Алгоритм программы

# Организация и использование данных

Данные с датчиков используются для:

* + 1. анализа данных, чтобы выявить тенденцию к изменению параметров и предпринять управляющие меры,
    2. отобразить текущие данные с датчиков на экране устройства,
    3. сохранить текущие данные с датчиков на удалённом компьютере.

Все данные с датчиков организованы в класс Sensor:

* + 1. Time – количество секунд, прошедших со старта программы,
    2. T – температура с датчика,
    3. H – влажность с датчика.
    4. Данные с датчиков сохраняются в двумерном массиве. Измерения массива:
    5. Номер строки в массиве.
    6. Номер датчика.

Массив данных нужен для анализа данных, чтобы выявить тенденцию к изменению параметров и предпринять управляющие меры. Глубина массива (TQ - time quantity) определена по-умолчанию 16 строками. Т.е. в массиве сохраняются только 16 последних измерений. Этого должно быть достаточно, однако, если в процессе работы выявится потребность в большем объеме данных для анализа, размер массива можно будет увеличить.

На экран устройства не всегда выводятся данные с датчиков, поэтому сохранённые текущие данные нужны для обновления информации на экране устройства при открывании соответствующего экрана.

Передача текущих данных на удалённый компьютер ведётся по последовательной шине, например, ModBus. Для передачи запускается отдельная задача, которая получает данные через очередь операционной системы. Задача передачи данных стоит в ожидании готовности данных в очереди. Как только данные появляются в очереди, задача просыпается, отрабатывает передачу и снова засыпает.

С массивом данных работают функции класса Sensor:

* + 1. Принять данные, положить в память – PutData().
    2. Выдать данные по запросу – GetData().

К сохраняемым данным нужно обращаться только через описанные функции.

Данные с датчиков на запись поступают при считывании данных с датчиков. Класс определяет граничные параметры данных и гарантирует, что данные в классе соответствуют заданным критериям.

Данные по температуре и влажности принимаются с датчиков формате int с точностью до десятых долей. Для вывода на экран нужно int перевести во float делением на 10.

Выдача данных осуществляется функцией GetData(TimeFromStart, SensNum, Param). Функция возвращает int число, содержащее температуру или влажность в соответствии с указанными параметрами:

* + 1. TimeFromStart время в секундах со старта,
    2. SensNum – номер датчика,
    3. Param – параметр = 1 для счётчика времени, 2 для температуры, 3 для влажности.

Все параметры управления организованы в структуру, определённую массивом Control.

В структуре присутствует признак вида данных:

* + 1. Время получения данных,
    2. Температура,
    3. Влажность.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Действие | Запуск | Задача | Способ обмена данными | Алгоритм |
| Таймер 1 сек  DataTimerFunc() | Таймер DataTimer | Tmr Svc | * Оповещение задач | Устанавливает флаг FLAG\_ReadData |
| Считывание данных с датчиков  ReadDataFunc() | Установленный флаг FLAG\_ReadData | ReadData | * Оповещение задач * Массив данных, запись * Переменные экрана, запись * Очередь передачи данных, запись | * Считывание с последовательной шины, * запись в массив данных, * запись в очередь передачи данных в удалённый компьютер * запись в переменные экрана, если есть изменения * установка флага FLAG\_DataAnalysis |
| Анализ данных  DataFunc() | Установленный флаг FLAG\_DataAnalysis | DataAnalysis | * Оповещение задач * Массив данных * Массив управляющих значений * Регистр управления | * Чтение из массива нескольких значений, * анализ изменений, * корректировка регистра управления * устанавливает флаг FLAG\_DevManage |
| Запись из регистра управления в выходные регистры | Установленный флаг FLAG\_DevManage | DevManage | * Оповещение задач * Регистр управления | * выявить изменения в регистре управления * изменить данные в регистрах управления устройствами |
| Поступление команды от удаленного компьютера | Прерывание от порта обмена с компьютером |  | * Очередь команд QueueInstr | * Записывает команду в очередь команд QueueInstr |
| Обмен данными с удалённым компьютером | Приём команды в очередь команд QueueInstr | RemoteControl | * Оповещение задач * Очередь команд * Очередь передачи данных | * Устанавливает флаг FLAG\_ReadData * Приём и обработка команды из очереди команд QueueInstr * Передача запрошенных данных через очередь передачи данных QueueData |
| Вывод данных на экран | Таймер GUI | Задача GUI | Переменные экрана | * Чтение из переменных экрана |
|  |  |  |  |  |

# Управление периферийными устройствами

К периферийным устройства относятся:

* + 1. ТЭНы,
    2. Вентиляторы,
    3. Клапан подачи воды на форсунки,
    4. Ворота.

Управление устройствами сводится к битовой операции – включен или выключен. Для каждого устройства выделяется один бит в регистре управления.

Существует один регистр аппаратного управления устройствами, загружаемый в модуль ввода-вывода, битовые выходы которого непосредственно управляют реле включения устройств. Также существуют два регистра состояния управления: автоматического и ручного, состояние которых загружается в регистр аппаратного управления в зависимости от режима работы дефростера (Рисунок 1).

Загрузка регистра аппаратного управления выполняется из регистра состояния устройств в основном цикле работы с датчиками.

Рисунок 1 Регистры управления устройствами

ВыхРГ: ТЭНы

ВыхРГ: Вентиляторы

ВыхРГ: Вода

ВыхРГ: Ворота

Регистр аппаратного управления

Регистр состояния ручного управления

Регистр состояния автоматического управления

Данные из регистра аппаратного управления записываются в выходной регистр модуля ввода-вывода, к выходам которого подключены реле управления устройствами. Таким образом, одной операцией записи в выходной регистр меняются состояния всех устройств.

Флаг состояния каждого устройства в регистрах состояния устройств изменяется битовой операцией, чтобы случайно не изменить состояние битов управления другими устройствами.

Состояние битов регистра аппаратного управления отображается на экране состояния в контроллере устройства. Включение одного из устройств включает анимацию отображаемого на экране устройства. Номер включенного устройства отображается рядом с пиктограммой устройства.

# Работа с данными

Для работы с данными создаётся задача DataProcessing. Задача обрабатывается функцией HandleDataProcessing().

Из функции производится вызов функции DataFunc(), которая обеспечивает обработку данных.

Текст программы работы с данными размещён в отдельном файле Data.c.

Задача стоит в ожидании семафора потребности в работе с данными.

# Считывание данных с датчиков

Данные с датчиков считываются по программному таймеру DataTimer операционной системы. Таймер настроен на 1 секунду, т.е. считывание данных с датчиков производится 1 раз в секунду.

Обработку прерывания от таймера выполняет функция DataTimerFunc(), которая обрабатывается в задаче TmrSvc (timer service). Перегружать таймерный сервис ожиданиями приема-передачи по последовательной шине не очень хорошо, поэтому из таймерного сервиса выполняется запуск задачи ReadData, которая специально создана для работы с последовательной шиной датчиков.

В начале обработки DataTimerFunc() зажигается красный светодиод. И устанавливается флаг FLAG\_ReadData. Красный светодиод гасится через 200 мс. Функция прекращает работу до следующего вызова. Мигание светодиода сообщает о выполнении функции считывания данных.

По флагу FLAG\_ReadData запускается задача ReadData, которая выполняет функцию ReadDataFunc() опроса всех датчиков с внесением данных в массив данных класса Sensor.

В цикле перебираются все номера датчиков, через шину ModBus последовательно с каждого датчика считывается информация.

Параметры шины ModBus:

* + 1. Скорость – 19200 бод,
    2. Бит чётности - нет,
    3. Стоп бит – 1.

Данные с каждого датчика записываются в массив данных класса Sensor в строку с моментом времени Time.

Адреса датчиков на шине фиксированы:

|  |  |
| --- | --- |
| **Адрес, dec** | **Устройство** |
| 101 | Температура и влажность внутренняя дефростера, слева |
| 102 | Температура и влажность внутренняя дефростера, справа |
| 103 | Температура и влажность внутренняя дефростера, по середине |
| 104 | Температура продукта, слева |
| 105 | Температура продукта, справа |
| 002 | Модуль ввода-вывода |

При записи данных в массив класса Sensor функция сравнивает полученные от датчика данные с переменной отображения на экране и, если есть отличие, устанавливает флаг наличия изменений и записывает новое значение в массив переменных экрана.

По окончании считывания данных со всех датчиков задача ReadData засыпает в ожидании новой установки флага FLAG\_ReadData.

# Вывод на экран, переменные

На экран устройства выводятся данные с датчиков в формате числа с плавающей точкой.

На экране визуально отображается информация о работе вентиляторов и ТЭНов.

На экране отображается состояние вкл/выкл клапана подачи воды.

Каждому устройству, отображаемому на экране, присвоен порядковый номер.

Информация с данными для вывода на экран для каждого устройства записывается в массивы:

* + 1. Значение датчика CurrentValue. Отдельно для Т и Н.
    2. Флаг наличия изменений FlagCurrentValueChanged. Отдельно для Т и Н.

