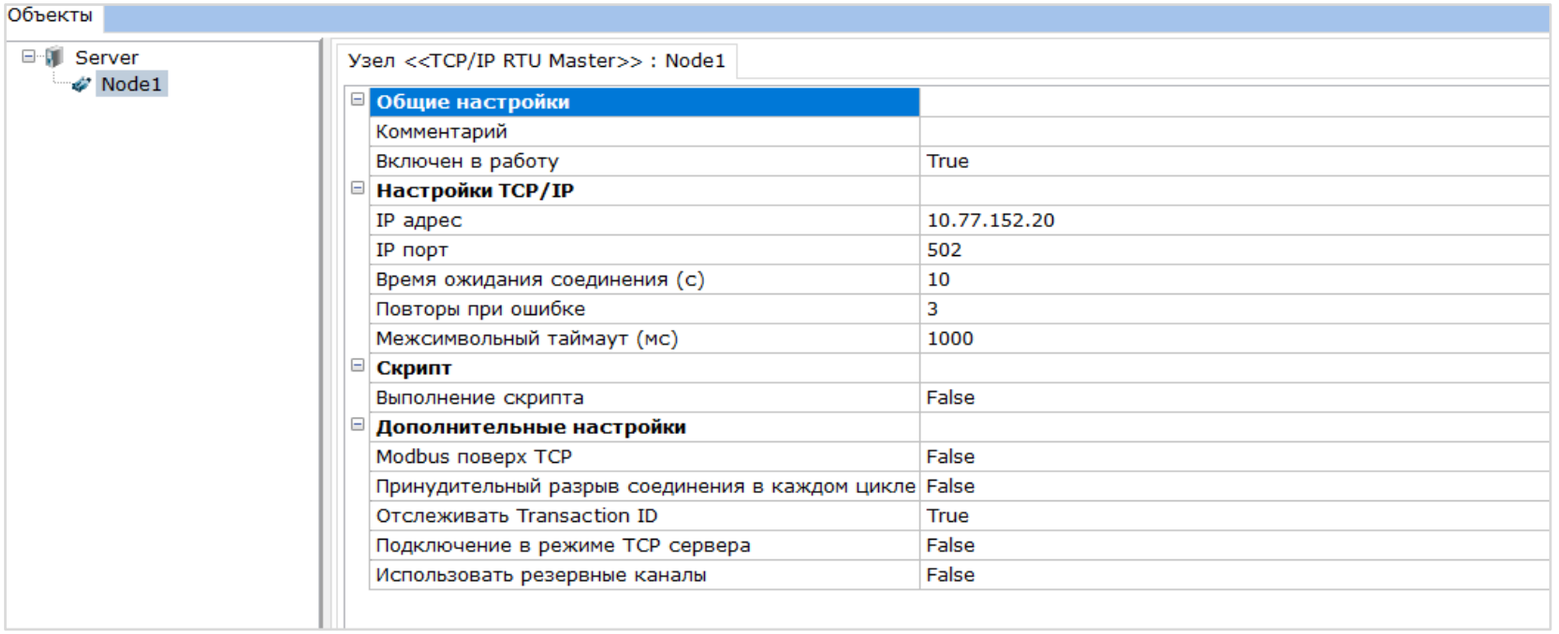
Для настройки обмена в режиме Modbus Master в OPC–сервере используются три типа объектов:

- **узел**, который соответствует используемому сетевому интерфейсу;
- **устройство**, которое соответствует опрашиваемому slave–устройству;
- **тег**, который соответствует одному параметру slave–устройства.

Настройки интерфейса рассмотрим на примере узла **TCP/IP**:

- IP–адрес и порт опрашиваемого slave–устройства;
- **Время ожидания соединения** – интервал времени, выделяемый на установку TCP–соединения со slave–устройством;
- **Повторы при ошибке** – количество повторных попыток установки соединения с устройством после ошибки соединения. Если все попытки установки соединения закончились неудачей, то для тегов устройства устанавливается статус **OPC_QUALITY_BAD**, что позволяет SCADA–системе определить отсутствие у них достоверных значений;
- **Межсимвольный таймаут** – максимально допустимый интервал между Ethernet–пакетами. Если интервал между пакетами меньше, то происходит«склейка» пакетов;
- **Modbus поверх TCP** – в случае установки данного параметра в значение **TRUE**, OPC–сервер использует не протокол **Modbus TCP**, а протокол **Modbus RTU over TCP.** В прошлом модуле мы выяснили, что он требуется для передачи данных через преобразователи интерфейсов (прозрачные шлюзы);
- **Принудительный разрыв соединения в каждом цикле** – при установке данного параметра в значение **TRUE** после очередного запроса slave–устройству происходит разрыв TCP–соединения, а перед отправкой следующего запроса – установка нового соединения. Это может быть полезным, когда период опроса достаточно велик, и нет необходимости поддерживать TCP–соединение в течение интервала времени между запросами (например, при опросе устройств с ограниченным аппаратными ресурсами) или при использовании канала связи с оплачиваемым трафиком (например, GPRS), чтобы снизить расходы;

- **Отслеживать Transaction ID** – согласно спецификации Modbus TCP, значение поля **Transaction ID** в ответе должно совпадать с тем, которое указано в запросе. Но некоторые slave–устройства отступают от спецификации и всегда отправляют в ответ одно и то же значение **Transaction ID** (чаще всего – **0**). Чтобы OPC–сервер не расценивал это как ошибку – нужно установить данный параметр в значение **FALSE**;
- **Подключение в режиме TCP–сервера** – в этом режиме OPC–сервер ожидает входящих подключений по-своему IP–адресу и порту, после чего начинает обмен в режиме Master. Это требуется для опроса slave–устройств, подключённых к сети Интернет, но не имеющих «белого» статического IP–адреса. В этом случае такое устройство может в режиме клиента установить TCP–соединение с сервером (на котором запущен OPC–сервер), имеющим «белый» статический IP–адрес (или фиксированное доменное имя), после чего OPC–сервер в режиме Master начнёт опрос устройства в рамках установленного им соединения. Более подробная информация приведена в этом документе: http://www.insat.ru/products/Universal_MasterOPC/work_with_modem.pdf ;
- **Использовать резервные каналы** – если этот параметр установлен в значение **TRUE**, то открывается новая группа настроек, в рамках которой настраиваются дополнительные каналы связи. Это полезно при использовании резервируемых ПЛК и подобных устройств – если один из них перестанет отвечать, то OPC–сервер сможет начать отправлять запросы второму ПЛК.



У других типов узлов присутствуют специфические настройки – например, для узла типа **COM** можно задать настройки модема (если для обмена используется GPRS–модем, подключённый к COM–порту ПК).

MasterOPC Universal Modbus Server. Настройки устройства

Существенная часть настроек уровня устройства нам уже известна и интуитивно понятна. Рассмотрим только те, с которыми мы ещё не сталкивались:

- **Реинициализация узла при ошибке** – если параметр имеет значение **TRUE** и заданное число попыток отправки запроса (определяемое параметрами **Повторы при ошибке** и **Повторы при ошибке записи**) исчерпано, то произойдёт или переоткрытие соединения, или закрытие/открытие COM–порта – в зависимости от типа узла;
- **Сброс команды записи при разрыве соединения** – если параметр имеет значение **TRUE**, то в случае разрыва соединения со slave–устройством неотправленные запросы записи удаляются из буфера OPC–сервера и не будут отправлены после возобновления обмена. Это полезно, так как к моменту восстановления связи эти запросы, вероятно, уже утратят актуальность;
- **Начальная фаза** – время от запуска OPC–сервера до отправки первого запроса slave–устройству. Значение параметра задаётся как смещение от начала суток (размерность задаётся параметром **Размерность фазы**). Настройка предназначена для разнесения опроса устройств по времени. Например: нужно опрашивать устройства раз в минуту с шагом в 15 секунд. Период опроса устанавливается равным 1 минуте, начальная фаза устройства 1 – 15 секунд, начальная фаза устройства 2 – 30 секунд. Первое устройство будет опрашиваться на 15 секунде каждой минуты, второе – на 30. Параметру **Старт после запуска** следует присвоить значение **FALSE** – в противном случае все устройства будут опрошены при старте, и лишь потом начнут опрашиваться согласно фазе опроса. Один из сценариев использования данной настройки – опрос slave–устройства с помощью нескольких master–устройств через последовательный интерфейс связи (например, через GPRS–модем с интерфейсом RS–485); в этом случае нужно обеспечить, чтобы master–устройства никогда не отправляли свои запросы одновременно, и настройки фазы позволяют это сделать;
- **Использовать преамбулу** – если этот параметр имеет значение **TRUE**, то перед отправкой запроса OPC–сервер отправит slave–устройству заданную последовательность байт. Это может быть полезным при настройке обмена с конкретными специфическими устройствами – например, в документации на теплосчётчик ВКТ7 с интерфейсом RS–232 указано, что перед оправкой Modbus–запроса необходимо передать не менее двух байт 0xFF для гарантированного «вывода» прибора из «спящего» режима;
- **Использовать ретрансляцию** – назначение этого параметра не описано в документации OPC–сервера.

Отметим, что первые 5 настроек вкладки **Настройки запросов** уже знакомы нам по модулю 4, в котором мы рассматривали реализацию Modbus в панелях оператора – они позволяют повлиять на автоформирование групповых запросов, указать максимальное количество регистров в групповых запросах и выбрать функцию записи для тегов, которые занимают один бит или регистр.

Объекты

- Server
  - Node1

Устройство <<MODBUS>> : Device4

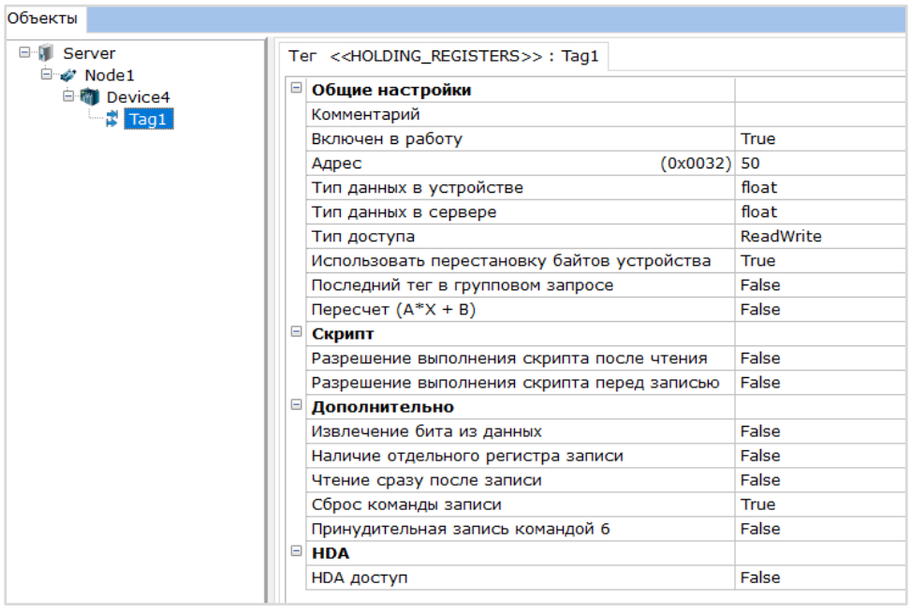
Общие настройки		
Комментарий		
Включено в работу		True
Адрес	(0x01)	1
Время ответа (мс)		1000
Повторы при ошибке		3
Повторы при ошибке записи		3
Сброс команд записи при разрыве соединения		True
Повторное соединение после ошибки через (с)		10
Реинициализация узла при ошибке		False
Период опроса		1000
Размерность периода опроса		ms
Начальная фаза		0
Размерность фазы		ms
Старт после запуска		True
Задержка запроса после получения ответа (мс)		4
Перестановка байтов в значении		Вызов редактора перестановки байтов...
Дополнительные настройки модема		
Наличие собственного номера телефона		False
Скрипт		
Выполнение скрипта		False
Настройка запросов		
Максимальное количество HOLDING регистров в запросе чтения		125
Максимальное количество INPUT регистров в запросе чтения		125
Не использовать команду WRITE_SINGLE_COIL (0x05)		True
Не использовать команду WRITE_SINGLE_REGISTER (0x06)		True
Максимальное допустимый разрыв адресов в запросе чтения		0
Использовать преамбулу		False
Использовать ретрансляцию		False

MasterOPC Universal Modbus Server. Настройки тега

Как и в панелях оператора – во многих OPC–серверах функция Modbus, используемая в запросе, выбирается не в явном виде, а косвенно – через регион (область памяти slave–устройства).

Интересно, что тип данных надо указать два раза – для устройства и сервера. Это связано с тем, что в OPC–сервере нет поддержки типов Int16 и Uint16. Вместо них можно использовать Int32 и Uint32 – и, в принципе, разработчики могли предусмотреть автоматический выбор соответствующего типа для сервера, убрав необходимость задавать его пользователю.

- **Тип доступа** – этот параметр влияет на запросы, которые отправляет OPC–сервер для данного тега (**ReadOnly** – только запросы чтения, **WriteOnly** – только запросы записи, **ReadWrite** – и запросы чтения, и запросы записи). Отправка запроса записи в OPC производится только при изменении значения тега SCADA–системой или при его ручном изменении в интерфейсе OPC–сервера, запущенного в режиме отладки. Организовать циклическую запись можно только с помощью написания скриптов для OPC–сервера;
- **Использовать перестановку байт устройства** – этот параметр позволяет тонко настроить порядок байт/регистров, применяемый к значению данного тега. Как мы неоднократно наблюдали в ходе курса – это крайне полезная настройка. Её можно задать как на уровне всего устройства, так и на уровне отдельного тега;
- **Пересчёт** – этот параметр позволяет задать коэффициенты линейного масштабирования для значения числового тега;
- **Извлечение бита из данных** позволяет использовать тег для доступа к заданному биту регистра;
- **Наличие отдельного регистра записи** – этот параметр позволяет указать разные адреса регистра для запросов чтения и записи данного тега;
- **Чтение сразу после записи** – если этот параметр имеет значение **TRUE**, то после запроса записи отправляется внеочередной запрос чтения;
- **Сброс команды записи** – если этот параметр имеет значение **TRUE**, то после отправки запроса записи, на который не был получен ответ или был получен ответ с кодом ошибки Modbus, OPC–сервер не делает попыток повторно отправить этот запрос;
- **Принудительная запись командой 6** – если этот параметр имеет значение **TRUE** и тег имеет тип **Int16** или **Uint16**, то для записи его значения используется функция **0x06 (Write Single Register)**. В противном случае используется функция **0x10 (Write Multiple Registers)**. Эту настройку можно задать как на уровне всего устройства, так и на уровне отдельного тега.