Массивы с данными для вывода на экран и флагами изменений сохраняется в модели экрана в классе Class Model в защищённой области. Доступ к элементу массива значений возможен только в модели. Вне модели доступ к значениям переменных возможен только через функции-члены класса Model.

Переменные определены как static, чтобы сохранять значение в течение всего времени работы программы и предоставлять доступ к значениям из любой копии объекта класса, в котором переменные определены. Инициализация static членов класса должна быть выполнена в \*.cpp области, в которой определен класс.

Функции-члены класса Model так же, как и переменные, определены как static. К static-членам класса возможен доступ без создания объекта с типом класса-владельца членов. Для доступа к static-членам класса извне класса перед именем переменной необходимо просто добавить область видимости Model::

* + 1. Model::CurrentValue [SensorNumber],
    2. Model::FlagCurrentValueChanged [SensorNumber].

Доступ к значениям переменных возможен только через функции-члены класса Model:

* + 1. Model::setCurrentVal(int8\_t SensNumber, float Val) – запись значения и установка флага изменений,
    2. float Model::getCurrentVal(int8\_t SensNumber) – возврат значения переменной,
    3. Model::clearFlagCurrentValChng(int8\_t SensNumber) – сброс флага изменений
    4. int8\_t Model::getFlagCurrentValChng(int8\_t SensNumber) – возврат текущего значения флага.

# Вывод на экран, алгоритм

Значения с датчиков считываются отдельной задачей и поступают в Data.cpp (Рисунок 2).

Data.cpp сравнивает новое значение с предыдущим, которое хранится в модели, и, если есть изменение, устанавливает флаг изменений и записывает новое значение датчика в модель.

Модель model.cpp сохраняет значения и обеспечивает безопасный доступ к данным.

Операционная система периодически запускает функцию Model::tick(), находящуюся в модели (model.cpp). Внутри этой функции надо прописать вызовы самописных процедур, в каждой из которых вызывается виртуальная функция, определённая в базовом классе ModelListener в modelListener.hpp. Особенностью виртуальной функции является то, что её может переопределить любой порождённый базовым классом класс. Соответственно, если в текущем ведущем (presenter) есть переопределение функции, то она будет выполнена. Если же переопределения нет, то и исполняться будет нечему.

Таким красивым образом реализован механизм исполнения функции из модели в конкретном ведущем, в котором есть переопределение функции.

Для каждого экрана имеется собственный ведущий, который знает, для каких датчиков организовано отображение информации на собственном экране. В каждом ведущем, где нужно отображать информацию из модели, определена функция ValUpdatePresenter(), которая переопределяет функцию ValUpdatePresenter(), описанную в modelListener.hpp.

Ведущий каждого экрана просматривает массив с данными для вывода на экран только для «своих» устройств и, если флаг изменений установлен, запускает функцию обновления информации в представлении …view.cpp.

После запуска функции обновления ведущий сбрасывает флаг изменений.

Функция обновления информации на экране запускается в соответствующем экрану представлении, она отрабатывает вывод информации на экран.

# Ввод данных с экрана, переменные

Model.cpp

Хранение значений:  
Флаг изменений  
Значение датчика

Data.cpp

Принимает значения с датчиков, если изменение есть, то установка флага и запись нового значения

…presenter.cpp

Ведущий экрана:  
Анализирует флаги «своих» датчиков. Если флаг установлен, запускает обновление поля экрана

…view.cpp

Вывод на экран:  
Преобразует значение в графику, выводит в нужное поле экрана

Рисунок 2. Вывод значений датчиков на экран

С экрана оператор устанавливает параметры процесса для автоматического управления:

* + 1. значение конечной температуры в теле продукта,
    2. значение температуры дефростирования.

Так же с экрана оператор может переключать управление дефростером между ручным и автоматическим режимами.

В ручном режиме оператор с экрана может включать и выключать следующие устройства:

* + 1. нагревателя 1,
    2. нагревателя 2,
    3. нагревателя 3,
    4. нагревателя 4,
    5. вентилятора 1,
    6. вентилятора 2,
    7. вентилятора 3,
    8. вентилятора 4,
    9. водяного клапана,
    10. открытие ворот,
    11. закрытие ворот.

В автоматическом режиме оператор с экрана может выполнить:

* + 1. запуск программы дефростирования,
    2. останов программы дефростирования.

Данные для управления дефростером сохраняются в следующих переменных:

|  |  |  |
| --- | --- | --- |
| **Переменная** | **Место изменения** | **Место хранения** |
| Температура дефростирования | GUI | Model::CurrentValue[8] |
| Конечная температура в теле продукта | GUI | Model::CurrentValue[9] |
| Регистр состояния ручного управления | GUI | Model:: |
| Регистр состояния автоматического управления | MCU |  |
| Регистр аппаратного управления | MCU |  |

Переменные, которые изменяются алгоритмом, сохраняются в программе управления дефростером (MCU).

Переменные, которые изменяются оператором через GUI, сохраняются в модели TouchGFX в классе Class Model в защищённой области. Доступ к элементу массива значений возможен только в модели. Вне модели доступ к значениям переменных возможен только через функции-члены класса Model.

Работа с переменными установки температуры аналогична работе с переменными, в которых сохраняются данные с датчиков в массиве Model::CurrentValue[], поскольку сохраняются в этом же массиве.

Номера датчиков, назначенные переменным параметров управления:

* + 1. Рабочая температура дефростирования – датчик номер 8,
    2. Конечная температура дефростации продукта – датчик номер 9,

# Ввод данных с экрана, алгоритм

Переменная «конечная температура дефростации продукта» определена на экране Settings1 в защищённой переменной класса Settings1View::CoreTSet.

Переменная Settings1View::CoreTSet изменяется встроенными процедурами класса.

Значение переменной из представления в модель передаётся параметром в вызываемой функции через ведущего соответствующего экрана.

Для вызова функции в ведущем используется ссылка на текущего ведущего в переменной presenter. Переменная presenter как ссылка на текущего ведущего определяется в файле view.hpp в классе View (presenter = &newPresenter;).

Из текущего ведущего вызывается функция setCurrentVal в модели (Model::setCurrentVal(SensNum\_DefrOperTemp, Val);), которая записывает значение установленной температуры в массив CurrentVal[].

# Сбор информации с датчиков GL-TH04-MT

Ресурсы MCU и RTOS для работы с датчиками

* + 1. Uart5 с DMA и глобальным прерыванием
    2. Задача MB\_Master\_Task
    3. Очередь MB\_MasterQ
    4. GPIO порт MB\_MASTER\_DE\_GPIO\_Port, бит MB\_MASTER\_DE\_Pin

Ресурсы MCU и RTOS для работы с сервером

* + 1. Uart7 с DMA и глобальным прерыванием
    2. Задача MB\_Slave\_Task
    3. Очередь MB\_SlaveQ
    4. GPIO порт MB\_SLAVE\_DE\_GPIO\_Port, бит MB\_SLAVE\_DE\_Pin
    5. Датчик температуры и влажности GL-TH04-MT.

Формат протокола ModBus RS485.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Направление | 1 | 2 | 3 | 4 | 5 |
| Host sends (TX, Read) | Client address,  1 Byte | Read Code 0x03,  1 Byte | Register address,  2 Bytes | Number of registers, 2 Bytes | CRC 16 verification, 2 Bytes |
| Client replies (RX, Read) | Client address,  1 Byte | Read Code 0x03,  1 Byte | Number of data bytes (N), 1 Byte | Data,  N Bytes | CRC 16 verification, 2 Bytes |
| Host sends (TX, Write) | Client address,  1 Byte | Write Code 0x06,  1 Byte | Register address,  2 Bytes | Data,  2 Bytes | CRC 16 verification, 2 Bytes |
| Client replies (RX, Write) | Client address,  1 Byte | Write Code 0x06,  1 Byte | Register address,  2 Bytes | Data,  2 Bytes | CRC 16 verification, 2 Bytes |

Регистры датчика GL-TH04-MT.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Register address | PLC address | Function of register | Function code | Explanation |
| 0000 H | 40001 | Humidity | 0x03 | 10 times of realtime humidity value |
| 0001 H | 40002 | Temperature | 0x03 | 10 times of realtime temperature value |
| 0050 H | 40081 | Temperature calibration value | 0x03, 0x06 | Integral (10x) |
| 0051 H | 40082 | Humidity calibration value | 0x03, 0x06 | Integral (10x) |
| 07D0 H | 42001 | Client address | 0x03, 0x06 | 1..254 (default 01) |
| 07D1 H | 42002 | Baud rate | 0x03, 0x06 | 0: 2400 bps 1: 4800 bps (default) 2: 9600 3: 19200 4: 38400 5: 57600 6: 115200 7: 1200 |

Далее в примерах опущены байты CRC в конце посылки.