Облачный сервис OwenCloud. Основная информация

Напоследок рассмотрим ещё одно ПО верхнего уровня системы автоматизации, которое работает в режиме Modbus Master – облачный сервис **OwenCloud**. Он позволяет организовать сбор данных и управление удалённым оборудованием, отображение этих данных в виде мнемосхем, таблиц, графиков и отчётов, отправку аварийных сообщений и т. д.

https://owen.ru/owencloud

https://web.owencloud.ru/

По функционалу облачный сервис близок к ограниченной по возможностям SCADA–системе, которая запускается не на локальном ПК, а на интернет–сервере, и используется для сбора данных с удалённо расположенных устройств.

Существует три основных способа подключения slave–устройств к облачному сервису. Два из них доступны для устройств с интерфейсом Ethernet, а последний – для устройств с интерфейсом RS–485.

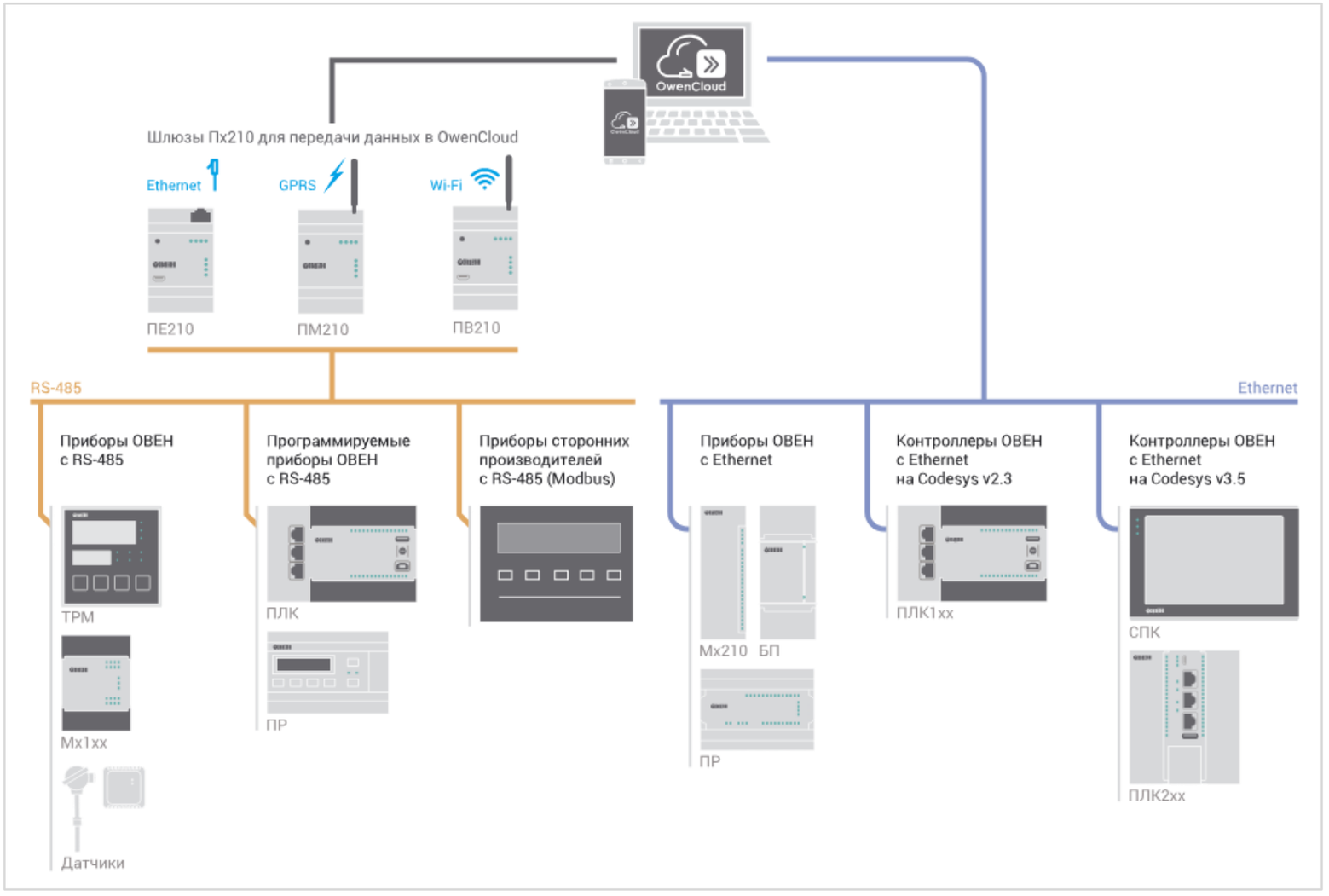
По интерфейсу Ethernet поддерживается подключение только устройств компании ОВЕН. Некоторые из них имеют функционал автоматического подключения к OwenCloud. В этом случае имеющиеся в них параметры экспортируются в облачный сервис автоматически. Чтение и запись данных осуществляется с помощью функций Modbus **0x14 (Read File Record)** и **0x15 (Write File Record)**. Второй способ – это ручная настройка, в рамках которой пользователю нужно вручную добавить каждый параметр опрашиваемого устройства. Поддерживается импорт списков параметров из сред разработки **CoDeSys V2.3** и **Owen Logic**. Их чтение и запись осуществляется с помощью стандартных функций Modbus – **0x03 (Read Holding Registers)**, **0x10 (Write Multiple Registers)** и т. д.

По интерфейсу RS–485 поддерживается подключение slave–устройств любых производителей через шлюзы линейки Пх210.

В данный момент доступно три модели шлюзов, каждая из которых имеет один интерфейс RS–485 и один специализированный интерфейс для подключения к OwenCloud:

- для ПМ210 таким интерфейсом является встроенный GPRS–модем;
- для ПЕ210 – Ethernet, через который шлюз должен быть подключен к сети, имеющей доступ в Интернет;
- для ПВ210 – Wi–Fi, через который шлюз должен быть подключен к сети, имеющей доступ в Интернет.

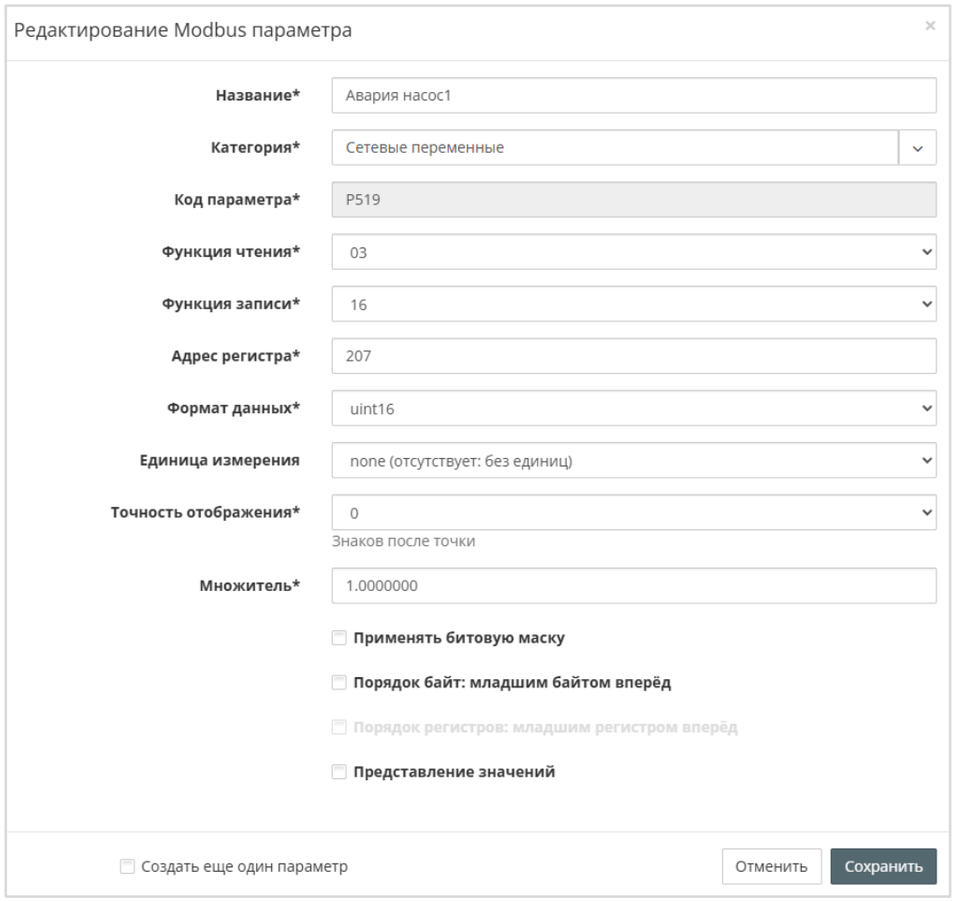
В этом случае настройка опрашиваемых параметров также производится вручную в личном кабинете OwenCloud.


OwenCloud
Шлюзы Пх210 для передачи данных в OwenCloud
Ethernet
GPRS
Wi-Fi
ПЕ210
ПМ210
ПВ210
RS-485
Ethernet
Приборы ОВЕН с RS-485
Программируемые приборы ОВЕН с RS-485
Приборы сторонних производителей с RS-485 (Modbus)
Приборы ОВЕН с Ethernet
Контроллеры ОВЕН с Ethernet на Codesys v2.3
Контроллеры ОВЕН с Ethernet на Codesys v3.5
ТРМ
Мх1хх
Датчики
ПЛК
ПР
Мх210
БП
ПР
ПЛК1хх
СПК
ПЛК2хх

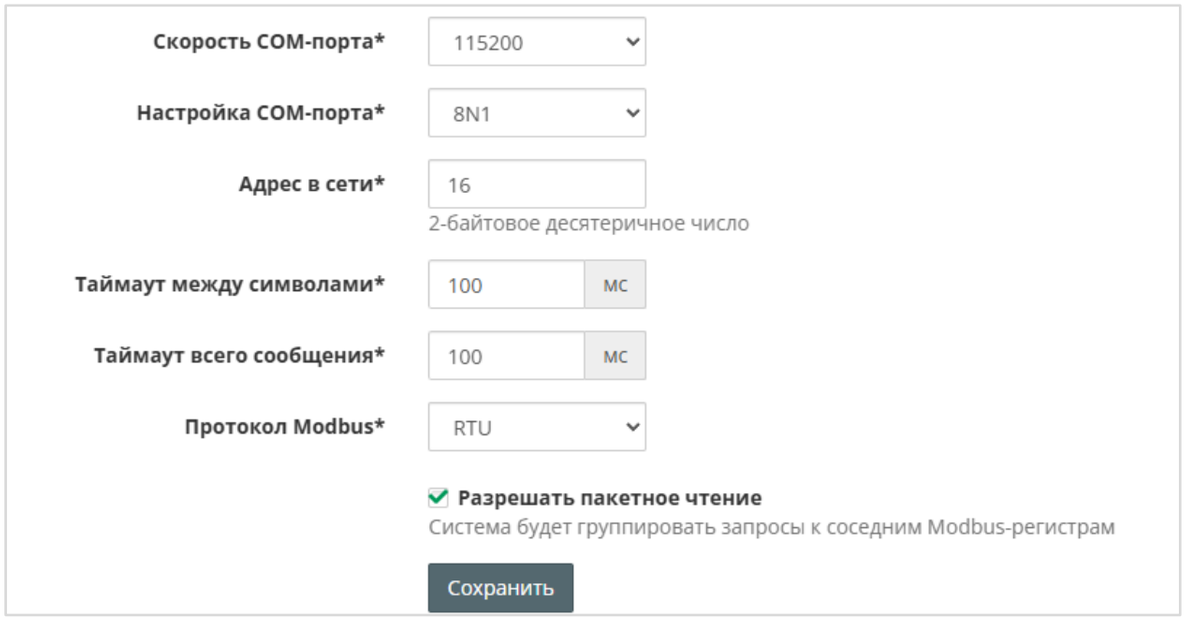
Облачный сервис OwenCloud. Настройки параметра произвольного Modbus–устройства

Рассмотрим настройки опроса одного параметра:

- **Код параметра** – уникальный идентификатор параметра, задаваемый на стороне облачного сервиса;
- **Функции чтения и записи** – функции Modbus, используемые для чтения и записи параметра;
- **Адрес регистра** – адрес начального регистра параметра в шестнадцатеричной системе счисления (HEX);
- **Формат данных** определяет тип параметра и, соответственно, количество занимаемых им регистров;
- **Точность отображения** позволяет указать количество отображаемых знаков после десятичной точки для параметров типа **Float** и **Double**;
- **Множитель** позволяет отмасштабировать считанное значение путём умножения на заданное число;
- галочка **Применить битовую маску** позволяет отобразить состояние заданного бита параметра;
- галочки **Порядок байт** и **Порядок регистров** работают понятным нам образом;
- галочка **Представление значений** доступна только для целочисленных типов и позволяет для конкретного значения параметра задать соответствующий ему текст (это работает по принципу перечислений [enumeration] в языках программирования).



На уровне настроек slave–устройства интерес представляет галочка **Разрешать пакетное чтение**. Если она установлена, то для чтения параметров используются групповые запросы. Группировка затрагивает только параметры с последовательными адресами регистров и одинаковым типом данных.



Как можно заметить – настройка обмена по Modbus в облачном сервисе не имеет существенных отличий от настроек обмена в OPC–сервере.

6.2 Тестовые задания

Ответы приведены в п. 9.