Читаем температуру

* + 1. Хост (TX): 01 03 00 01 00 01 – адрес клиента 01, команда read 03, адрес регистра 0001, количество регистров для чтения 0001,
    2. Клиент (RX): 01 03 02 TT TT - адрес клиента 01, команда read 03, количество байт данных 02, данные температуры TTTT.

Читаем влажность:

* + 1. Хост (TX): 01 03 00 00 00 01 - адрес клиента 01, команда read 03, адрес регистра 0000, количество регистров для чтения 0001,
    2. Клиент (RX): 01 03 02 HH HH - адрес клиента 01, команда read 03, количество байт данных 02, данные влажности HHHH.

Читаем влажность и температуру:

* + 1. Хост (TX): 01 03 00 00 00 02 - адрес клиента 01, команда read 03, адрес регистра 0000, количество регистров для чтения 0002,
    2. Клиент (RX): 01 03 04 HH HH TT TT - адрес клиента 01, команда read 03, количество байт данных 04, данные влажности HHHH, температуры TTTT.
    3. Температура ниже нуля представлена дополнением к двум. Если датчик выдаст 0xFF10 – это 1111 1111 0001 0000 в формате дополнения к двум, нужно инвертировать биты и прибавить единицу: 0000 0000 1110 1111 +1 = 0000 0000 1111 0000, т.е. 240 в десятичной форме. Реальная температура составляет -240/10 = -24℃.
    4. Порядок следования байтов в словах датчика: сначала старший, затем младший. В STM32 принят другой порядок байт в слове: сначала младший, затем старший. Требуется смена байт местами в слове при отправке и приёме данных с датчика.

Читаем скорость передачи:

* + 1. Хост (TX): 01 03 07 D1 00 01 – адрес клиента 01, команда read 03, адрес регистра 07D1, количество регистров для чтения 0001,
    2. Клиент (RX): 01 03 02 00 01 - адрес клиента 01, команда read 03, количество байт данных 02, скорость передачи 0001 = 4800 bps.

Читаем адрес клиента:

* + 1. Хост (TX): 01 03 07 D0 00 01 – адрес клиента 01, команда read 03, адрес регистра 07D0, количество регистров для чтения 0001,
    2. Клиент (RX): 01 03 02 00 01 - адрес клиента 01, команда read 03, количество байт данных 02, адрес клиента 0001.

Изменяем адрес клиента с 01 на 02:

* + 1. Хост (TX): 01 06 07 D0 00 02 – адрес клиента 01, команда write 06, адрес регистра 07D0, адрес клиента 0002,
    2. Клиент (RX): 01 06 07 D0 00 02 - адрес клиента 01, команда write 06, адрес регистра 07D0, адрес клиента 0002.

Если адрес клиента неизвестен, адрес можно запросить у клиента посылкой на широковещательный адрес 0xFF при условии, что клиент на шине один:

* + 1. Хост (TX): FF 03 07 D0 00 01 – адрес клиента 01, команда read 03, адрес регистра 07D0, количество регистров для чтения 0001,
    2. Клиент (RX): 02 03 02 00 02 - адрес клиента 02, команда read 03, количество байт данных 02, адрес клиента 0002.

# Калибровка температуры и влажности датчиков GL-TH04-MT

Регистры 0x0050 и 0x0051 используются, если значение температуры/влажности должно быть отрегулировано. Значение по умолчанию - 0. В регистры записывается смещение кривой вверх (положительное смещение) или вниз (отрицательное смещение).

ВАЖНО!

Чтобы компенсировать температуру на + 2 ° C, вы можете установить значение регистра 0x0050 в 0x14 (2х10=20).

* + 1. Хост (TX): 01 06 0050 0014
    2. Клиент (RX): 01 06 0050 0014

Чтобы увеличить влажность на + 5%, можно установить значение регистра 0x0051 на 0x32 (5х10=50).

* + 1. Хост (TX): 01 06 0051 0032
    2. Клиент (RX): 01 06 0051 0032
    3. Примечание: Отрицательное значение смещения должно быть выражено в виде дополнения двух, например, -2° C, будет представлено как 0xFFEC.

# Управление шиной ModBus

Для управления шиной используется конвертер TTL в ModBus. Его особенностью является аппаратное управление направлением передачи. Сигнал DE=1 включает передачу. DE=0 включает приём. Управление сигналом осуществляется с вывода MB\_MASTER\_DE, контакт PC11.

Передача в шину осуществляется с вывода UART5\_TX, контакт PC12 на плате.

Приём с шины производится на вывод UART5\_RX, контакт PD2 на плате.

# Паузы между символами и фреймами (t1.5 и t3.5)

Одним из самых сложных аспектов реализации протокола Modbus RTU является контроль паузы между байтами и фреймами. В соответствии со спецификацией ([2], п. 2.5.1.1) передаваемые фреймы (пакеты) должны разделяться интервалом тишины определенной длительности на линии связи. В спецификации этот интервал называется «паузой между фреймами» (inter-frame delay) и обозначается как t3.5 (о расчете его длительности мы поговорим ниже). Кроме того, фрейм должен передаваться в виде непрерывной последовательности символов. Если в течение интервала времени t1.5 (в спецификации он назван «межсимвольным таймаутом» (inter-character time-out) ) не приходит новых символов, то устройство может приступать к его проверке на корректность. Если же после истечения времени t1.5 и до истечения времени t3.5 приходит хотя бы один новый символ – то данный фрейм должен считаться некорректным и не обрабатываться устройством (при этом все символы, полученные после истечения межсимвольного таймаута, также отбрасываются, и началом нового фрейма считается первый символ, полученный после выдержки паузы между фреймами).

Под «символом» в данном случае подразумевается один байт фрейма с «обвязкой» – стартовым битом, стоповым битом и битом четности. Согласно спецификации ([2], п. 2.5.1) в протоколе Modbus RTU один байт данных передается в виде 11 бит (8 бит данных и уже упомянутые 3 служебных бита; при этом бит четности используется даже в случае отсутствия контроля четности).

Длительность паузы между фреймами должна составлять не менее интервала времени, который требуется для передачи 3.5 символов для используемой скорости обмена. Формула расчета выглядит следующим образом: 𝑡3.5(мс)=((3.5∙11)/(скорость (бит/с)))∙1000

Например, для скорости 9600 бит/с минимальная длительность паузы между фреймами составит ≈ 4 мс.

Межсимвольный таймаут вычисляется аналогичным образом: 𝑡1.5(мс)=((1.5∙11)/(скорость (бит/с)))∙1000

Если скорость обмена превышает 19200 бит/с, то спецификация рекомендует вместо формул использовать константы: 0.750 мс для t1.5 и 1.750 мс для t3.5. Это связано с тем, что на высоких скоростях обмена при расчете интервалов времени по формулам, прерывания аппаратных таймеров будут срабатывать слишком часто (например, для скорости 115200 бит/с интервал t3.5 составлял бы ≈ 0.33 мс), что может мешать выполнению других задач.

# Алгоритм сбора информации с датчиков

При инициализации системы запускается задача ReadData, в которой в бесконечном цикле крутится функция ReadDataFunc() из файла Data.cpp. Внутри функции встроено бесконечное ожидание флага запуска чтения датчиков (Рисунок 3).

RTOS

Task ReadData

Timer DataTimer

Уст.флага ReadDataEvent

LED on  
Pause  
LED off

Ожид.флага ReadDataEvent

STOP

Ожидание семафора

TX\_Compl\_Sem

Ожидание семафора

RX\_Compl\_Sem

DMA запись буфера в передатчик UART

Все датчики?

Запуск приёма UART

N

Y

START

Передача UART

CallBack

Окончание приёма

CallBack

Окончание передачи

Приём UART,  
DMA запись в буфер

Обработка принятых данных

Рисунок 3. Алгоритм сбора информации с датчиков.

Инициализация при запуске задачи ReadData. Проверка работоспособности датчиков при запуске. Опрос всех датчиков, регистрация активности в таблице.

По таймеру DataTimer запускается функция DataTimerFunc(), внутри которой устанавливается флаг ReadDataEvent для запуска задачи ReadData.

Установленный флаг разрешает функции ReadDataFunc() опросить датчик.

Выполняется запуск передачи запроса на датчик в режиме DMA через функцию RTOS. На время передачи UART задача ReadData уходит в ожидание окончания передачи данных по UART, поскольку загрузка в передатчик в режиме DMA производится в разы быстрее, чем передача по последовательному каналу.

Обработка ошибок передачи данных передатчику UART в режиме DMA:

* + 1. HAL\_OK – всё хорошо, идём дальше;
    2. HAL\_ERROR – проблема, передать не удалось, надо повторить передачу;
    3. HAL\_BUSY – передатчик занят предыдущей передачей, надо повторить передачу;
    4. HAL\_TIMEOUT – время на передачу вышло, надо повторить передачу.
    5. После повторной неудачи передачи надо сообщить оператору об аварии передатчика.

Если передача DMA выполнена успешно, задача ReadData переходит в ожидание окончания передачи по последовательному каналу. Ожидается семафор TX\_Compl\_Sem.

Семафор TX\_Compl\_Sem устанавливается в прерывании окончания передачи HAL\_UART\_TxCpltCallback.

Обработка ошибок передачи по последовательному каналу:

* + 1. Прерывание окончания передачи сработало, семафор TX\_Compl\_Sem установился, задача ReadData продолжает работу;
    2. Прерывание отсутствует, ожидание семафора завершилось по таймауту – нужно повторить отправку.
    3. После повторной неудачи передачи надо сообщить оператору об аварии передатчика.

В продолжение работы задача ReadData выполняет запуск приёма данных из порта UART в режиме DMA.

Обработка ошибок передачи данных передатчику UART в режиме DMA:

* + 1. HAL\_OK – всё хорошо, идём дальше;
    2. HAL\_ERROR – проблема, передать не удалось, надо повторить передачу;
    3. HAL\_BUSY – передатчик занят предыдущей передачей, надо повторить передачу;
    4. HAL\_TIMEOUT – время на передачу вышло, надо повторить передачу.
    5. После повторной неудачи передачи надо сообщить оператору об аварии передатчика.

По окончании приёма функция обработки прерывания выполняет установку семафора RX\_Compl\_Sem.

Обработка ошибок приёма по последовательному каналу:

* + 1. Прерывание окончания приёма сработало, семафор RX\_Compl\_Sem установился, задача ReadData продолжает работу;
    2. Прерывание отсутствует, ожидание семафора завершилось по таймауту – нужно повторить обмен с датчиком.
    3. После повторной неудачи приёма надо снять флаг активности с датчика и сообщить оператору об аварии датчика.

Задача ReadData выполняет обработку и запись принятых данных из буфера в переменные класса Sensor.

Далее повторяется процесс опроса для следующего датчика. Опрашиваются только активные датчики. Процесс опроса повторяется для всех датчиков системы.

# Процесс обмена по шине UART

Данные из буфера в режиме DMA записываются в передатчик. Запускается UART.

Обработка ошибки DMA.

Ждём семафор окончания передачи UART.

Обработка таймаута ожидания.

По окончании передачи UART – прерывание по TC. Устанавливается семафор окончания передачи.

Обработка ошибки UART.

Запускается приём UART в режиме DMA. Принятые данные записываются в буфер в режиме DMA.

По окончании приёма UART – прерывание по EVENT. Устанавливается семафор окончания приёма.

Обработка ошибки UART.

Ждём семафор окончания приёма UART.

Обработка таймаута ожидания.

Принятые в буфер данные обрабатываются.

# Реализация процесса обмена по шине UART в программе

Есть три логических уровня работы с шиной ModBus: высокий, средний и низкий.

На низком уроне осуществляется непосредственная работа с шиной: установка сигналов, работа с DMA, обработка прерываний. За обмен по шине UART отвечает функция Master\_Request().

На среднем уровне формируются данные для отправки устройствам на шине и принимаются данные в переменные среды, зависящие от команды устройству. За формирование данных отвечает функция Master\_RW().

На высоком уровне производится формирование программной среды для работы с устройствами на шине, формируются запросы устройствам по типам устройств, обрабатываются принятые данные и ошибки. Параметры порта UART, GPIO, буферы и семафоры передачи и приёма передаются в программной среде работы с шиной - структуре MB. Для каждой шины создаётся своя структура MB.

Для рабочей шины на высоком уровне работает функция Sensor\_Read().

Для шины программирования на высоком уровне – ProgrammingSensor(), в которую включены две функции: ScanSensor() и WriteToSensor().

# Особенности процесса программирования датчика

Подготавливает данные для передачи в датчик и проверяет полученные данные из датчика функция Master\_RW().

Функция Master\_RW() использует для обмена по шине функцию Master\_Request().

Функция Master\_RW() получает в качестве параметров следующее:

* + 1. среда работы с датчиком,
    2. адрес датчика,
    3. команда датчику,
    4. начальный регистр,
    5. данные (для чтения - кол-во считываемых регистров, для записи - данные для записи в регистр).

Функция Master\_RW() возвращает статус обработки запроса к датчику.

# Программирование датчика GL-TH04-MT на шине UART

Каждый датчик типа GL-TH04-MT поставляется с установленными заводскими параметрами. Для установки в систему датчик нужно запрограммировать.

Для программирования:

* + 1. Отдельное окно в интерфейсе.
    2. Отдельная задача для программирования ProgrammingSens. Создаётся при открытии окна (Settings5View::setupScreen()) и удаляется при закрытии окна (Settings5View::tearDownScreen()). Процедура программирования датчиков находится в модуле ModBus.cpp – ProgrammingSensor().
    3. Отдельный порт UART4 в контроллере, биты PA0 (TX), PA1 (RX).
    4. Отдельная колодка для подключения датчика.
    5. Отдельный бит переключения прием/передача - бит PC2 (PROG\_MASTER\_DE).
    6. Отдельные семафоры PR\_TX\_ComplSem и PR\_RX\_ComplSem.
    7. Подпрограммы управления находятся в ModBus.cpp.

При запуске окна программирования параметров датчика программа запускает процесс сканирования шины на разных скоростях обмена по шине.

По-умолчанию датчик с завода идёт с установленной скоростью 4800 бит/с и адресом на шине 01.

Сканирование шины производится командой чтения данных по широковещательному адресу 0xFF из регистров:

* + 1. 0x7D0 – адрес устройства,
    2. 0x7D1 – скорость обмена по шине.

После ответа датчика фиксируется скорость обмена для подачи команды записи в регистры адреса и скорости.

В окне программирования выбираются значения из списка предустановленных параметров как адреса, так и скорости. После нажатия на кнопку «Записать» новые параметры передаются в устройство.

Далее производится регулярное считывание данных температуры и влажности и отображение их в окне программирования.

При распиновке UART4 я столкнулся с тем, что, задействуя некоторые выводы процессора, я останавливал работу системы. По-видимому, я занимал выводы, которые использовал и LTDC, и отладчик SWD.

Всегда в нуле был вывод RX PA1 для UART4. Пришлось пару RX TX перенести на выводы PC11, PC10.

Сбивалась работа дисплея при использовании выводов PD4, PA14 и ещё нескольких, я их не записал и не запомнил.

Не работал выход PROG\_MASTER\_DE на каком-то выводе.

Нет

Нет

Да

СТАРТ

Установка скорости

Запрос Master\_RW()

Есть ответ?

Отображение на экране полученных данных

Есть запись?

Запись адреса  
Master\_RW()

Запись скорости  
Master\_RW()

Да

Master\_Request()

Master\_Request()

Master\_Request()

При очистке вывода PC2 дисплей переставал отображать информацию. Я оставил вывод PC2 для выхода PROG\_MASTER\_DE.

Обработка функции программирования находится в файле ModBus.cpp.

Структура вложенности процедур:

|  |  |
| --- | --- |
| Функция | Описание |
| ProgrammingSensor() | Запускает сканирование шины, сравнивает предыдущее значение с текущим. Если есть разница, отправляет на экран. |
| ScanSensor(), WriteToSensor() | ScanSensor выполняет запрос и считывание с шины, устанавливая последовательно все разрешённые скорости обмена, WriteToSensor() записывает данные в датчик и проверяет записанное. |
| Master\_RW() | Формирует запрос на шину для считывания с датчика установленных скорости и адреса и записывает их в переменные, если получает ответ. |
| Master\_Request() | Отправляет запрос на шину из буфера MB->Tx\_Buffer и считывает ответ, если он последует, в буфер MB->Rx\_Buffer. |
| PR\_UART4\_Init() | Проводит инициацию порта UART4 на заданную скорость обмена. |

# Передача считанных данных с программируемого датчика на экран

Полученные от датчика данные сохраняются в переменных SensPortNumber – адрес датчика, SensBaudRateIndex – индекс скорости.

Если после цикла опроса шины на всех доступных скоростях ответа не последовало, в переменные записывается SensNullValue (константа 255). При выводе на экран это значение заменяется прочерками «---».

Если после цикла опроса шины ответ есть, то данные из переменных SensPortNumber – адрес датчика, SensBaudRateIndex – индекс скорости сравниваются с сохраняемыми в модели текущими данными датчика.

Данные датчика из модели можно получить процедурами Model::getCurrentBoadRate\_PR() и Model::getCurrentAddress\_PR().

Если полученные данные датчика отличаются от сохранённых в модели, то данные в модели перезаписываются процедурой Model::setCurrentVal\_PR(SensPortNumber, SensBaudRateIndex). Эта процедура кроме записи текущих данных устанавливает флаг изменения данных FlagCurrentValue\_PR\_sensor.

Модель вызывает в текущем ведущем процедуру ValUpdatePresenter(), и, если флаг изменения данных установлен, то ведущий вызывает изменение данных на экране и сбрасывает флаг изменения данных:

view.Val\_Addr\_UpdateView(Model::getCurrentAddress\_PR())

view.Val\_BoadRate\_UpdateView(Model::getCurrentBoadRate\_PR()

Model::clearFlagCurrentVal\_PR\_Chng()

Представление перед выводом на экран проверяет данные и, если данные равны SensNullValue, то выводит стринг «---», иначе выводит на экран новое значение.