1. С помощью каких функций Modbus устройства ОВЕН с Ethernet автоматически (в режиме автоопределения) подключаются к облачному сервису OwenCloud?
   - 0x10 Write Multiple Registers
   - 0x15 (Write File Record)
   - 0x14 (Read File Record)
   - 0x03 (Read Holding Registers)
   - 0x06 Write Single Register
   - 0x04 (Read Input Registers)
   - 0x10 Read/Write Multiple Registers

2. Какие интерфейсы шлюзы Пх210 используют для подключения к OwenCloud?
   - Wi-Fi
   - USB
   - RS-485
   - LTE (4G)
   - Ethernet
   - GSM (2G)
   - Bluetooth

3. Сопоставьте задачи и типовой вариант их решения:

Управление объектом автоматизации с локальным доступом (например, объект электроэнергетики)	Облачный сервис
Сбор данных с нескольких приборов разных производителей с различными протоколами обмена	SCADA-система
Отображение и архивирование данных, получаемых с удаленных устройств, расположенных на разных объектах	OPC-сервер

4. Выберите все верные утверждения про основные функции OPC–сервера:
   - "Прокладка" между приборами и SCADA-системой
   - Маршрутизация сетевого трафика
   - Эмуляция master- и/или slave-устройств для тестирования обмена
   - Создание виртуальных COM-портов
   - Диспетчеризация и визуализация параметров

7. Дополнительные уроки

7.1 Специфические функции Modbus

В уроке 2.5 мы привели список всех функций, описанных в спецификации протокола Modbus [3, п. 6]:

Категория	Функции Modbus
Работа с регистрами (типовые функции)	0x03 Read Holding Registers 0x04 Read Input Registers 0x06 Write Single Register 0x10 Write Multiple Registers
Работа с битами	0x01 Read Coils 0x02 Read Discrete Inputs 0x05 Write Single Coil 0x0F Write Multiple Coils
Работа с регистрами (специфичные функции)	0x16 Mask Write Register 0x17 Read/Write Multiple Registers 0x18 Read FIFO queue
Работа с файлами	0x14 Read File Record 0x15 Write File Record
Диагностика (для RS-485/RS-232)	0x07 Read Exception Status 0x08 Diagnostic 0x0B Get Com Event Counter 0x0C Get Com Event Log 0x11 Report Server ID
Остальное	0x2B Encapsulated Interface Transport

В рамках курса мы рассмотрели большинство из них:

- типовые функции работы с регистрами использовались нами повсеместно;
- функции работы с битами мы рассмотрели в уроке 2.8 и ещё несколько раз упоминали их в дальнейших уроках;
- функции работы с файлами упоминались в уроке 2.21;
- функции диагностики были рассмотрены в уроке 2.15 – за исключением **0x11 (Report Server ID)**. В этом же уроке была рассмотрена функция **0x17 (Read/Write Multiple registers)**.

Таким образом, остались неупомянутыми следующие функции:

- 0x11 (Report Server ID);
- 0x16 (Mask Write Register);
- 0x18 (Read FIFO queue);
- 0x2B (Encapsulated Interface Transport).

Все эти функции используются крайне редко, но, тем не менее, заслуживают хотя бы короткого обзора.

Мы познакомимся с ними в следующих шагах.

Функция 0x11 (Report Server ID)

Функция **0x11 (Report Server ID)** определена только для Modbus Serial.

Запрос не включает никаких дополнительных параметров.

Пример запроса к slave–устройству с адресом 16 (0x10):

> *10 11 CC 7C*

Ответ должен содержать:

- поле количества байт данных (занимает 1 байт);
- идентификатор устройства (занимает произвольное количество байт);
- состояние светодиода RUN (занимает 1 байт; **0x00** соответствует отключенному индикатору, **0xFF** – включенному);
- другую произвольную информацию об устройстве (занимает произвольное количество байт).

> *«Произвольное количество байт» не является совсем произвольным; следует учесть, что размер пакета Modbus Serial не может превышать 256 байт.*

Изначально функция предназначалась для опроса контроллеров Modicon. Светодиод RUN соответствует режиму работы программы контроллера (запущена или остановлена). Что касается идентификатора устройства – он представлял собой модель контроллера. В состав «произвольной информации» входил заводской номер ПЛК, количество его входов/выходов, начальные адреса памяти для различных областей памяти (например, input–регистров) и т. д.

Когда Modbus стал универсальным протоколом – авторы спецификации не внесли уточнения в описание данной функции, поэтому у производителей оборудования просто не могло возникнуть понимания, для чтения каких данных она предназначена. Кроме того, все приходящие в голову параметры (заводской номер, модель прибора, версия прошивки и т. д.) проще разместить в регистрах устройства и предоставить к ним доступ функцией **0x03 (Read Holding Registers)** или **0x04 (Read Input Registers)**.

В качестве примера устройства с поддержкой данной функции можно рассмотреть индикатор СМИ2, в прошлом выпускаемый компанией ОВЕН (к настоящему моменту он снят с продаж и заменен на СМИ2–М). На запрос с функцией **0x11** индикатор отправляет следующий ответ:

> *10 11 0E **83 77 73 45 50 32 86 49 46 48 52 32 32 32** DD 6B*

Жирным выделены данные, которые занимают 14 байт (0x0E).

Это ASCII–символы строки 'SMI–2 V1.04 ' (без кавычек, с тремя пробелами в конце), в которой закодирована модель индикатора и версия его прошивки. Байт состояния светодиода отсутствует по той простой причине, что у индикатора его нет.

***

Интересно порассуждать о том, могли ли сделать авторы актуальной версии спецификации Modbus эту функцию более полезной.

Например, можно было бы использовать её для определения адреса slave–устройства в том случае, если он неизвестен.

> *Но в любом случае мы должны были бы знать настройки COM–порта slave–устройства.*

Мы могли бы подключить этот прибор к ПК (убедившись перед этим, что никаких других приборов к линии связи не подключено) и отправить запрос на определение его адреса (в качестве адреса устройства в запросе мы могли бы указать любой – всё равно настоящий адрес нам пока неизвестен):

> *10 11 CC 7C*

Предположим, адрес slave–устройства – 1. Тогда оно бы отправило нам следующий ответ:

> *10 11* ***01*** *BD 95*

И в своих следующих запросах (с другими кодами функций) мы бы уже использовали этот настоящий адрес.

Функция 0x16 (Mask Write Register)

Функция **0x16** близка в функции **0x06 (Write Single Register),** но вместо записи в holding–регистр конкретного значения она позволяет изменить значения его бит.

В уроке 4.2 мы рассматривали возможность привязки к переключателю панели оператора бита конкретного регистра и обсуждали, что панель считывает значение всего регистра, затем изменяет в нём привязанный к переключателю бит и производит запись этого изменённого значения в регистр. Функция **0x16** позволяет обойтись без предварительного чтения.

Предположим, что holding–регистр **4** slave–устройства с адресом **16** имеет значение **15**. В двоичной системе оно будет выглядеть так:

> *0000_0000_0000_1111*

Предположим, мы хотим преобразовать его в такое значение:

> ***11***00_0000_0000_11***00***

То есть надо установить два самых старших бита, а два самых младших – сбросить. Остальные биты должны сохранить свои текущие значения.

Тогда запрос будет выглядеть следующим образом:

> *10 16 00 04* ***3F FC*** *C0 00 5B 22*

- 3F FC – это битовая маска сбрасываемых бит (And_Mask);
- C0 00 – это битовая маска устанавливаемых бит (Or_Mask).

Новое значение регистра формируется по формуле:

`Result = (текущее значение регистра AND And_Mask) OR (Or_Mask AND (NOT And_Mask) )`

Давайте перейдем в двоичную систему и поэтапно рассмотрим приведённый выше пример, используя в своих рассуждениях таблицы истинности для операторов AND, OR и NOT.

1. Итак, текущее значение регистра = 0000_0000_0000_1111, и мы хотим преобразовать его в **11**00_0000_0000_11**00**.

Переведём в двоичную систему обе маски:

> *And_Mask = 3F FC = 0011_1111_1111_1100 (битовая маска сбрасываемых бит)*
>
> *Or_Mask = C0 00 = 1100_0000_0000_0000 (битовая маска устанавливаемых бит)*

2. Вычислим `(текущее значение регистра регистра **AND** And_Mask)`

> *0000_0000_0000_1111*
>
> ***AND***

0011_1111_1111_1100

=

*0000_0000_0000_**11**00 (сохранили те биты, которые не надо сбрасывать)*

3. Вычислим `(NOT And_Mask)`

> *NOT(0011_1111_1111_1100) = **11**00_000_0000_00**11***

4. Вычислим `(Or_Mask` **AND** `(NOT And_Mask) )`

> *1100_0000_0000_0000*
>
> ***AND***
>
> *1100_000_0000_0011 (из пп. 3)*
>
> *=*
>
> ***11**00_000_0000_0000 (подготовили маску устанавливаемых бит)*

5. Вычислим `(текущее значение регистра AND And_Mask)` **OR** `(Or_Mask AND (NOT And_Mask) )`

> *0000_0000_0000_1100 (из. пп. 2)*
>
> ***OR***
>
> *1100_000_0000_0000 (из пп. 4)*
>
> *=*
>
> ***11**00_0000_0000_**1100***

Получили именно то, что хотели.

Ответ на запрос с кодом функции **0x16** эквивалентен запросу:

> *10 16 00 04 **3F FC** C0 00 5B 22*

Функция 0x18 (Read FIFO Queue)

Функция **0x18** используется для чтения очереди элементов, ассоциированных с указанным адресом holding–регистра. Соответственно, на уровне slave–устройства должна быть поддержка такой очереди для данного регистра (на практике таких устройств в настоящее время практически не выпускается).

В очереди может содержаться до 31 элемента, размер каждого элемента – 2 байта (т. е. регистр). Чтение значений из очереди не приводит к их удалению. Если при получении запроса slave–устройство обнаруживает, что очередь содержит более, чем 31 элемент, то в ответ отправляется код ошибки **0x03 (ILLEGAL DATA VALUE)**.

Запрос на чтение очереди регистра с адресом 50 (0x32) slave–устройства с адресом 16 (0x10) выглядит следующим образом:

> *10 18 00 32 05 36*

Предположим, в очереди содержится 3 элемента (которые занимают 6 байт) со значениями 1, 2 и 3. Тогда ответ будет выглядеть следующим образом:

> *10 18 00 08 00 03 00 01 00 02 00 03 53 9C*

где:

- 0x08 – количество байт данных далее, не включая байты контрольной суммы;
- 0x03 – количество событий в очереди;
- 0x0001 – значение первого элемента очереди;
- 0x0002 – значение второго элемента очереди;
- 0x0003 – значение третьего элемента очереди.

Функция 0x2B (Encapsulated Interface Transport)

Функция **0x2B** – одна из самых редко используемых и, пожалуй, самая специфичная функция протокола Modbus. Мы не будем рассматривать структуру её запросов и ответов – ограничимся обзорным описанием.

Эта функция разработана для передачи в рамках Modbus–пакетов запросов и ответов других протоколов. Такая «упаковка» пакетов одних протоколов в пакеты других называется «инкапсуляцией».

В рамках функции определено понятие подфункции – **MEI (MODBUS Encapsulated Interface)**.

Текущая версия спецификации определяет лишь две подфункции:

1. 0x0D (CANopen General Reference Request and Response PDU)

Эта подфункция используется для передачи в рамках Modbus–пакетов запросов и ответов протокола CANopen, и предназначена для работы со шлюзами CANopen / Modbus. Более подробная информация об этом приведена в спецификации CiA 309:

https://forum.opencyphal.org/uploads/short–url/mOjUw9o8JDWgbP0dNdnp2i8CHkB.pdf

Отметим, что данная спецификация выпущена в 2006 году.

В настоящий момент на рынке присутствует множество шлюзов, позволяющих использовать «стандартные» функции Modbus – такие, как **0x03 (Read Holding Registers)**, **0x10 (Write Multiple Registers)** и т. д.

2. 0x0E (Read Device Identification)

Эта подфункция позволяет считать информацию о slave–устройстве.

Спецификация определяет три обязательных информационных параметра:

- **VendorName** – название производителя устройства;
- **ProductCode** – идентификатор модели устройства;
- **MajorMinorRevision** – версию модели устройства.

Также указано 5 опциональных параметров, и кроме того, разработчик устройства может добавлять свои собственные параметры.

Допускается как чтение параметров по одному (individual access), так и группой (stream access).

Object Id	Object Name / Description	Type	M/O	category
0x00	VendorName	ASCII String	**Mandatory**	**Basic**
0x01	ProductCode	ASCII String	**Mandatory**	
0x02	MajorMinorRevision	ASCII String	**Mandatory**	
0x03	VendorUrl	ASCII String	Optional	**Regular**
0x04	ProductName	ASCII String	Optional	
0x05	ModelName	ASCII String	Optional	
0x06	UserApplicationName	ASCII String	Optional	
0x07 … 0x7F	*Reserved*		Optional	
0x80 … 0xFF	*Private objects may be **optionally** defined. The range [0x80 – 0xFF] is Product dependant.*	device dependant	Optional	**Extended**

В качестве примера прибора, который поддерживает эту функцию, можно назвать расходомер ВЫМПЕЛ–500, выпущенный НПО Вымпел:

https://vympel.group/products/flowmeters/vympel–500/

3 Идентификация прибора

Идентификация прибора (**Read Device Identification**) возможна следующими способами:

- чтение идентификационных данных из **input** регистров;
- вызов функции **Read Device Identification (0x2B/0x0E)** (в соответствии со спецификацией «**MODBUS Application Protocol Specification V1.1b3**»).

3.1 Основные параметры идентификации прибора

Основные параметры идентификации прибора (**Basic Device Identification (Mandatory)**) в соответствии с **Таблицей 3.1**.

Таблица 3.1

Запрос		Ответ	
Поле	**Значение**	**Поле**	**Значение**
Function	0x2B	Function	0x2B
MEI Type	0x0E	MEI Type	0x0E
Read Dev Id code	0x01	Read Dev Id Code	0x01
Object Id	0x00	Conformity Level	0x01
		More Follows	0x00
		Next Object Id	0x00
		Number Of Objects	0x03
		Object Id	0x00
		Object Length	0x0C
		Object Value	SPA "VYMPEL"
		Object Id	0x01
		Object Length	0x0E
		Object Value	GFC Vympel-500
		Object Id	0x02
		Object Length	0x01
		Object Value	4

Как указано в документации – те же самые значения можно считать из input–регистров, что, конечно, будет существенно проще использования функции 0x2B с подфункцией 0x0E.

«Пользовательские» функции Modbus

Спецификация Modbus указывает, что коды функций в диапазоне 65...72 (DEC) и 100...110 (DEC) являются «пользовательскими» (user–defined). Это означает, что производители оборудования могут реализовывать собственные функции. Обычно так поступают производители специфического измерительного оборудования с возможностью сохранения архивов.