# Ввод данных для записи в датчик с экрана, алгоритм

Переменная «адрес для установки» определена на экране Settings5View.cpp в переменной SetAddress.

Переменная «скорость для установки» определена на экране Settings5View.cpp в переменной SetSpeed.

Значение переменной из представления в модель передаётся параметром в вызываемой функции через ведущего соответствующего экрана.

Для вызова функции в ведущем используется ссылка на текущего ведущего в переменной presenter. Переменная presenter как ссылка на текущего ведущего определяется в файле D:/ST/Defrost/Middlewares/ST/touchgfx/framework/include/mvp /view.hpp в классе View (presenter = &newPresenter;).

В текущем ведущем из представления запускается функция Settings5Presenter::PR\_Sensor\_Data\_Write(), в которой переданные из представления данные записываются в модель.

Установленный флаг Model::Flag\_WR\_to\_sensor запускает процесс записи данных в датчик.

# Алгоритм программирования датчика

При программировании можно в один приём записать в датчик только один регистр – либо адрес, либо скорость обмена, поэтому после записи первого значения нужно будет записывать второй регистр уже для обновленного установленного значения регистра.

Выбран следующий режим записи: сначала записывается адрес, затем скорость.

Параметры для обмена с датчиком формируют «среду для работы с датчиком». Набор параметров организован в структуру с типом MB\_Active\_t. В основном элементы структуры – это указатели на объекты RTOS: порты ввода-вывода, семафоры. Буферы передачи и приёма, а также номер разряда порта вывода для управления направлением передачи формируются в отдельных областях памяти.

Структура «среды для работы с датчиком» заполняется при инициации функции обмена с датчиком и зависит от функционала конкретной линии связи с датчиком: только чтение (huart5) или чтение и запись (huart5).

Структура «среды для работы с датчиком» передаётся в функцию как параметр в виде указателя на структуру.

Для функции Master\_Request(), выполняющей обмен данными по шине, в неявном виде передаются следующие параметры для обмена с датчиком:

* + 1. Адрес,
    2. Стартовый регистр,
    3. Количество регистров,
    4. Команда.
    5. Данные для записи в регистр,
    6. Контрольная сумма.

Из всех передаваемых данных для датчика формируется массив в буфере обмена, который функция Master\_Request() использует для передачи.

Функция Master\_RW () использует Master\_Request() для записи в датчик. Единовременно в датчик можно записать только один регистр, поэтому явно в функцию передаём адрес регистра и данные.

Таблица длины сообщения в байтах при обмене с датчиком.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Bite N | Read | Read answer | Broadcast read | Broadcast answer | Write | Write answer |
| 0 | ADDR | ADDR | 0xFF | ADDR | ADDR | ADDR |
| 1 | CMD 03 | CMD 03 | CMD 03 | CMD 03 | CMD 06 | CMD 06 |
| 2 | REG addr | N Bytes 04 | REG addr | N Bytes 02 | REG addr | REG addr |
| 3 | DATA0 | DATA0 |
| 4 | Number of REG | DATA1 | Number of REG | DATA1 | DATA0 | DATA0 |
| 5 | DATA2 | DATA2 | DATA1 | DATA1 |
| 6 | CRC | DATA3 | CRC | DATA3 | CRC | CRC |
| 7 | CRC | CRC |
| 8 |  |  |  |  |

# Вызов функций при работе с датчиками

Определение рабочих датчиков при инициализации: MB\_Master\_Init() -> Sensor\_Read() -> Master\_RW() -> Master\_Request().

Чтение рабочей информации: Sensor\_Read() -> Master\_RW() -> Master\_Request().

Программирование датчика, поиск: ProgrammingSensor() -> {ScanSensor(), WriteToSensor()} -> Master\_RW() -> Master\_Request().

Программирование датчика, запись: Master\_RW() -> Master\_Request().

# Сбор информации с датчика PT100 с RS485 - PT21A01

Использую информацию о программировании датчика PT21A01/PT22B01.

Формат протокола RS-485:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Направление | 1 | 2 | 3 | 4 | 5 |
| Host sends (TX, Read) | Client address,  1 Byte | Read Code 0x03,  1 Byte | Register address,  2 Bytes | Number of registers, 2 Bytes | CRC 16 verification, 2 Bytes |
| Client replies (RX, Read) | Client address,  1 Byte | Read Code 0x03,  1 Byte | Number of data bytes (N), 1 Byte | Data,  N Bytes | CRC 16 verification, 2 Bytes |
| Host sends (TX, Write) | Client address,  1 Byte | Write Code 0x06,  1 Byte | Register address,  2 Bytes | Data,  2 Bytes | CRC 16 verification, 2 Bytes |
| Client replies (RX, Write) | Client address,  1 Byte | Write Code 0x06,  1 Byte | Register address,  2 Bytes | Data,  2 Bytes | CRC 16 verification, 2 Bytes |

Регистры датчика:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Register address | Function of register | Function code | Units | Explanation |
| 0000 H | CH0 Temperature value | 0x03 | 0.1 ℃ | For example, the data is 255, which is equal to 25.5 ℃ |
| 0020 H | CH0 PT100 resistance value | 0x03 | 0.1 Ω | For example, the data is 1000, which is equal to 100.0Ω |
| 0040 H | CH0 Temperature correction value. This register can only be written. Data if read: 0XFFFF | 0x03, 0x06 | 0.1 ℃ | If the temperature is deviated, please input the correct temperature value for correction |
| 0060 H | CH0 PT100 resistance correction positive value. This register can only be written. Data if read: 0XFFFF | 0x03, 0x06 | 0.1 Ω | If the PT100 resistance value is deviated, please input the correct resistance value for correction |
| 00F9 H | Temperature Unit | 0x03, 0x06 |  | 0: Degrees Celsius  1: Degree Fahrenheit |
| 00FA H | Automatic upload of temperature | 0x03, 0x06 | s | 0: Query function (default)  1-255: Automatically report, the unit is second.  For example: 1: Report every 1 second 2: Report every 2 seconds 10: Report every 10 seconds  Maximum interval of 255 seconds |
| 00FB H | Factory data reset | 0x06 |  | The address code is 0xFF, and at the current baud rate, enter the following commands to restore the factory settings:  FF 06 00 FB 00 00 ED E5 |
| 00FC H | Data return delay | 0x03, 0x06 | ms | After receiving the command, return the data interval time (unit ms) |
| 00FD H | RS485 address (Station address) | 0x03, 0x06 |  | Read Broadcast Address: 0xFF  Write Address: 1-254 |
| 00FE H | Baud rate | 0x03, 0x06 |  | 0: 1200 1: 2400 2: 4800 3: 9600 4: 19200 5: 38400 6: 57600 7: 115200  8: Factory reset |
| 00FF H | Parity | 0x03, 0x06 |  | 0: None Parity 1: Even Parity 2: Odd Parity |

Таблица длины сообщения в байтах при обмене с датчиком.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Bite N | Read | Read answer | Broadcast read | Broadcast answer | Write | Write answer |
| 0 | ADDR | ADDR | 0xFF | ADDR | ADDR | ADDR |
| 1 | CMD 03 | CMD 03 | CMD 03 | CMD 03 | CMD 06 | CMD 06 |
| 2 | REG addr | N Bytes 02 | REG addr | N Bytes 02 | REG addr | REG addr |
| 3 | DATA0 | DATA0 |
| 4 | Number of REG | DATA1 | Number of REG | DATA1 | DATA0 | DATA0 |
| 5 | CRC | CRC | DATA1 | DATA1 |
| 6 | CRC | CRC | CRC | CRC |
| 7 |  |  |

Заводские параметры (по-умолчанию): Serial baud rate: 9600, N, 8, 1.

Чтение температуры:

* + 1. Номер регистра 0000h.
    2. Возвращаемые данные о температуре составляют два байта, старший байт в первом и младший байт в последующем, преобразуем их в десятичные и делим на 10, это текущее значение температуры.
    3. Старший бит 1 указывает на отрицательное значение, это значение в дополнительном коде. Из полученного значения нужно вычесть 65536, результат будет являться текущим значением температуры.
    4. Например, читаем температуру из датчика 01: 01 03 00 00 00 01 84 0A.
    5. Где 01 – адрес датчика, 03 – команда, 0000 – номер регистра, 0001 – считать из одного регистра, 840A – CRC код.
    6. Получаем ответ: 01 03 02 00 DB F8 1F.
    7. Где 01 – адрес датчика, 03 – команда, 02 – длина передаваемых данных, 00DB – значение температуры, F81F – CRCкод.
    8. 00DBh – положительная величина, dec 219, 219/10=21.9 – текущая температура в ℃.
    9. Если получаем ответ: 01 03 02 FF 90 F2 3F.
    10. Где 01 – адрес датчика, 03 – команда, 02 – длина передаваемых данных, FF90 – значение температуры, F23F – CRCкод.
    11. FF90h – отрицательная величина, dec 65424, преобразуем (65424-65536)/10=-11.2 – текущая отрицательная температура в ℃.