Например, счётчики воды Протей и СВЭУ, выпускаемые компанией ООО «Сфера экономных технологий», поддерживают функции:

- 65 (0x41) – получение текущего значения одного или нескольких регистров по серийному номеру и специализированному адресу устройства 253 (0xFD);
- 66 (0x42) – установить новые значения одного регистра по серийному номеру и специализированному адресу устройства 253 (0xFD);
- 67 (0x43) – установить новые значения нескольких регистров по серийному номеру и специализированному адресу устройства 253 (0xFD);
- 68 (0x44) – чтение архивных значений;
- 69 (0x45) – чтение архивных значений по серийному номеру и специализированному адресу устройства 253 (0xFD).

Тепловычислители Взлёт, выпускаемые одноимённой компанией, поддерживают функции:

- 65 (0x41) – чтение архивов;
- 69 (0x45) – передача зашифрованных данных;
- 70 (0x46) – чтение контрольных данных;
- 71 (0x47) – выполнение команды.

Тепловычислители ТВ7, выпускаемые компанией Термотроник, поддерживают функцию:

- 72 (0x48) – расширенная запись и чтение регистров с нумерацией.

Можно было бы привести ещё существенное количество примеров устройств с поддержкой подобных функций – но, вероятно, перечисленных выше будет достаточно.

Отдельно стоит упомянуть про набор функций Modbus, разработанных для своих устройств компанией WirenBoard в рамках поддержки концепции «быстрого Modbus»:

https://wirenboard.com/wiki/Fast_Modbus

https://habr.com/ru/companies/wirenboard/articles/772308/

https://github.com/wirenboard/wb–modbus–ext–scanner/blob/main/docs/protocol.ru.md

7.2 Модели адресации регистров

> *Материал урока частично основан на п. 1.1 статьи [Заметки о Modbus](#).*

В течение всего курса мы опрашивали параметры различных slave–устройств, указывая их адреса регистров.

Если в документации slave–устройства для интересующего нас параметра был указан адрес регистра 2250, то при настройке опроса на стороне master–устройства мы вводили именно «2250».

Такая система адресации регистров называется «**физической**»; она проста и понятна.

Но в документации на некоторые slave–устройства и в некоторых программах, используемых для настройки обмена по Modbus, можно столкнуться с другой моделью адресации – **логической**.

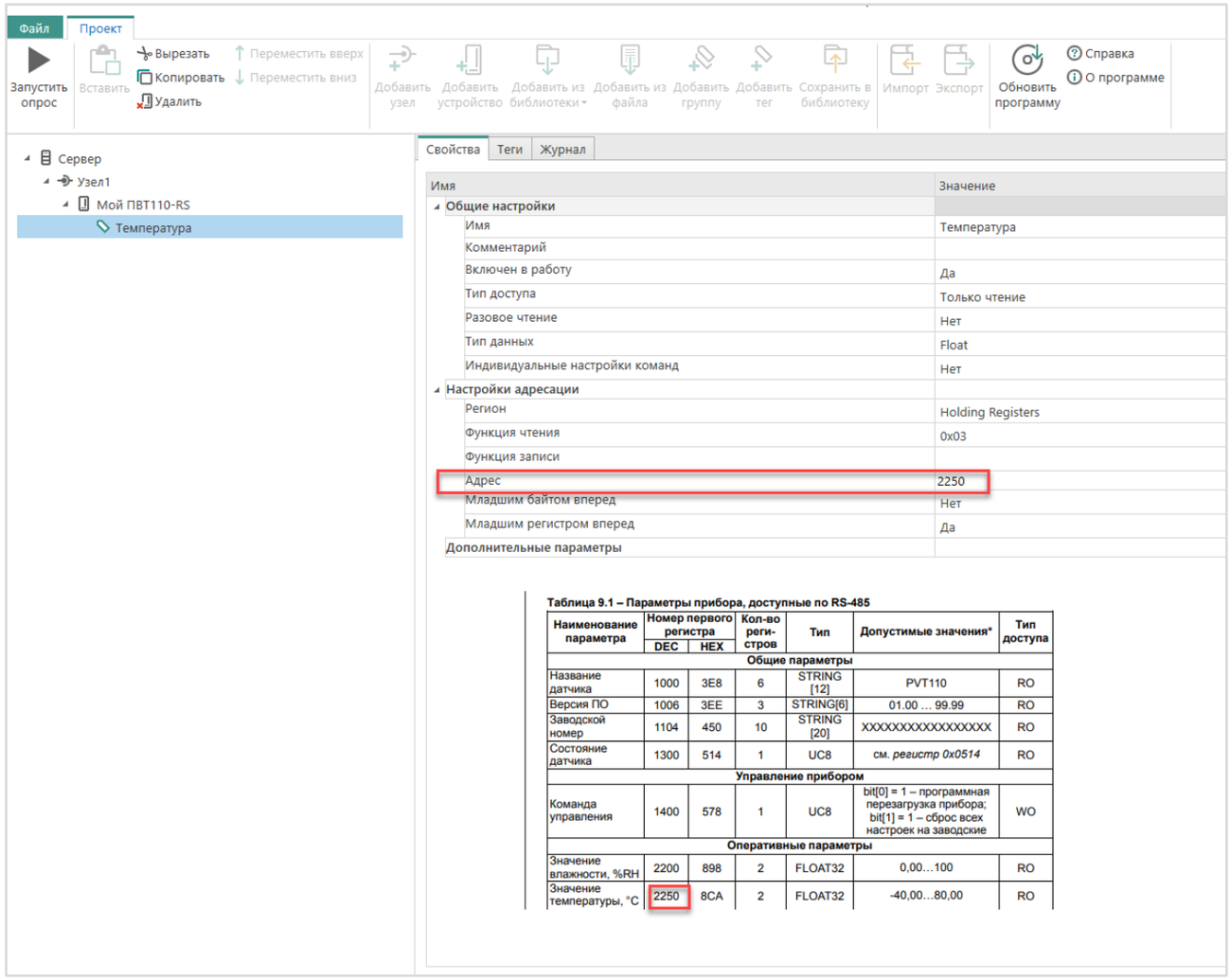
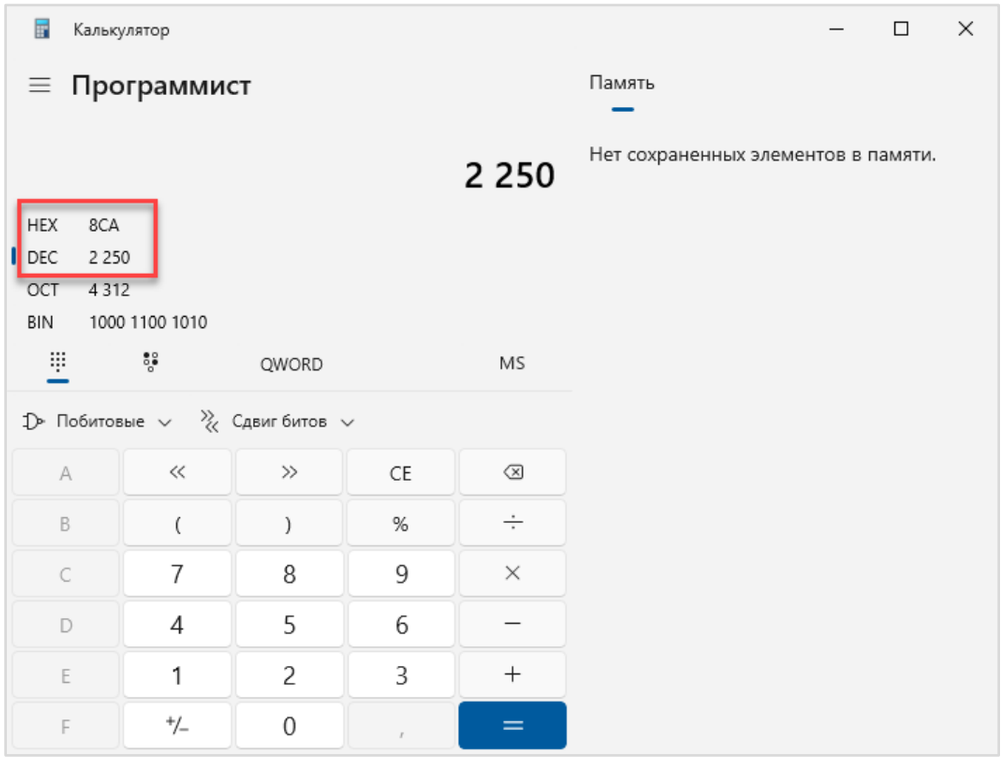


Таблица 9.1 – Параметры прибора, доступные по RS-485

Наименование параметра	Номер первого регистра		Кол-во регистров	Тип	Допустимые значения*	Тип доступа
	DEC	HEX				
Общие параметры						
Название датчика	1000	3E8	6	STRING [12]	PVT110	RO
Версия ПО	1006	3EE	3	STRING[6]	01.00 … 99.99	RO
Заводской номер	1104	450	10	STRING [20]	XXXXXXXXXXXXXXXX	RO
Состояние датчика	1300	514	1	UC8	см. *регистр 0x0514*	RO
Управление прибором						
Команда управления	1400	578	1	UC8	bit[0] = 1 – программная перезагрузка прибора; bit[1] = 1 – сброс всех настроек на заводские	WO
Оперативные параметры						
Значение влажности, %RH	2200	898	2	FLOAT32	0,00…100	RO
Значение температуры, °C	2250	8CA	2	FLOAT32	-40,00…80,00	RO

В редких случаях (например, при настройке опроса в облачном сервисе OwenCloud) адрес регистра нужно было указывать в шестнадцатеричной системе счисления (HEX); тогда вместо «2250» нужно было бы ввести «8CA».

Для большинства приборов ОВЕН в документации указан адрес регистра и в десятичной, и в шестнадцатеричной системе счисления; из рассмотренных нами устройств исключение составляет ТРМ210, для которого адреса были указаны только в HEX.



Логическая модель адресации. Часть 1

Знакомство с логической моделью адресации у многих инженеров происходит примерно по одному и тому же сценарию.

Представьте, что вам нужно опросить slave–устройство, с которым вы раньше не сталкивались – например, модуль аналогового ввода ZT–2017 от компании ICP DAS.

https://www.icpdas.com/web/product/download/wireless/zigbee/io/zt–2000/analog/zt–2017/document/manual/ZT–2017(C)_usermanual_en.pdf

Вы решаете начать со считывания значения нулевого аналогового входа (как видно на фото выше – входы и выходы у устройств ICP DAS нумеруются с нуля) и быстро находите в документации соответствующие регистры:

30001 ~ 30008	The analog input value for channels 0 to 7	R

У модуля 8 аналоговых входов, каждый из них занимает регистр. В документации указано, что для различных типов датчиков значения в этих регистрах будут принадлежать разному диапазону. Например, для датчика 4...20 мА регистр будет иметь значение из диапазона 0...65535, где 0 соответствует 4 мА, а 65535 – 20 мА.

В настройках master–устройства вы указываете адрес регистра **30001**. В документации указано, что модуль поддерживает обе функции чтения регистров – как **0x03 (Read Holding Registers)**, так и **0x04 (Read Input Registers)**.

Вы указываете функцию **0x03**, и в ответ на запрос модуль возвращает вам код ошибки **0x02 (ILLEGAL DATA ADDRESS)**.

Тогда вы выбираете функция **0x04**, но модуль опять возвращает ту же самую ошибку.

После этого вы, вероятно, решите действовать по одному из следующих сценариев:

- поищите информацию по вашей ситуации в интернете;
- обратитесь в техническую поддержку компании–дистрибьютора, у которой был приобретён данный модуль, или напрямую в ICP DAS;
- спросите совета у коллег;
- внимательно перечитаете руководство на модуль.

В конечном итоге вы выясните, что для чтения значения нулевого аналогового входа в настройках master–устройства нужно указать:

- функцию **0x04 (Read Input Registers)**;
- адрес регистра **0**.

И это даже описано в документации:

➢ Modbus RTU Command Format

Field 1	Field 2	Field 3	Field 4~n	Field n+1~n+2
Module Address	Function Code	Sub Function	Configuration Field	CRC16

Function Code	Description
0x04	Reads the input channels
0x46	Reads/writes the module settings

Examples:

A. To read the analog input value for module 01, the following command should be sent:
01 04 00 00 00 08 F1 CC

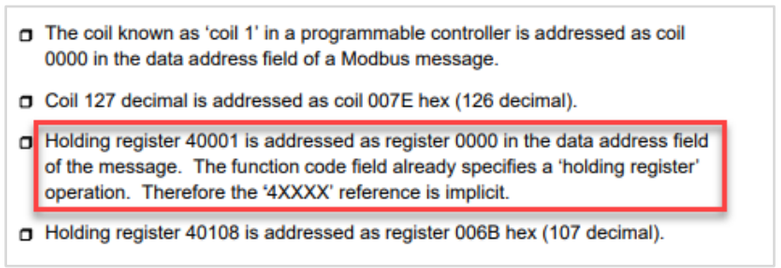
B. To read the name of the module, the following command should be sent:
01 46 00 12 60

У вас остаётся только один вопрос – почему же в карте регистров для интересующего вас параметра был указан адрес регистра **30001**, а в настройках master–устройства пришлось вводить адрес **0**?

Как вы уже догадались – дело в логической модели адресации, использованной в документации на модуль.

Логическая модель адресации. Часть 2

История логической модели адресации уходит своими корнями в старые версии спецификации протокола Modbus. См., например, раздел **Data Addresses in Modbus Messages** (стр. 18) в документе Modicon Modbus Protocol Reference Guide (1996):



- The coil known as 'coil 1' in a programmable controller is addressed as coil 0000 in the data address field of a Modbus message.
- Coil 127 decimal is addressed as coil 007E hex (126 decimal).
- Holding register 40001 is addressed as register 0000 in the data address field of the message. The function code field already specifies a 'holding register' operation. Therefore the '4XXXX' reference is implicit.
- Holding register 40108 is addressed as register 006B hex (107 decimal).

В рамках логической адресации адрес регистра представляет собой пятизначное число формата **Xyyyy**, в котором:

- **X** – идентификатор области памяти slave–устройства;
- **yyyy** – адрес регистра в диапазоне **1...9999**.

Мы неоднократно говорили про идентификаторы областей памяти Modbus, но давайте повторим их:

- 0x – coils (биты, доступны для чтения и записи);
- 1x – discrete inputs (биты, доступные только для чтения);
- 3x – input–регистры (доступны только для чтения);
- 4x – holding–регистры (доступны для чтения и записи).

Обратите внимание, что в рамках логической адресации нумерация регистров ведётся с **1**.

Соответственно, чтобы преобразовать адрес из логической модели в физическую, надо сделать следующее:

- по первой цифре адреса определить область памяти slave–устройства, в которой размещён данный параметр (и, соответственно, коды функций Modbus, используемые для доступа к нему);
- отбросить первую цифру адреса;
- из оставшегося 4–значного числа вычесть 1.

Примеры:

- логический адрес **00123** соответствует coil'у с физическим адресом **122**;
- логический адрес **10010** соответствует discrete input'у с физическим адресом **9**;
- логический адрес **30001** соответствует input–регистру с физическим адресом **0**;
- логический адрес **41400** соответствует holding–регистру с физическим адресом **1399**.

Самое важное, что следует запомнить – логическая адресация может встречаться в документации и графическом интерфейсе различных утилит. Но внутри Modbus–пакетов всегда используются физические адреса.

Логическая модель адресации. Часть 3

Как упоминалось в прошлом шаге – в рамках логической адресации каждая область памяти slave–устройства состоит максимум из **9999** объектов (потому что на адрес отводится только 4 цифры).

Поэтому некоторые производители используют логическую адресацию с 6–значными адресами. Например, её можно встретить в документации на тепловычислители Взлёт:

Таблица регистров хранения со значением в диапазоне целое 1 байт

Логический Modbus адрес	Название параметра	Тип	Режимы функционирования для доступа	Значения	Комментарии
400001	Modbus адрес при обмене с ПК, б/р	u16	(Работа), Сервис, Конфигурация, Настройка	1..247	Доступ на запись в режиме Работа задаётся рег. 400007
400002	Индекс скорости обмена с ПК, б/р	u16	(Работа), Сервис, Конфигурация, Настройка	0: 1200 бит/с 1: 2400 бит/с 2: 4800 бит/с	Доступ на запись в режиме Работа задаётся рег. 400007
400003	Задержка ответа при	u16	(Работа),	0..255	Доступ на запись в режиме Работа задаётся

Соответственно, логический адрес **400001** соответствует holding–регистру с адресом **0** и т. д.

В документации некоторых производителей адреса параметров указаны сразу для обоих моделей. Например, посмотрите, как это сделано в руководстве на датчик MSU44R от компании RAZUMDOM:

Рег	*Адрес*	*Диапазон данных*	*назначение*	*Модиф*
IR 0	30001	0 … 270	Напряжение питания (0.1В)	все
IR 11	30012	-32768…32767	Значение канала 1 (АЦП 12б *K/N+B)	Все
IR 12	30013	-32768…32767	Значение канала 2 (АЦП 12б *K/N+B)	все

В столбце **Рег** указаны физические адреса параметров, а в столбце **Адрес** – логические.

Логическая модель адресации. Часть 4

Теперь, увидев в документации адреса типа **30001** и **40001** – вы, возможно, вспомните про логическую модель адресации и интерпретируете их как «input–регистр с адресом 0» и «holding–регистр с адресом 0» соответственно.

Но не спешите – ведь в рамках физической модели адресации использование адресов **30001** и **40001** тоже является совершенно легитимным, и тогда в настройках вашего master–устройства придётся вводить именно **30001** и **40001**.

Поэтому настройка обмена с каждым конкретным slave–устройством требует тщательного изучения документации на это устройство.

Давайте используем полученные знания на практике.

В следующих шагах приведены фрагменты документации на различные приборы.

Вам нужно определить, какая модель адресации в них используется.

Вопрос 1. Электропривод ГЗ КС 15 от компании [ГЗ Электропривод]

Какая модель адресации используется в документации на этот прибор?

https://gz–privod.ru/wp–content/uploads/2021/02/mnogoob–obshheprom–ks–15–novyj–modbus–.pdf

Таблица 2. Адреса рабочих регистров необходимых для работы с модулем.

Адрес	Канал	Назначение	Атрибут	Примечание
Дискретные входные сигналы				
00001	0	Дискретный входной сигнал	R	Сигнал 13
00002	1	Дискретный входной сигнал	R	Сигнал 14
00003	2	Дискретный входной сигнал	R	Сигнал 15
00004	3	Дискретный входной сигнал	R	Сигнал 41
00005	4	Дискретный входной сигнал	R	Сигнал 42
Дискретные выходные сигналы				
00017	0	Дискретный выходной сигнал	R/W	Сигнал 5 «Открыть»
00018	1	Дискретный выходной сигнал	R/W	Сигнал 7 «Закрыть»
Аналоговые входные сигналы				
40001	0	Аналоговый входной сигнал	R	Сигнал 2 «Степень открытия»
Вспомогательные регистры				
40211		Версия программы	R	
40212		Статус сброса модуля	R	Сброс по COP
40213		Скорость RS-485	R/W	6-10 6 - 9600 7 – 19200 8 – 38400 9 – 57600 10 - 115200
40214		Действие при обрыве связи	R/W	0-2 0 – ничего не делать 1 – открыть 2 - закрыть
40215		Сброс флага COP	R/W	
40216		Флаг обрыва связи	R/W	
40217		Таймаут обрыва связи	R/W	0 - 1000мс
40218		Настройка бита четности RS-485	R/W	0-2 0 - нет 1 - дополнение до нечётности 2 - дополнение до чётности
40219		Количество стоповых бит RS-485	R/W	0 - 1 1 - 2

Пояснения к вопросу 1

Пятизначные адреса **00001** и **40001** намекают на логическую модель адресации.

Но смотрите:

00001 относится к дискретным входным сигналам, а **40001** – к аналоговым входным.

Они доступны только для чтения (на это указывает и значение R в столбце атрибут).

В рамках логической адресации эти параметры должны были бы иметь адреса **10001** и **30001**, чтобы находиться в областях discrete inputs и input–регистров соответственно (по крайней мере – извините за невольный каламбур – это было бы *логично*).

Поэтому перед нами физическая модель адресации.

Вопрос 2. Датчик ветра u[sonic] Modbus от компании LAMBRECHT meteo GmbH

Какая модель адресации используется в документации на этот прибор?

https://www.lambrecht.net/upload/manuals/usonic_Modbus_16470_Manual_4.pdf

The following measured values are provided by the LAMBRECHT meteo sensors.

Register address	Parameter name	Unit	Divisor	Quantity of registers	Access type
30001	Wind speed	m/s	10	1	Read only
30201	Wind direction	°	10	1	Read only

Example: Retrieving the wind speed

0D	04	75	31	00	01	7A	C5	0D	04	02	00	1F	E8	F9

LEN 6	Transmission Query =>	Source Master	Dest Slave 13	Function Read Input Register (4)	Func Desk Address=30001, Quantity of Register=1		Checksum OK:C57A
LEN 5	Transmission Response <=	Source Slave 13	Dest Master	Function Read Input Register (4)	Func Desk Byte count=2	Data 00 1F	Checksum OK:F9E8

Пояснение к вопросу 2

В документации указан адрес **30001**, который опять намекает на логическую модель, но смотрите – там же приведён пример Modbus–запроса, который открывает правду:

The following measured values are provided by the LAMBRECHT meteo sensors.

Register address	Parameter name	Unit	Divisor	Quantity of registers	Access type
30001	Wind speed	m/s	10	1	Read only
30201	Wind direction	°	10	1	Read only

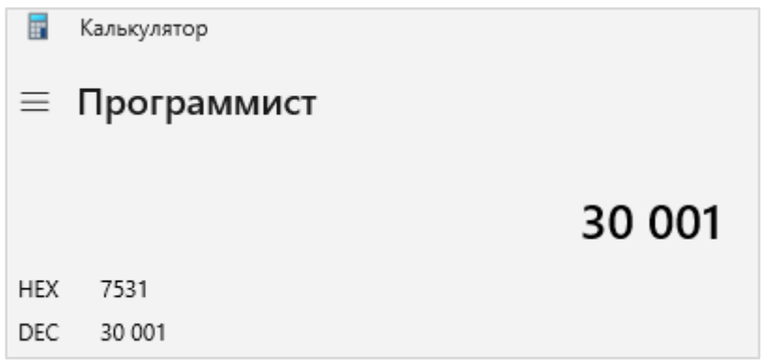
Example: Retrieving the wind speed

0D	04	75	31	00	01	7A	C5	0D	04	02	00	1F	E8	F9

LEN 6	Transmission Query =>	Source Master	Dest Slave 13	Function Read Input Register (4)	Func Desk Address=30001, Quantity of Register=1		Checksum OK:C57A
LEN 5	Transmission Response <=	Source Slave 13	Dest Master	Function Read Input Register (4)	Func Desk Byte count=2	Data 00 1F	Checksum OK:F9E8

0x7531 – это шестнадцатеричное представление числа **30001**.

И снова перед нами физическая модель адресации, затуманенная специфическим подходом разработчиков прибора к выбору адресов регистров.



Заключение

Возможно, вы сейчас пытаетесь понять – зачем разработчики приборов, используя в документации физическую модель адресации, выбирают такие странные адреса типа **30001** и **40001**, которые только путают пользователя?

Можно предположить, что они изучают документацию других производителей, в которой используется логическая адресация – но, не вникая в её суть, просто начинают считать, что **30001** и **40001** – это какие–то «общепринятые» для приборов с Modbus начальные адреса регистров (что, естественно, не соответствует истине).

Если вы хотите посмотреть, как выглядит документация, в которой используется логическая адресация – то напомним, что мы приводили такой пример в шаге 2 данного урока.

7.3 Примеры сомнительной реализации Modbus

Иногда встречаются slave–устройства, в которых протокол Modbus реализован с отступлениями от спецификации.

Часто это делает невозможным их опрос с помощью master–устройств, в которых Modbus реализован корректно.

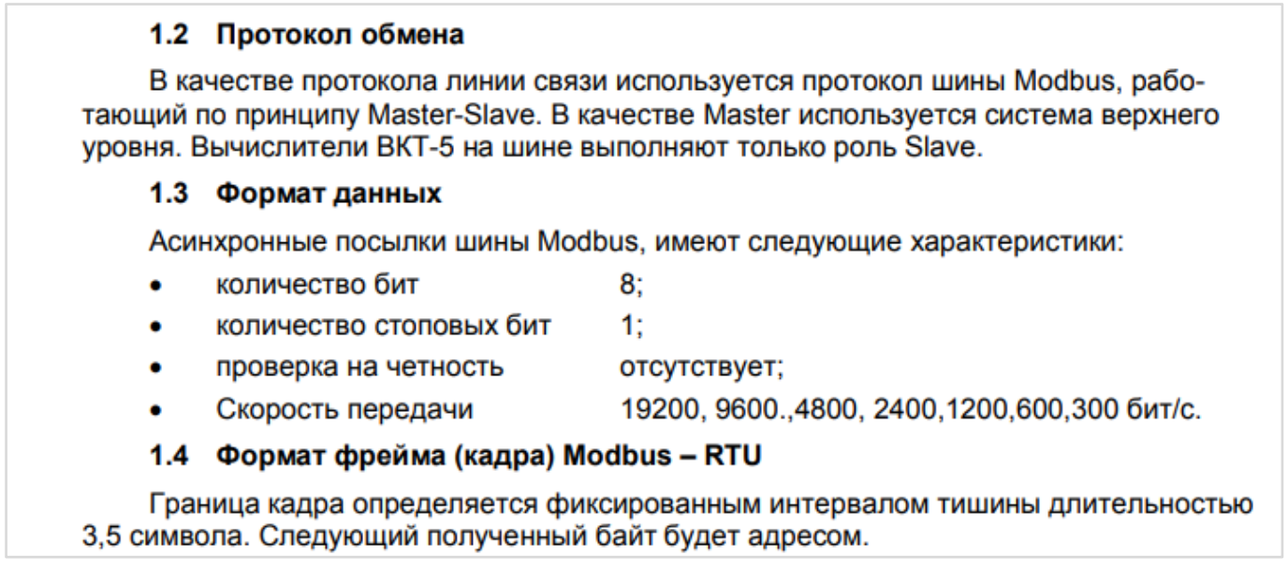
В следующих шагах приведены примеры некоторых подобных устройств.

Тепловычислители ВКТ–5 и ВКТ–7

Тепловычислители ВКТ–5 и ВКТ–7, выпускаемые компанией Теплоком, формально поддерживают протокол Modbus RTU.

По крайней мере, в документе Протокол связи вычислителя ВКТ–5 с системой верхнего уровня можно увидеть такую фразу:

> *В качестве протокола линии связи используется протокол шины Modbus, работающий по принципу Master–Slave.*



1.2 Протокол обмена

В качестве протокола линии связи используется протокол шины Modbus, работающий по принципу Master-Slave. В качестве Master используется система верхнего уровня. Вычислители ВКТ-5 на шине выполняют только роль Slave.

1.3 Формат данных

Асинхронные посылки шины Modbus, имеют следующие характеристики:

- количество бит 8;
- количество стоповых бит 1;
- проверка на четность отсутствует;
- Скорость передачи 19200, 9600.,4800, 2400,1200,600,300 бит/с.

1.4 Формат фрейма (кадра) Modbus – RTU

Граница кадра определяется фиксированным интервалом тишины длительностью 3,5 символа. Следующий полученный байт будет адресом.

Но при более внимательном изучении документа выясняется, что поле **Количество регистров в запросе** в большинстве случаев не соответствует реальному количеству передаваемых данных. Например, для записи настроек регуляторов требуется в поле **Количество регистров** передать значение **4**, а в следующем поле указать, что будет передано **16** байт данных.

Но 4 регистра – это 8 байт, а не 16. Ни один корректно спроектированный Modbus Master не позволит составить такой запрос.

3.2 Передача настроек регуляторов

Начиная с версии ПО 4 прибора появилась возможность получать и записывать некоторые настройки регуляторов.