Чтение сопротивления датчика:

* + 1. Номер регистра 0020h.
    2. Возвращаемые данные о сопротивлении составляют два байта, старший байт в первом и младший байт в последующем, преобразуем их в десятичные и делим на 10, это текущее значение сопротивления.
    3. Например, читаем сопротивление из датчика 01: 01 03 00 20 00 01 85 C0.
    4. Где 01 – адрес датчика, 03 – команда, 0020 – номер регистра, 0001 – считать из одного регистра, 85C0 – CRC код.
    5. Получаем ответ: 01 03 02 04 64 BB 6F.
    6. Где 01 – адрес датчика, 03 – команда, 02 – длина передаваемых данных, 0464 – значение сопротивления, BB6F – CRCкод.
    7. 0464h, dec 1124, 1124/10=112.4 – текущее сопротивление в Ω.
    8. Прочитать адрес датчика:
    9. Номер регистра 00FDh.
    10. Широковещательный адрес: 0xFF. На шине должен быть только один датчик!
    11. Например, читаем адрес из датчика на шине: FF 03 00 FD 00 01 00 24.
    12. Где FF – широковещательный адрес, 03 – команда, 00FD – номер регистра, 0001 – считать из одного регистра, 0024 – CRC код.
    13. Получаем ответ: FF 03 02 00 01 50 50.
    14. Где FF – широковещательный адрес, 03 – команда, 02 – длина передаваемых данных, 0001 – текущий адрес модуля, 50 50 – CRCкод.

Записать новый адрес датчика:

* + 1. Номер регистра 00FDh.
    2. Например, знаем адрес датчика: 0x01.
    3. Посылаем запрос датчику: 01 06 00 FD 00 03 58 3B.
    4. Где 01 – адрес датчика, 06 – команда, 00FD – номер регистра, 0003 – данные для записи, 583B – CRC код.
    5. Получаем ответ: 01 06 00 FD 00 03 58 3B.
    6. Где 01 – адрес датчика, 06 – команда, 00FD – номер регистра, 0003 – новый адрес модуля, 583B – CRCкод.

Прочитать скорость датчика:

* + 1. Номер регистра 00FEh.
    2. Например, знаем адрес датчика: 0x01. В инструкции про ответ на широковещательный запрос по скорости ответа нет.
    3. Читаем скорость из датчика: 01 03 00 FE 00 01 E5 FA.
    4. Где 01 – адрес датчика, 03 – команда, 00FE – номер регистра, 0001 – считать из одного регистра, E5FA – CRC код.
    5. Получаем ответ: 01 03 02 00 03 F8 45.
    6. Где FF – широковещательный адрес, 03 – команда, 02 – длина передаваемых данных, 0003 – текущая скорость модуля, F845 – CRCкод.
    7. Цифра означает одно из списка: 0: 1200, 1: 2400, 2: 4800, 3: 9600, 4: 19200, 5: 38400, 6: 57600, 7: 115200, 8: Factory reset.
    8. 03h означает скорость 9600 бод.

Записать новую скорость датчика:

* + 1. Номер регистра 00FEh.
    2. Например, знаем адрес датчика: 0x01 и меняем скорость на 4800 бод.
    3. Посылаем запрос датчику: 01 06 00 FE 00 02 69 FB.
    4. Где 01 – адрес датчика, 06 – команда, 00FE – номер регистра, 0002 – данные для записи, 69FB – CRC код.
    5. Получаем ответ: 01 06 00 FE 00 02 69 FB.
    6. Где 01 – адрес датчика, 06 – команда, 00FE – номер регистра, 0002 – новая скорость модуля, 69FB – CRCкод.
    7. Цифра для записи означает одно из списка: 0: 1200, 1: 2400, 2: 4800, 3: 9600, 4: 19200, 5: 38400, 6: 57600, 7: 115200, 8: Factory reset.
    8. Установлена скорость 4800 бод.
    9. ВАЖНО! Скорость обмена модуля сменится на новую только после отключения питания модуля и включения вновь!
    10. ВАЖНО! Скорость обмена модуля можно вернуть в заводскую установку путём записи кода 8, например: 01 06 00 FE 00 08 E9 FC

Коррекция температуры.

* + 1. Номер регистра 0040h.
    2. Если текущая температура отличается от фактической, можно скорректировать прямую сопоставления сопротивление-температура в модуле путём записи фактической температуры в модуль.
    3. Так как для датчика PT100 линейная зависимость сопротивления от температуры, то, как минимум, нужно два измерения температуры для корректной коррекции датчика.
    4. Пример1: фактическая температура 25.5 ⁰С, а из датчика прочитали 26.4 ⁰С. Прочитанную температуру можно скорректировать, записав в модуль число 25.5\*10=255, или FFh в hex формате.
    5. Отправим: 01 06 00 40 00 FF C8 5E
    6. Получим: 01 06 00 40 00 FF C8 5E
    7. Пример2: запишем в модуль отрицательную температуру -12.1 ⁰С. Это число 65536-121=65415, или FF87h в hex формате.
    8. Отправим: 01 06 00 40 FF 87 89 8C
    9. Получим: 01 06 00 40 FF 87 89 8C

Коррекция сопротивления.

* + 1. Номер регистра 0060h.
    2. Если текущее сопротивление отличается от фактического, можно скорректировать прямую сопоставления сопротивление-температура в модуле путём записи фактического сопротивления в модуль.
    3. Так как для датчика PT100 линейная зависимость сопротивления от температуры, то, как минимум, нужно два измерения сопротивления для корректной коррекции датчика.
    4. Пример1: фактическое сопротивление датчика 100 Ω, а из датчика прочитали 102 Ω. Прочитанное сопротивление можно скорректировать, записав в модуль число 100\*10=1000, или 03E8h в hex формате.
    5. Отправим: 01 06 00 60 03 E8 89 6A
    6. Получим: 01 06 00 60 03 E8 89 6A

Читаем установку автоматического рапорта о температуре

* + 1. Номер регистра 00FAh.
    2. Модуль умеет передавать температуру по запросу от мастера на шине или отправлять мастеру рапорт о температуре автоматически с установленным интервалом времени. Интервал времени записывается в регистр и должен быть выражен в секундах.
    3. Отправим: 01 03 00 FA 00 01 A4 3B
    4. Где 01 – адрес датчика, 03 – команда чтения, 00FA – номер регистра, 0001 – читаем один регистр, A43B – CRC код.
    5. Получим: 01 03 02 00 00 B8 44
    6. Где 01 – адрес датчика, 03 – команда чтения, 02 – длина данных в байтах, 0000 – включена функция отправок данных по запросу, B844 – CRC код.
    7. Установим функцию автоматического рапорта о температуре
    8. Номер регистра 00FAh.
    9. Для рапорта каждые 1 секунду отправим: 01 06 00 FA 00 01 68 3B
    10. Для рапорта каждые 2 секунды отправим: 01 06 00 FA 00 02 28 3A
    11. Для рапорта каждые 3 секунды отправим: 01 06 00 FA 00 03 E9 FA
    12. Для рапорта каждые 4 секунды отправим: 01 06 00 FA 00 04 A8 38
    13. Для рапорта каждые 5 секунд отправим: 01 06 00 FA 00 05 69 F8
    14. Для рапорта каждые 10 секунд отправим: 01 06 00 FA 00 0A 29 FC
    15. Для отмены функции рапорта отправим: 01 06 00 FA 00 00 A9 FB
    16. Ответ от датчика должен быть таким же, как запрос.

Прочитать параметр контроля чётности

* + 1. Номер регистра 00FFh.
    2. Отправим: 01 03 00 FF 00 01 B4 3A
    3. Где 01 – адрес датчика, 03 – команда чтения, 00FF – номер регистра, 0001 – читаем один регистр, B43A – CRC код.
    4. Получим: 01 03 02 00 00 B8 44
    5. Где 01 – адрес датчика, 03 – команда чтения, 02 – длина данных в байтах, 0000 – контроль чётности выключен, B844 – CRC код.
    6. Возможные коды ответа: 0 – контроль чётности отключен; 1 – чёт (even parity); 2 – нечет (odd parity).

Установить параметр контроля чётности

* + 1. Номер регистра 00FFh.
    2. Отправим: 01 06 00 FF 00 01 78 3A
    3. Где 01 – адрес датчика, 06 – команда записи, 00FF – номер регистра, 0001 – установить контроль по чётности, 783A – CRC код.
    4. Получим: 01 06 00 FF 00 01 78 3A
    5. Где 01 – адрес датчика, 06 – команда записи, 00FF – номер регистра, 0001 –контроль по чётности установлен, 783A – CRC код.
    6. Возможные коды записи и ответа: 0 – контроль чётности отключен; 1 – чёт (even parity); 2 – нечет (odd parity).