3.2.1 Запись заданных значений для регуляторов

(Запрос реализован начиная с версии 4 ПО прибора)

Slave Address	сетевой адрес ВКТ-5	
Function	0×10	
Starting Address h	0×15	
Starting Address l	0	
No of Registers h	0	
No of Registers l	4	
Byte Count	16	
Значение дневной температуры для регулятора №1		тип float
Значение ночной температуры для регулятора №1		тип float
Значение дневной температуры для регулятора №2		тип float
Значение ночной температуры для регулятора №2		тип float
Error Check l		
Error Check h		

3.2.2 Ответ от ВКТ-5

(Ответ реализован начиная с версии 4 ПО прибора)

Slave Address	сетевой адрес ВКТ-5	
Function	0×10	
Starting Address h	0×15	
Starting Address l	0	
No of Registers h	0	
No of Registers l	4	
Error Check l		
Error Check h		

Конечно, запись настроек регуляторов – довольно редко используемая функция, но та же ситуация наблюдается и при чтении измеренных прибором значений – например, в запросе будет указано **Количество считываемых регистров** = **4**, а в ответ ВКТ–5 отправит **8** регистров.

Это связано с тем, что в данном тепловычислителе поле **Количество регистров** трактуется разработчиками как К**оличество параметров**, а параметр может занимать несколько регистров (например, параметр типа Float занимает 2 регистра).

В документации на ВКТ–7 это указано в явном виде. Строго говоря, в документации на ВКТ–7 нигде не написано, что тепловычислитель поддерживает именно протокол Modbus. Косвенно можно понять, что речь идёт о «Modbus–подобном» протоколе.

2.1 Отступления от требований стандарта Modbus в протоколе в ВКТ-7

- Все поля в протокольных кадрах, имеющие длину более 1-го байта, должны передаваться в следующем порядке: сначала младший байт, затем старший. Это требование не относится к полям «**Начальный адрес**» **и «Количество регистров**». Они должны передаваться в следующем порядке: сначала старший байт, затем младший байт;
- Поле «**Количество регистров**» не анализируется прибором при обработке запроса и **может быть** установлено в произвольное значение, если иное не оговорено специально при описании запроса;
- Поле «**Количество записываемых байт данных**» может содержать значение, не соответствующее реально записываемому количеству байт данных. Например, в запросах «**Запись состояний дискретных выходов**» и «**Начало сеанса связи**»;
- Граница кадра определяется фиксированным интервалом тишины длительностью 62.5 мс или по переполнению входного буфера длиной 264 байта.

Интересно, что в ВКТ–7 для некоторых запросов поле **Количество регистров** должно быть установлено в значение 0.

Опять же, ни один корректно спроектированный Modbus Master не позволит составить такой запрос.

4.4 Запрос на запись даты

Запрос предназначен для того, чтобы указать вычислителю, относительно какой хронологической метки следует передавать архивные данные. Запрос содержит хронологическую метку с указанием даты и времени. Вычислитель передает архивные данные в ответ на запрос «Чтение данных…» относительно той хронологической метки, которая была записана в вычислитель в последний раз. При записи даты для чтения суточного или месячного архива поле «час» должно быть установлено в значение 23.

Начальный адрес = **0x3FFB**;

Количество регистров = **0x0000**.

Дата передается в формате VT_DATA_RAP (описание в Приложении Б).

Пример:

```
Frame [0 ] = 0x00;
Frame [1 ] = 0x10;
Frame [2 ] = 0x3F;
Frame [3 ] = 0xFB;
Frame [4 ] = 0x00;
Frame [5 ] = 0x00;
Frame [6 ] = 0x04;
Frame [7 ] = день;
Frame [8 ] = месяц;
Frame [9 ] = год;
Frame [10] = час;
Frame [11] = CheckSum l;
Frame [12] = CheckSum h;
```

Готовый вариант запроса: (30 января 2003г. 0 часов)

```
0x00 0x10 0x3f 0xfb 0x00 0x00 0x04 0x1e 0x01 0x03 0x00 0xfa 0xaf
```

Интересно, что в тепловычислителе ВКТ–9, выпускаемом той же компанией, реализован «настоящий» Modbus RTU.

Две истории от одного клиента

Однажды к нам обратился клиент, который пытался настроить обмен с индикаторным табло, выпущенным некой российской компанией. Табло поддерживало только протокол Modbus ASCII (что уже является отступлением от спецификации Modbus: согласно ей, поддержка Modbus RTU является обязательной, а поддержка Modbus ASCII – опциональной).

Табло никак не реагировало на корректный запрос.

После изучения примера, предоставленного производителем, выяснилось, что если в отправляемом запросе присутствуют стоп–символы (а согласно спецификации Modbus ASCII – они должны присутствовать), то табло игнорирует этот запрос.

Если отправить запрос без стоп–символов – то табло обрабатывает его и отображает присланную строку, но при этом не отправляет ответа master–устройству, что также является отступлением от спецификации Modbus.

Насколько нам известно – клиент проинформировал производителя об этих проблемах, и в следующей версии прошивки устройства они были устранены.

***

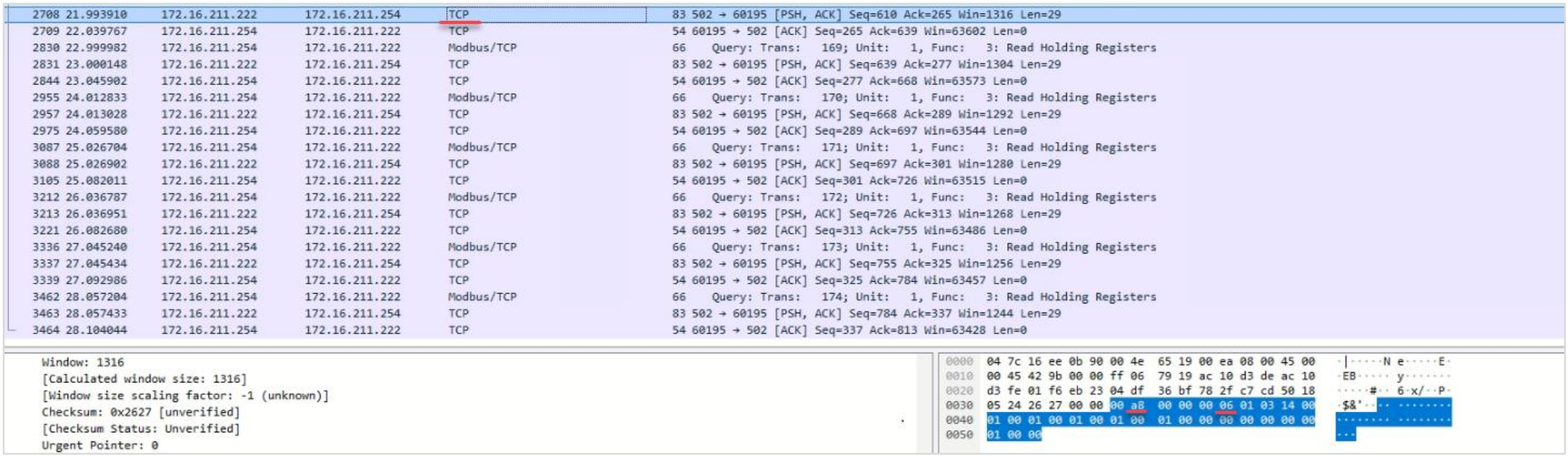
Спустя некоторое время тот же самый клиент обратился по поводу настройки обмена с другим прибором – блоком управления освещением, выпущенным некой российской компанией. Обмен с блоком происходил по протоколу Modbus TCP.

Блоку отправлялся запрос на чтение параметров:

> *00 A8 00 00* ***00 06*** *01 03 00 00 00 0A*

И блок отправлял следующий ответ:

> *00 A8 00 00* ***00 06*** *01 03 14 00 01 00 01 00 01 00 01 00 01 00 00 00 00 00 00 00 00 01 00 00*



Выделенное жирным значение соответствует полю **Length** (*«число байт далее»*) заголовка Modbus TCP–пакета (MBAP Header).

В запросе оно является корректным – за ним действительно расположены 6 байт.

А вот в ответе с ним что–то не так – за этим полем размещены 23 байта данных, а оно имеет значение 6.

Очевидно, что slave–устройство при подготовке ответа копирует MBAP Header из запроса. Это разумно для полей **Transaction ID** и **Protocol Identifier**, но совершенно некорректно для поля **Length**, которое должно быть рассчитано на основании реального количества байт ответа.

Соответственно, master–устройство, которое производит проверку этого поля, посчитает данный ответ некорректным и не будет извлекать из него данные. Контроллер клиента производил такую проверку.

Насколько нам известно – разработчики блока не признали существование проблемы; клиенту пришлось собственноручно реализовать в контроллере отдельный Modbus TCP Master, который не проверял в ответе значение поля **Length**.

Россыпь мелких историй про Modbus TCP

Один из клиентов рассказал нам о встретившемся ему slave–устройстве, которое не отвечало на запрос, если значение поля **Transaction ID** в нём превышало **255**.

***

Master–устройство другого клиента всегда использовало в запросах для поля **Unit ID** значение **0**. Это не является нарушением спецификации Modbus TCP; но slave–устройство клиента (ПЛК ОВЕН) не позволяло принимать запросы с таким **Unit ID**. Ошибка на стороне slave–устройства через некоторое время была исправлена, но её можно было бы обойти сразу, если бы master–устройство использовало в качестве **Unit ID** значение **255** (которое является первым из рекомендуемых в спецификации) или позволяло бы пользователю задать его самостоятельно.

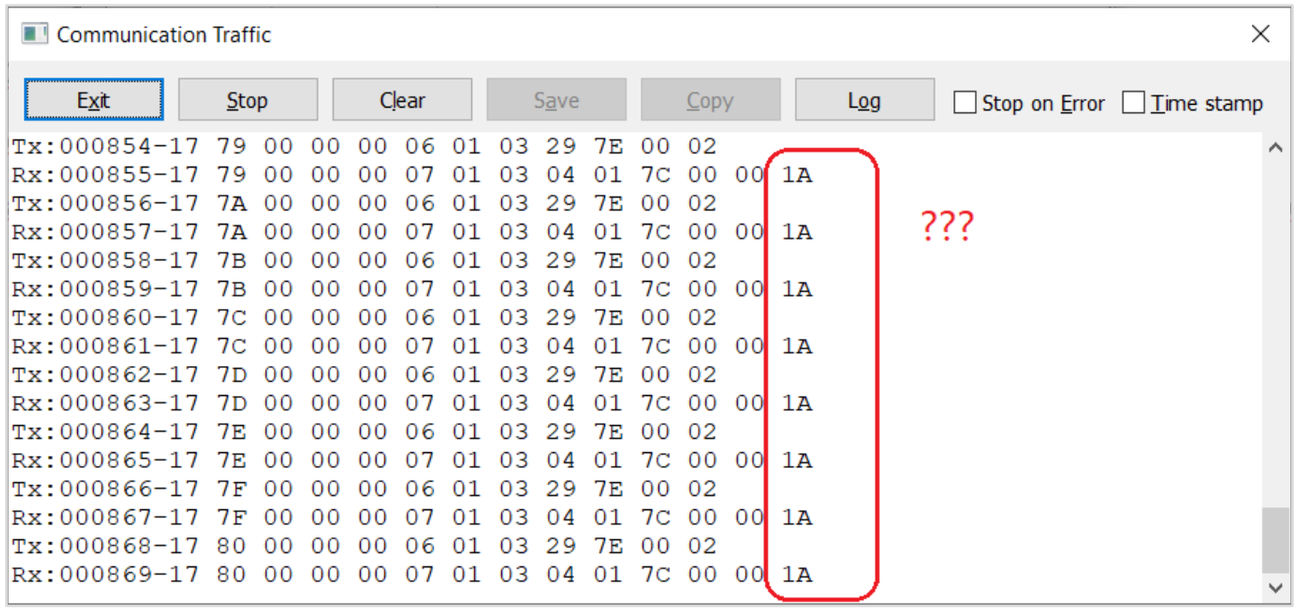
***

Ещё один клиент столкнулся со slave–устройством, в котором в состав пакета Modbus TCP входила контрольная сумма (как в Modbus RTU). Соответственно, master–устройство клиента не могло сформировать такой запрос, потому что согласно спецификации протокола Modbus TCP – в его пакетах нет поля контрольный суммы.

Запрос вида							
ID Транзакции	ID протокола	Длина запроса не изменять	Адрес	КОД_ОПЕР	Адрес Начального регистра	Регистров далее	CRC16
0001	0000	0006	11	03	0000	0038	406d
Ответ вида							

***

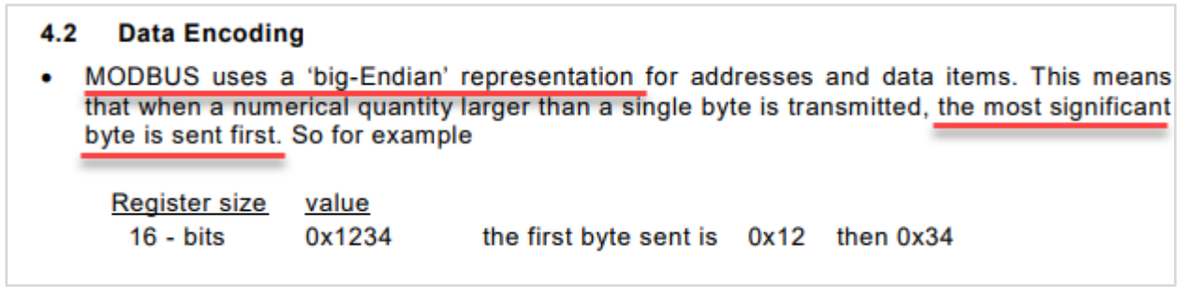
Несколько раз мы слышали рассказы о slave–устройствах, которые добавляют в конец пакета лишний байт.



KUNKIN KL7206

Прибор для тестирования характеристик электронных компонентов KL7206, выпускаемый компанией KUNKIN, в своих ответах передаёт контрольную сумму с порядком «старшим байтом вперёд».

Действительно, в MODBUS APPLICATION PROTOCOL SPECIFICATION V1.1b3 (п. 4.2) указан именно такой порядок байт



4.2 Data Encoding

- MODBUS uses a 'big-Endian' representation for addresses and data items. This means that when a numerical quantity larger than a single byte is transmitted, the most significant byte is sent first. So for example

Register size	value			
16 - bits	0x1234	the first byte sent is	0x12	then 0x34

Но в этом документе не обсуждается принцип формирования контрольной суммы – он описан в MODBUS over serial line specification and implementation guide V1.02 (п. 6.2.2), и там уточняется, что контрольная сумма передаётся **младшим** байтом вперёд.

In the CRC 16, the 1st byte transmitted is the least significant one.

Отсутствие в первом документе примечания об особенностях порядка байт для контрольной суммы и невнимательность разработчиков прибора к деталям спецификации привели к тому, что его невозможно опросить с помощью корректно реализованного master–устройства, которое проверяет контрольную сумму ответа.

Ваши истории

Возможно, вы тоже встречались с приборами, в которых Modbus был реализован с отступлениями от спецификации – и это, вероятно, создало вам немало проблем.

Если так – то напишите, пожалуйста, историю об этом в комментариях.

7.4 Как разработать прибор с удобной реализацией Modbus?

Мы завершим наш курс серией советов о том, как сделать поддержку протокола Modbus в вашем приборе удобной для пользователей.

Эти советы являются субъективными.

Их стоит воспринимать не в качестве некой декларируемой догмы, а как дополнительную информацию для ваших собственных рассуждений.

Интерфейс RS–485

Если вы планируете оснастить ваш прибор интерфейсом RS–485, то стоит обдумать следующие моменты:

- используемая микросхема UART: тут можно только порекомендовать не экономить на ней и использовать промышленный вариант микросхемы от проверенного вендора;
- тип разъёма: с точки зрения монтажа наиболее удобным вариантом являются съёмные клеммные колодки или быстрозажимные / самозажимные клеммы. Разъёмы типа DB–9 или RJ–45/RJ–12 являются существенно менее удобными. Проверьте, насколько просто завести в одну клемму вашего разъёма два провода типового промышленного кабеля (потому что будет использоваться топология «шина» – ваш прибор будет соединён с предыдущим и последующим подключёнными к ней устройствами);
- индикатор обмена: спецификация Modbus требует у прибора наличие индикатора, который будет мигать в момент получения и отправки пакета (допускается использовать два отдельных индикатора). Рекомендуемый цвет индикатора – жёлтый.

Можно порекомендовать использовать трёхцветный индикатор:

- зелёный цвет будет соответствовать приёму корректного запроса, адресованного данному slave–устройству. На такой запрос будет отправлен ответ (за исключением broadcast–запроса);
- жёлтый цвет будет соответствовать приёму запроса, адресованного другому slave–устройству. На такой запрос не будет отправлен ответ;
- красный цвет будет соответствовать приёму некорректного запроса, адресованного данному slave–устройству. На такой запрос будет отправлен ответ с кодом ошибки Modbus или же (например, в случае ошибки CRC или некорректной структуры пакета) не будет отправлен ответ.

Наличие жёлтого цвета позволяет отследить сам факт передачи по шине каких–либо данных. Это полезно на этапе отладки, но вы можете предусмотреть возможность такой индикации в «рабочем» режиме. Если ваш прибор подразумевает возможность установки в помещении (например, в жилой комнате) – то будет полезным предусмотреть возможность отключения индикации или её включения на заданный интервал времени после запуска прибора (чтобы не раздражать конечного пользователя).

Подумайте о том, потребуется ли в вашем приборе:

- гальваническая изоляция интерфейса RS–485;
- клемма для подключения «общего провода»;
- встроенные согласующие резисторы (терминаторы) и подтягивающие резисторы;
- DIP– и поворотные переключатели для изменения сетевых настроек (скорости, режима контроля чётности, адреса и т. д.).

Функции Modbus и модель памяти

Используйте «общую» модель памяти, в рамках которой все регистры прибора одновременно являются и holding–регистрами, и input–регистрами – и, соответственно, доступны для чтения как функцией **0x03 (Read Holding Registers)**, так и **0x04 (Read Input Registers)**. Соответственно, какую бы функцию не выбрал пользователь в настройках своего master–устройства – он не ошибется.

Для записи значений нужно обязательно поддержать как функцию **0x10 (Write Multiple Registers)**, так и функцию **0x06 (Write Single Register)**. При этом для параметров прибора, занимающих один регистр – запись должна поддерживаться с помощью любой из этих функций. Это упростит настройку обмена в тех случаях, когда master–устройством является панель оператора, OPC–сервер или другое ПО, выбор функции записи в котором осуществляется не всегда очевидным для пользователя образом.

Не стоит поддерживать «битовые» функции Modbus. Вместо этого группируйте биты в регистры, используя битовые маски.

Исключение может быть сделано для сохранения обратной совместимости. При этом у пользователя всё равно должна быть возможность обратиться к тем же самым значениям с помощью «регистровых» функций (т. е. битовые функции должны дополнять регистровые, а не замещать их).

Нет особого смысла предоставлять пользователю какие–либо другие функции Modbus (но вы можете сделать это для своих собственных целей, если считаете полезным – например, для вашего конфигурационного ПО).

Если ваш прибор ведёт архивы – то разместите их данные в регистрах и предоставьте к ним доступ с помощью функций **0x03** и **0x04**, а не используйте функцию **0x14 (Read File Record)**, которая поддерживается крайне ограниченным числом master–устройств.

> *Поскольку архивы могут занимать много места, а количество регистров slave–устройства ограничено – вы можете выделить область регистров для «выбранных архивных записей» и создать ряд «управляющих» регистров – например, тип архива (часовой/суточный/месячный и т. д.), начальную метку времени архива, количество выгружаемых записей и т. д., которые будут определять содержимое этой области в конкретный момент времени.*

Аналогично и с информационными параметрами (версией прошивки, заводским номером и т. д.) и параметрами диагностиками – лучше разместить их в регистрах, чем поддерживать функции **0x07**, **0x08**, **0x0B**, **0x0C**, **0x11** и **0x2B** с подфункцией **0x0E**.

Запись параметров

1. Не создавайте параметры с типом доступа «только запись». Некоторые master–устройства автоматически выполняют чтение регистра после его записи – и в случае отсутствия ответа будут диагностировать ошибку, которая может тревожить пользователя.

2. Если параметр доступен для записи, то он должен быть доступен для записи **всегда**. Например, в некоторых регуляторах запись некоторых настроек возможна только в том случае, если для регулятора установлен определённый режим работы. В других режимах попытка записи этих параметров приведёт к ошибке **0x02 (ILLEGAL DATA ADDRESS)**. Логику разработчиков можно понять, но это совершенно непрозрачно для пользователя. Возможно, он хотел бы сначала записать значения этих настроек, и только потом изменить режим работы регулятора. Если он запишет значения настроек и не увидит изменений в поведении регулятора – то, вероятно, решит проверить другие его настройки (в том числе, и режим работы). Естественно, такие особенности должны быть тщательно задокументированы.

3. Проработайте механизм записи энергонезависимых параметров во flash–память. Мы обсуждали связанные с этим нюансы в уроке 2.6.

Поддержка групповых запросов

Для обеспечения возможности опроса с помощью групповых запросов – адреса регистров устройства должны быть распределены последовательно и не иметь «разрывов». Если вы хотите оставить «запас» адресов для потенциальных будущих функций прибора – то сразу создайте параметры с соответствующими адресами и перечислите их в карте регистров, дав им названия типа «Резерв». Таким образом, они не помешают использованию групповых запросов. Эти «резервные» параметры должны иметь значение **0** или, например, специфическое значение, описанное в документации и индицирующее их «резервность» (например, **0xFFFF**).

Располагайте параметры, поддерживающие функции записи, рядом (на соседних адресах регистров), чтобы обеспечить возможность их записи одним групповым запросом.

По возможности избегайте искусственных ограничений, связанных с групповыми запросами.

Пример такого ограничения из документации преобразователя частоты EKF Vector 100:

> **Основные команды**
>
> Код команды: 03H, чтение N слов/регистров (до 12 слов)
>
> Код команды: 06H, запись одного слова/регистра

Типы данных параметров

Выбирайте для параметров «естественные» типы данных.

Например, для значения с плавающей точкой разумно выбрать тип **Float** вместо того, чтобы представить его целочисленным значением со смещением десятичной точки, которое после считывания потребуется разделить, например, на 100.0.

Для метки времени имеет смысл использовать широко распространённый формат Unixtime, в котором время выражается в виде количества секунд, прошедших с полуночи 1 января 1970 года; используйте в качестве точки отсчёта именно 1970 год, а не 1990–й, 2000–й или какой–то ещё.

Другим разумным решением является представление метки времени в виде 6 отдельных регистров, каждый из которых содержит целочисленное значение одного из разрядов времени. Избегайте «упаковки» двух разрядов в один регистр и использования формата BCD – это сэкономит незначительное количество памяти, но создаст сложности пользователю, которому придётся преобразовывать всё это на стороне master–устройства.

Порядок байт/регистров

В ходе курса при настройке опроса нам не раз приходилось указывать порядок регистров. Напомним, это связано с тем, что спецификация протокола Modbus не определяет порядок регистров при передачи параметров, занимающих два и более регистра.

Если вы поддержите возможность настройки порядка регистров на стороне slave–устройства – это будет очень удобно для конечного пользователя.

С этим связан ряд нюансов:

- наиболее важной является настройка порядка регистров. Некоторые устройства поддерживают и настройку порядка байт, хотя он определён в спецификации Modbus; нужно ли реализовывать настройку порядка байт для вашего конкретного устройства – решать вам;
- настройка порядка байт/регистров должна примениться только к полю данных пакета. Она не должна влиять на служебные поля – такие, как адрес регистра, количество регистров и контрольная сумма;
- вы можете распространить влияние этой настройки лишь на некоторые (наиболее важные) параметры вашего прибора. Например, в модулях ОВЕН МВ110–2А Н/W 2.0 и МВ110–8А Н/W 2.0 данная настройка влияет только на значения аналоговых каналов модуля:

Порядок байт для оперативных FLOAT	0 – не менять; 1 – инверсия байтов; ***2 – инверсия регистров;*** 3 – инверсия байтов и регистров;	Uint16	0x0BB8	3000

Если вы разрабатываете конфигурационное ПО для вашего прибора – то нужно учесть, что прибор имеет настройку порядка байт/регистров. Есть два основных варианта:

1. При подключении к прибору конфигурационное ПО в первую очередь считывает регистр, содержащий настройку порядка байт/регистров прибора, после чего применяет её для корректного отображения параметров;

2. Для подключения к прибору с помощью конфигурационного ПО используется отдельный интерфейс (например, USB), и настройка порядка байт/регистров не применяется к запросам, поступающим по этому интерфейсу.

Modbus TCP

При разработке slave–устройства с поддержкой протокола Modbus TCP стоит учесть следующие моменты:

- устройство должно отвечать на запросы с Unit ID = **0** и **255** (это следует из спецификации Modbus TCP), а также **1** (это исторически общепринятый адрес, которые используют в запросах многие master–устройства; некоторые из них не позволяют его изменить). Вполне допустимо, чтобы устройство отвечало на запрос с любым Unit ID;
- разумно поддержать возможность изменения номера TCP–порта. По умолчанию, как указывает спецификация Modbus TCP, должен использоваться порт **502**. Но в сетях некоторых предприятий этот порт запрещён к использованию как «общеизвестный» и, соответственно, являющийся потенциальной целью кибератак;
- полезно добавить регистр, который будет отображать количество текущих установленных соединений с данным устройством. Это упрощает диагностику (например, чтобы определить, что к вашему slave–устройству случайно или намеренно подключился кто–то ещё и тем самым исчерпал лимит доступных соединений). Максимальное количество одновременных соединений должно быть указано в документации на устройство.

Если вы разрабатываете master–устройство с поддержкой протокола Modbus TCP, то не забудьте учесть возможность его использования совместно со шлюзами Modbus TCP/Modbus Serial. Для этого устройство должно уметь отправлять в рамках одного TCP–соединения запросы с разными Unit ID.

Специфические функции

Возможно, вы посчитаете, что вашему устройству нужна поддержка специфических функций.

Одним их примеров таких функций является режим **Modbus Spy,** который мы упоминали в уроке 2.12.

Этот режим позволяет интегрировать устройство в уже эксплуатирующиеся системы управления и «подслушивать» данные, передаваемые по последовательной линии связи. Одновременно с этим у устройства появляется возможность выделять «свои» данные из широковещательного запроса. Это, например, полезно, если вы хотите синхронно обновить значения для группы устройств (например, одновременно переключить выходы всех модулей, подключённых к данной линии связи).

Другим примером является так называемый «**пользовательский modbus mapping**» – механизм, позволяющий клиенту самостоятельно задавать адреса регистров для параметров прибора. Это позволяет:

- разместить адреса интересующих пользователя параметров так, чтобы обеспечить возможность их чтения/записи как можно меньшим количеством групповых запросов;
- «подстроить» карту регистров прибора под другой прибор (например, для замены вышедшего из строя аналогичного прибора другого производителя).

При реализации данного механизма рекомендуется:

- не менять «заводские» адреса регистров, а «дублировать» параметры (т. е. один и тот же параметр будет доступен по двум адресам – «заводскому» и «пользовательскому»). Это, например, позволит при любых обстоятельствах подключиться к прибору с помощью его конфигурационного ПО;
- валидировать задаваемые пользователем адреса регистров и предоставить средства диагностики (в частности, отображать информацию о том, что заданные «пользовательские» адреса пересекаются с «заводскими» или сами с собой).

> *См. также статью Протокол Modbus: Настраиваемая карта регистров (Маппинг) и Best Practices разработки от Электрошамана (Cs–Cs).*

Документация

При настройке опроса вашего прибора – пользователь будет ориентироваться на документацию; от её качества зависит, насколько быстро у него получится это сделать.

Перечислите в документации:

- поддерживаемые функции Modbus;
- используемую модель памяти;
- порядок байт/регистров;
- поддерживается ли широковещательный запрос;
- механизм энергонезависимости параметров, упомянутых в карте регистров (например, если вы записываете новое значение настройки – будет ли оно сразу сохранено во flash–память прибора?);
- заводские настройки коммуникационных интерфейсов;
- все особенности и ограничения, связанные с поддержкой Modbus в вашем приборе (например, для прибора с Modbus TCP – количество одновременных клиентских подключений).