Прочитать время задержки отклика датчика на запрос

* + 1. Номер регистра 00FCh.
    2. Отправим: 01 03 00 FC 00 01 44 3A
    3. Где 01 – адрес датчика, 03 – команда чтения, 00FC – номер регистра, 0001 – читаем один регистр, 443A – CRC код.
    4. Получим: 01 03 02 00 00 B8 44
    5. Где 01 – адрес датчика, 03 – команда чтения, 02 – длина данных в байтах, 0000 –время задержки 0 мс, B844 – CRC код.
    6. Возможные коды записи и ответа: 0 – контроль чётности отключен; 1 – чёт (even parity); 2 – нечет (odd parity).

Установить время задержки отклика датчика на запрос

* + 1. Номер регистра 00FCh.
    2. Размер устанавливаемой величины задержки – 2 байта в миллисекундах.
    3. Например, установим задержку ответа 200 мс, отправим: 01 06 00 FC 00 C8 48 6C
    4. Где 01 – адрес датчика, 06 – команда записи, 00FC – номер регистра, 00C8 – передаём число 200 (00C8h), 486C – CRC код.
    5. Получим: 01 06 00 FC 00 C8 48 6C
    6. Где 01 – адрес датчика, 06 – команда записи, 00FC – номер регистра, 00C8 – время задержки 200 мс (00C8h), 486C – CRC код.

Прочитаем установку единиц измерения температуры

* + 1. Номер регистра 00F9h.
    2. Отправим: 01 03 00 F9 00 01 54 3B
    3. Где 01 – адрес датчика, 03 – команда чтения, 00F9 – номер регистра, 0001 – читаем один регистр, 543B – CRC код.
    4. Получим: 01 03 02 00 00 B8 44
    5. Где 01 – адрес датчика, 03 – команда чтения, 02 – длина данных в байтах, 0000 –единица измерения в ⁰С, B844 – CRC код.
    6. Возможные коды записи и ответа: 0 – в Цельсиях (⁰С); 1 – в Фаренгейтах (F).

Установим единицу измерения температуры

* + 1. Номер регистра 00F9h.
    2. Например, установим единицу измерения в Фаренгейтах.
    3. Отправим: 01 06 00 F9 00 01 98 3B
    4. Где 01 – адрес датчика, 06 – команда записи, 00F9 – номер регистра, 0001 – установить в Фаренгейты, 983B – CRC код.
    5. Получим: 01 06 00 F9 00 01 98 3B
    6. Возможные коды записи и ответа: 0 – в Цельсиях (⁰С); 1 – в Фаренгейтах (F).

Сброс в заводские установки

* + 1. Номер регистра 00FBh.
    2. Сбросить в заводские установки можно через широковещательный адрес.
    3. Отправим: FF 06 00 FB 00 00 ED E5
    4. Где FF – широковещательный адрес датчика, 06 – команда записи, 00FB – номер регистра, 0000 – код сброса, EDE5 – CRC код.
    5. Получим: FF 06 00 FB 00 00 ED E5

Аппаратный сброс в заводские установки – замкнуть джампер RES на включенном модуле на 5 секунд, затем снять питание и снова включить датчик.

# Калибровка датчиков

Калибровка датчиков производится на странице настроек Settings6.

Действия по калибровке выполняются с работающими на шине датчиками, поэтому запускать отдельную задачу не нужно, калибровка добавляется в конец цикла опроса датчиков в функции ReadDataFunc() из файла Data.cpp.

Settings 6

Выбор датчика

Установка типа датчика по его адресу

Считывание данных:  
Т, Н, dТ, dН

Тип датчика

Тип 1

Тип 2

Считывание данных:  
Т, R

Выбор параметра корректировки:  
Т или R

Установка параметра корректировки

Запись параметра корректировки

Выбор параметра корректировки:  
Т или H

Установка параметра корректировки

Запись параметра корректировки

При выборе датчика тип этого датчика выбирается из таблицы датчиков и далее алгоритм работает с установленным типом датчика.

Об окончании выбора датчика и готовности к корректировке сообщает установленный флаг Flag\_CORR\_ready, размещённый в модели.

Датчик типа 1 корректируется записью смещения в регистр. Для него нужно определить корректировочное смещение и записать его в соответствующий регистр. Корректировочное смещение сохраняется в регистре корректировки. Чтобы правильно скорректировать значение, надо прочитать текущую корректировку из регистра, изменить её и записать обратно.

Благодаря тому, что датчик типа 1 сохраняет не само значение, а его корректировочное смещение, можно вернуть датчик к заводским настройкам простым обнулением регистра с корректировочным смещением. Это делается кнопкой СБРОС на экране корректировки.

Датчик типа 2 вычисляет смещения для корректировки сам. Возможно, он может изменять наклон температурной кривой. Для корректировки нужно записать в корректировочный регистр правильное значение температуры или сопротивления.

Предположительно, для изменения наклона корректировочной кривой в датчике типа 2 нужно записать два правильных значения: в верхней и в нижней частях кривой. Утверждение требует опытной проверки.

Датчик типа 1 корректирует показания температуры и влажности.

Датчик типа 2 может скорректировать как температуру, вычисляемую по измеренному сопротивлению датчика, так и измеряемое сопротивление датчика.

Текущие параметры для корректировки, сохраняемые в модели:

* + 1. Index\_CORR\_sensor - индекс корректируемого датчика в массиве датчиков,
    2. Flag\_CORR\_ready – флаг готовности к корректировке датчика, индекс датчика установлен, можно начинать работу с датчиком,
    3. T\_CORR\_sensor - Текущая температура,
    4. HR\_CORR\_Sensor - Текущая влажность (сопротивление для типа 2),
    5. dT\_CORR\_sensor - Корректировка температуры,
    6. dHR\_CORR\_sensor – Корректировка влажности (сопротивления для типа 2),

Для вывода на экран изменений значений используется флаг из функции программирования - FlagCurrentValue\_PR\_sensor.

# Модуль ввода-вывода

В качестве модуля ввода-вывода используем китайский модуль MBSL 16DI16RO. Это 16 изолированных цифровых входа и 16 релейных выходов до 5А каждый.

Модуль ввода-вывода обеспечивает доступ к информации на входах и управлению выходами по шине RS485 ModBus RTU.

Модуль рассчитан на работу от 24 В, на это напряжение рассчитаны реле выхода. Но модуль хорошо отзывается на шину ModBus и при питании от 12 В, это удобно для целей программирования модуля, т.к. питание можно подать с колодки программирования.

Модуль на шине ModBus может получить адрес от 1 до 247. Адрес 0 используется для широковещательной передачи данных всем устройствам, а адреса 248–255 считаются зарезервированными согласно спецификации ModBus, их использование не рекомендуется.

Наш китайский модуль не реагирует на широковещательные адреса, ни на 0х00, ни на 0хFF.

Модуль содержит следующие типы регистров:

Output coil registers – регистры выходных реле. Имеется 16 регистров, каждый из которых обеспечивает 1 бит информации о состоянии одного выходного реле. Регистр можно считывать и записывать.

Input digital quantity – регистры состояния цифровых входов. Имеется 16 регистров, каждый из которых содержит 1 бит состояния одного цифрового входа.

Holding registers – регистры хранения. Регистры хранения содержат информацию о состоянии: входов, выходов, адрес устройства на шине, скорость обмена по шине, включение бита четности при обмене по шине, установка времени для защиты от отключения шины ModBus, версия прошивки модуля.

Каждый тип регистра имеет свой блок адресации. Адресные пространства регистров, также называемые таблицами или блоками, различны для всех типов регистров. Это значит, что значения регистров с одинаковым адресом, но разным типом, разные.

Адресное пространство каждого блока регистров – 16 разрядов, от 0 до 65535.

Чтобы обозначить в документации обращение к разным типам регистров используется нестандартная адресация. Признаки использования нестандартной адресации:

* + 1. Адреса записываются в десятичном формате.
    2. Во всех адресах пять или шесть цифр.
    3. Адреса с недискретными данными (показания датчиков и т.п.) начинаются на 30 или 40.

В нашем случае адреса начинаются с цифры (префикса):

* + 1. 0 для выходных регистров,
    2. 1 для входных регистров,
    3. 4 для регистров хранения.

При физическом обращении префикс адреса использовать не нужно, обращения к различным адресным блокам производится разными командами (функциями), обслуживающими только свой блок регистров.