В карте регистров используйте **физическую модель адресации** и указывайте для каждого параметра:

- идентификатор или порядковый номер;
- адрес начального регистра в DEC и HEX;
- количество занимаемых регистров;
- тип доступа (только чтение или чтение/запись);
- тип данных. В случае использования специфических, не «общепринятых» типов – разместите их описание перед картой регистров;
- диапазон значений. Для перечислений и битовых масок – укажите все возможные значения параметра;
- значение по умолчанию;
- единицы измерения;
- дополнительную информацию о параметре и его влиянии на работу прибора. По возможности полезно сделать название параметра гиперссылкой на раздел документа, в котором описывается его назначение.

Укажите в документации, какие коды ошибок Modbus может возвращать ваш прибор и в каких именно случаях это происходит (например, приборы разных производителей возвращают код ошибки **0x04 SERVER DEVICE FAILURE** при совершенно различных обстоятельствах).

Для прибора с интерфейсом RS–485 опишите в документации аппаратные особенности:

- наличие и характеристики гальванической изоляции;
- наличие подтягивающих и/или терминирующих резисторов;
- unit load (электрическая «нагрузка» прибора на последовательную линию связи);
- предполагаемое время ответа (это позволит пользователю разумно настроить таймаут на стороне master–устройства);
- и т. д.

8. Заключение

8.1 Заключение

На этом наш курс закончен.

Хочется надеяться, что он был для вас полезным и интересным.

Мы будем признательны, если вы оставите свой отзыв на странице курса.

Такие отзывы обычно носят довольно ёмкий и обзорный характер.

Если у вас есть развёрнутые пожелания по доработке данного курса – то, пожалуйста, оставьте их в комментарии к этому уроку.

Желаем удачи в настройке обмена по протоколу Modbus с имеющимися у вас устройствами!

9. Ответы на тестовые задания

2.4 Тестовые задания

1. Как расшифровывается название протокола Modbus
   - ***Modicon bus***
   - Module bus
   - Modern bus
   - Modified bus

2. Укажите исторический период, в котором был создан протокол Modbus.
   - 1990-е - 2000-е
   - 2010-е - 2020-е
   - ***1970-е - 1980-е***

3. Какие протоколы определены в спецификации Modbus?
   - Modbus UDP
   - ***Modbus TCP***
   - Modbus RTU over TCP
   - Modbus RTPS
   - ***Modbus ASCII***
   - ***Modbus RTU***

4. Какой из документов описывает особенности физического уровня протоколов Modbus RTU и Modbus ASCII?
   - MODBUS Messaging on TCP/IP Implementation Guide
   - MODBUS APPLICATION PROTOCOL SPECIFICATION
   - ***MODBUS over Serial Line. Specification and Implementation Guide***

2.7 Тестовые задания

1. На какой архитектуре основан протокол Modbus?
   - Peer-to-peer
   - ***Master/Slave (Client/Server)***
   - Publisher/Subscriber
   - Manager/Agent

2. Сколько master–устройств может быть подключено к последовательной линии связи?
   - 255
   - 247
   - ***1***
   - 2

3. Какие типы данных определены в спецификации Modbus?
   - Uint32
   - Int32
   - ***Бит***
   - Float
   - String
   - Uint16
   - Int16
   - ***Регистр***

4. Укажите размер регистра в битах.
   - ***16***

5. Вставьте пропущенную фразу.

"Спецификация Modbus определяет <....> при передаче данных".
   - Порядок регистров
   - Порядок байт и регистров
   - Типы данных
   - ***Порядок байт***

6. Сопоставьте функцию Modbus с её категорией.

0x03 (Read Holding Registers)	**Работа с регистрами**
0x05 (Write Single Coil)	**Работа с битами**
0x14 (Read File Record)	**Работа с файлами**
0x07 (Read Exception Status)	**Диагностика**

7. Укажите максимальное количество объектов (битов или регистров), которое может быть размещено в одной области памяти slave–устройства.
   - ***65536***

2.10 Тестовые задания

1. Сопоставьте параметр и его значение.

Максимальный размер пакета Modbus RTU в байтах	**256**
Максимальное количество считываемых регистров для функций 0x3 и 0x04	**125**
Максимальное количество записываемых регистров для функции 0x10	**123**
Количество бит в байте	**8**

2. Рассмотрим следующий пакет протокола Modbus RTU:

> *4D 04 14 00 02 00 00 04 00 06 00 08 00 0A 00 0A 08 00 06 00 04 00 02 09 DA*

Это запрос или ответ?

- Запрос
- ***Ответ***
- Может быть как запросом, так и ответом

3. Рассмотрим следующий пакет протокола Modbus RTU:

> *4D 04 14 00 02 00 00 04 00 06 00 08 00 0A 00 0A 08 00 06 00 04 00 02 09 DA*

Соотнесите поля пакета и соответствующие им байты.

Адрес slave–устройства	**4D**
Код функции	**04**
Количество байт данных	**14**
Данные	**00 02 00 00 04 00 06 00 08 00 0A 00 0A 08 00 06 00 04 00 02**
Контрольная сумма	**09 DA**

4. Рассмотрим следующий пакет протокола Modbus RTU:

> *4D 04 14 00 02 00 00 04 00 06 00 08 00 0A 00 0A 08 00 06 00 04 00 02 09 DA*

Предположим, что первый из считываемых регистров имеет адрес **100** (в DEC).
Введите запрос, на который был отправлен данный ответ (в HEX, с пробелами между байтами).

> ***Примечание**: вы можете использовать Rapid SCADA Modbus Parser для конструирования запроса.*

4D 04 00 64 00 0A 3F DE

5. *Рассмотрим следующий пакет протокола Modbus RTU:*

> *4D 10 00 64 00 01 02 FF FF CB 07*

Предположим, что slave–устройство не поддерживает данную функцию Modbus.
Как будет выглядеть его ответ? (в HEX, с пробелами между байтами).

4D 90 01 4C 17

6. *Какой тип контрольной суммы используется в протоколе Modbus RTU?*
   - *CRC-64*
   - *CRC-8*
   - *CRC-32*
   - ***CRC-16***
   - *LRC*

2.13 Тестовые задания

1. Укажите размер символа (в битах) для протокола Modbus RTU.
   11

2. Укажите формулировки, справедливые для протокола Modbus ASCII.
   - Бинарный протокол
   - Позволяет передавать данные быстрее, чем Modbus RTU
   - Контрольная сумма является болеё надежной, чем у протокола Modbus RTU
   - ***Позволяет передать данные даже при возникновении существенных задержек между передаваемыми символами***
   - ***Текстовый протокол***
   - ***Размер пакета больше, чем у аналогичного пакета Modbus RTU***

3. Для какой цели в спецификации Modbus введён интервал времени t(3.5)?
   - Для того, чтобы slave-устройство успело переключить свой COM-порт из режима передачи в режим приёма
   - Для того, чтобы master-устройство успело переключить свой COM-порт из режима передачи в режим приёма
   - ***Для определения конца передаваемого пакета***
   - Для определения момента отправки очередного запроса к slave-устройству
   - Для определения отсутствия связи между slave-устройством и master-устройством

4. Какой адрес slave–устройства используется при отправке широковещательного (broadcast) запроса?
   - ***0***
   - 255
   - 248
   - 247

5. Что такое таймаут ответа?
   - Время, в течение которого slave-устройство ожидает ответ от master-устройства
   - Пауза перед ответом на некорректный запрос для формирования правильного кода ошибки
   - Пауза, которая позволяет slave-устройству успеть переключить свой COM-порт из режима передачи в режим приёма
   - ***Время, в течение которого master-устройство ожидает ответ от slave-устройства***

6. Интервалы времени t(1.5) и t(3.5) являются параметрами протокола...
   - Modbus TCP
   - Modbus ASCII
   - ***Modbus RTU***

7. Расположите в правильном порядке состояния master-устройства после отправки запроса slave-устройству в случае получения некорректного ответа, например: несоответствия выделенной из ответа и рассчитанной контрольной суммы, некорректной структуры пакета и т.д.

Idle (ожидание)
Waiting for reply (ожидание ответа)
Processing reply (обработка ответа)
Processing error (обработка ошибки)
Idle (ожидание отправки нового запроса)

2.16 Тестовые задания

1. Какие резисторы используются для предотвращения отражения сигнала от конца линии связи?
   - ***Терминирующие***
   - Ограничивающие
   - Шунтирующие
   - Подтягивающие

2. Какой номинал сопротивления обычно используется для терминирующих резисторов?
   - ***120 Ом***
   - 600 Ом
   - 480 Ом
   - 60 Ом
   - 240 Ом

3. Как называются устройства, подключаемые в шину RS–485 для усиления ослабленного сигнала?
   - Конвертеры
   - ***Повторители***
   - Разветвители
   - Концентраторы

4. С точки зрения спецификации Modbus Serial – сколько slave–устройств можно подключить к последовательной линии связи?
   - ***247***

2.19 Тестовые задания

1. В уроке 2.17 мы с помощью утилиты VSPE мы создали на ПК **два** виртуальных COM–порта и организовали обмен по протоколу Modbus RTU с помощью MasterOPC Universal Modbus Server. В рамках примера master–устройство опрашивало одно slave–устройство. Сколько виртуальных COM–портов нам потребуется для опроса 4 slave–устройств?
   - ***2***
   - 8
   - 5
   - 4

2. Как называется программа–терминал, работающая в режиме прослушивания?
   Сниффер

2.22 Тестовые задания

1. Какие протоколы определены в спецификации Modbus?
   - ***Modbus TCP***
   - Modbus RTU over TCP
   - Modbus UDP
   - Modbus ASCII over TCP

2. Какой TCP–порт должен использоваться на стороне slave–устройства для подключения по протоколу Modbus TCP? (согласно спецификации)
   - ***502***

3. Для чего в запросе Modbus TCP используется адрес slave–устройства? (Unit ID)
   - Для идентификации Modbus TCP slave-устройства, которому нужно доставить данный пакет
   - Для возможности опроса устройств, которые имеют одинаковый IP-адрес
   - Поле адреса не используется, зарезервировано под будущие расширения протокола
   - ***Для возможности пересылки запроса через шлюз Modbus TCP/Modbus RTU***

4. Какие типы запросов не поддерживаются в протоколе Modbus TCP?
   - Запросы с битовыми функциями
   - ***Широковещательные***
   - Групповые
   - Запросы с функциями записи

5. Какое поле пакета протокола Modbus RTU отсутствует в пакете Modbus TCP?
   - Код функции
   - Адрес slave-устройства
   - ***CRC (контрольная сумма)***

4.8 Тестовые задания

Чтобы закрепить изученную информацию – ответьте на ряд тестовых вопросов.

1. Какие режимы работы по Modbus поддерживают панели оператора СП3хх?
   - ***Modbus RTU Master***
   - ***Modbus RTU Slave***
   - ***Modbus ASCII Master***
   - ***Modbus TCP Slave***
   - ***Modbus TCP Master***
   - Modbus RTU over TCP Master
   - Modbus ASCII Slave

2. Какой из вариантов обмена, доступных в панелях оператора СП3хх, поддерживает настройку периода опроса как для запросов чтения, так и для запросов записи?
   - Опрос через элементы
   - Опрос через функциональную область
   - ***Опрос через макросы***

3. Сопоставьте варианты настройки обмена, доступные в панелях оператора СП3хх, и их описания.

Опрос через элементы	***Наиболее прост в настройке, не позволяет детально настраивать групповые запросы и организовать циклическую отправку запросов записи***
Опрос через функциональную область	***Компромиссный вариант, позволяющий ограниченно настраиваить групповые запросы и организовать циклическую отправку запросов записи***
Опрос через макросы	***Наиболее сложен в настройке, позволяет полностью управлять обменом, детально настраивать групповые запросы и организовать циклическую отправку запросов записи***

4. Панель оператора СП3хх работает в режиме Modbus Slave. Какой адрес регистра нужно ввести в Modbus Master, чтобы обратиться к локальному регистру панели **PFW40001**?
   - ***50001***

5.5 Тестовые задания

1. Какое устройство требуется для настройки обмена между приборами, один из которых поддерживает только протокол Modbus TCP, а другой – только Modbus RTU?
   - Коммутатор
   - Конвертер интерфейсов
   - Арбитр
   - ***Конвертер протоколов***

2. Требуется настроить опрос Modbus RTU Slave–устройства, подключённого к преобразователю интерфейсов Ethernet/COM. Какой протокол обмена следует использовать на стороне master–устройства?
   - Modbus TCP over RTU
   - Modbus TCP
   - Modbus RTU
   - ***Modbus RTU over TCP***

3. Сопоставьте режимы работы шлюза Moxa MGate с их описанием:

Классический	***«Запоминает» поступающие запросы с дальнейшей самостоятельной отправкой***
Интеллектуальный	***Пересылает запросы от master к slave без обработки данных***
Агент	***Самостоятельно опрашивает slave-устройства и кэширует результаты***

4. Что такое «прозрачный шлюз»?
   - Устройство, способное преобразовывать протокол в обе стороны
   - Шлюз с маршрутизацией трафика
   - ***Устройство, передающее данные между интерфейсами без преобразования протокола***
   - Устройство для прослушки трафика сети

6.2 Тестовые задания

1. С помощью каких функций Modbus устройства ОВЕН с Ethernet автоматически (в режиме автоопределения) подключаются к облачному сервису OwenCloud?
   - 0x10 Write Multiple Registers
   - ***0x15 (Write File Record)***
   - ***0x14 (Read File Record)***
   - 0x03 (Read Holding Registers)
   - 0x06 Write Single Register
   - 0x04 (Read Input Registers)
   - 0x10 Read/Write Multiple Registers

2. Какие интерфейсы шлюзы Пх210 используют для подключения к OwenCloud?
   - ***Wi-Fi***
   - USB
   - RS-485
   - ***LTE (4G)***
   - ***Ethernet***
   - ***GSM (2G)***
   - Bluetooth

3. Сопоставьте задачи и типовой вариант их решения:

Управление объектом автоматизации с локальным доступом (например, объект электроэнергетики)	**SCADA-система**
Сбор данных с нескольких приборов разных производителей с различными протоколами обмена	**OPC-сервер**
Отображение и архивирование данных, получаемых с удаленных устройств, расположенных на разных объектах	**Облачный сервис**

4. Выберите все верные утверждения про основные функции OPC–сервера:
   - ***Прокладка" между приборами и SCADA-системой***
   - Маршрутизация сетевого трафика
   - ***Эмуляция master- и/или slave-устройств для тестирования обмена***
   - Создание виртуальных COM-портов
   - Диспетчеризация и визуализация параметров