Коды функций для различных блоков регистров:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Register type | PLC Address range | Function code | Description of the function code | Operating |
| Output coil registers | 00001-00016 | 0x01H | Read multiple coil registers | Read the value of one or more coil registers |
| 0x05H | Write a coil register | Write a coil register value |
| 0x0FH | Write one or more coil registers | Write the value of one or more coil registers |
| Holding registers | 40001-40016 | 0x03H | Read multiple holding registers | Read the value of one or more holding registers |
| 0x 06H | Write a single holding register | Write a data to holding register |
| 0x 10H | Write multiple holding registers | Write one or more data to holding register |
| Input digital quantity | 10001-10016 | 0x02H | Read input discrete | Discrete input register |

Описание регистров Output coil registers (function code: 0x01H,0x05H, 0x0FH):

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| PLC Address | Physical address | Parameter | Length | Read/Write | MIN | MAX | Description |
| 00001 | 0x0000 | DO1 | 1 | Read/Write | 0 | 1 | Status of digital output bit 1 |
| 00002 | 0x0001 | DO2 | 1 | Read/Write | 0 | 1 | Status of digital output bit 2 |
| 00003 | 0x0002 | DO3 | 1 | Read/Write | 0 | 1 | Status of digital output bit 3 |
| 00004 | 0x0003 | DO4 | 1 | Read/Write | 0 | 1 | Status of digital output bit 4 |
| 00005 | 0x0004 | DO5 | 1 | Read/Write | 0 | 1 | Status of digital output bit 5 |
| 00006 | 0x0005 | DO6 | 1 | Read/Write | 0 | 1 | Status of digital output bit 6 |
| 00007 | 0x0006 | DO7 | 1 | Read/Write | 0 | 1 | Status of digital output bit 7 |
| 00008 | 0x0007 | DO8 | 1 | Read/Write | 0 | 1 | Status of digital output bit 8 |
| 00009 | 0x0008 | DO9 | 1 | Read/Write | 0 | 1 | Status of digital output bit 9 |
| 00010 | 0x0009 | DO10 | 1 | Read/Write | 0 | 1 | Status of digital output bit 10 |
| 00011 | 0x000A | DO11 | 1 | Read/Write | 0 | 1 | Status of digital output bit 11 |
| 00012 | 0x000B | DO12 | 1 | Read/Write | 0 | 1 | Status of digital output bit 12 |
| 00013 | 0x000C | DO13 | 1 | Read/Write | 0 | 1 | Status of digital output bit 13 |
| 00014 | 0x000D | DO14 | 1 | Read/Write | 0 | 1 | Status of digital output bit 14 |
| 00015 | 0x000E | DO15 | 1 | Read/Write | 0 | 1 | Status of digital output bit-15 |
| 00016 | 0x000F | DO16 | 1 | Read/Write | 0 | 1 | Status of digital output bit-16 |

Описание регистров Discrete input registers (function code: 0x02H):

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| PLC Address | Physical address | Parameter | Length | Read/Write | MIN | MAX | Description |
| 10001 | 0x0000 | DI1 | 1 | Read | 0 | 1 | Status of digital input bit 1 |
| 10002 | 0x0001 | DI2 | 1 | Read | 0 | 1 | Status of digital input bit 2 |
| 10003 | 0x0002 | DI3 | 1 | Read | 0 | 1 | Status of digital input bit3 |
| 10004 | 0x0003 | DI4 | 1 | Read | 0 | 1 | Status of digital input bit4 |
| 10005 | 0x0004 | DI5 | 1 | Read | 0 | 1 | Status of digital input bit5 |
| 10006 | 0x0005 | DI6 | 1 | Read | 0 | 1 | Status of digital input bit6 |
| 10007 | 0x0006 | DI7 | 1 | Read | 0 | 1 | Status of digital input bit7 |
| 10008 | 0x0007 | DI8 | 1 | Read | 0 | 1 | Status of digital input bit8 |
| 10009 | 0x0008 | DI9 | 1 | Read | 0 | 1 | Status of digital input bit9 |
| 10010 | 0x0009 | DI10 | 1 | Read | 0 | 1 | Status of digital input bit10 |
| 10011 | 0x000A | DI11 | 1 | Read | 0 | 1 | Status of digital input bit11 |
| 10012 | 0x000B | DI12 | 1 | Read | 0 | 1 | Status of digital input bit12 |
| 10013 | 0x000C | DI13 | 1 | Read | 0 | 1 | Status of digital input bit13 |
| 10014 | 0x000D | DI14 | 1 | Read | 0 | 1 | Status of digital input bit 14 |
| 10015 | 0x000E | DI15 | 1 | Read | 0 | 1 | Status of digital input bit15 |
| 10016 | 0x000F | DI16 | 1 | Read | 0 | 1 | Status of digital input bit16 |

Описание регистров Holding registers (function codes: 0x03H, 0x06H, 0x10H)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| PLC Address | Physical address | Parameter | Length | Read/Write | MIN | MAX | Description |
| 40001 | 0x0000 | DI(1~8) | 2 | Read | 0 | 0xff | Read the status of 8-bit digital input |
| 40002 | 0x0001 | DI(9~16) | 2 | Read | 0 | 0xff | Read the status of 8-bit digital input |
| 40003 | 0x0002 | Device address | 2 | Read/Write | 1 | 247 | 1(default) |
| 40004 | 0x0003 | Baud rate | 2 | Read/Write | 1 | 5 | 1(4800)  2(9600) default  3(19200)  4(38400)  5(57600)  6(115200) |
| 40005 | 0x0004 | Check Digit | 2 | Read/Write | 1 | 3 | 1 (no check) default  2 (odd check)  3 (even check) |
| 40006 | 0x0005 | Time setting for network disconnection protection | 2 | Read/Write | 1 | 2000 | If set to> = 1800, the network disconnection protection function will be cancelled.  0-1799S range is the network disconnection protection time, DO output reset beyond this range |
| 40007 | 0x0006 | product version | 2 | Read | 0 | -- | Year + month + day |
| 40009 | 0x0008 | DO(1~8) | 2 | Read/Write | 0 | 0xff | Write 8-bit digital output status |
| 40010 | 0x0009 | DO(9~16) | 2 | Read/Write | 0 | 0xff | Write 8-bit digital output status |

В текущей работе программного обеспечения дефростера будут использоваться регистры состояния оборудования, в которых каждый бит регистра состояния будет означать состояние вкл/выкл единицы оборудования. Управлять оборудованием в этом случае удобнее, передавая в модуль ввода-вывода регистр состояния целиком. Потому в текущей работе будем использовать только регистры Holding registers – регистры хранения.

Формат протокола ModBus RS485.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Направление | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Host sends (TX, Read), чтение нескольких регистров | Client address,  1 Byte | Read multiply Code, 0x03,  1 Byte | Register №1 address,  2 Bytes | Number of registers, 2 Bytes | CRC 16 verification, 2 Bytes |  |  |
| Client replies (RX, Read) | Client address,  1 Byte | Read multiply Code, 0x03,  1 Byte | Number of data bytes (N), 1 Byte | Data,  N Bytes | CRC 16 verification, 2 Bytes |  |  |
| Host sends (TX, single Write), запись одного регистра | Client address,  1 Byte | Write single Code, 0x06,  1 Byte | Register address,  2 Bytes | Data,  2 Bytes | CRC 16 verification, 2 Bytes |  |  |
| Client replies (RX, single Write) | Client address,  1 Byte | Write Code 0x06,  1 Byte | Register address,  2 Bytes | Data,  2 Bytes | CRC 16 verification, 2 Bytes |  |  |
| Host sends (TX, multiply Write), запись нескольких регистров | Client address,  1 Byte | Write multiply Code, 0x10,  1 Byte | Register №1 address,  2 Bytes | Number of registers, 2 Bytes | Data number,  1 Byte | Data,  N Bytes | CRC 16 verification, 2 Bytes |
| Client replies (RX, multiply Write) | Client address,  1 Byte | Write Code 0x10,  1 Byte | Register №1 address,  2 Bytes | Number of registers, 2 Bytes | CRC 16 verification, 2 Bytes |  |  |

Команда 0x0F

Эта команда используется для записи нескольких значений дискретного выхода DO.

Пример записи в несколько DO с регистрами от #20 до #29 для SlaveID адреса устройства 17. Адрес регистра будет 0013 hex = 19, т.к. счет ведется с 0 адреса.

|  |  |  |  |
| --- | --- | --- | --- |
| Байт | Запрос | Байт | Ответ |
| (Hex) | Название поля | (Hex) | Название поля |
| 11 | Адрес устройства | 11 | Адрес устройства |
| 0F | Функциональный код | 0F | Функциональный код |
| 00 | Адрес первого регистра Hi байт | 00 | Адрес первого регистра Hi байт |
| 13 | Адрес первого регистра Lo байт | 13 | Адрес первого регистра Lo байт |
| 00 | Количество регистров Hi байт | 00 | Кол-во записанных рег. Hi байт |
| 0A | Количество регистров Lo байт | 0A | Кол-во записанных рег. Lo байт |
| 02 | Количество байт далее | 26 | Контрольная сумма CRC |
| CD | Значение байт DO 27-20 (1100 1101) | 99 | Контрольная сумма CRC |
| 01 | Значение байт DO 29-28 (0000 0001) |  |  |
| BF | Контрольная сумма CRC |  |  |
| 0B | Контрольная сумма CRC |  |  |

В ответе возвращается количество записанных регистров.

# Работа с модулем ввода-вывода

Модуль ввода-вывода используется так же, как и датчики температуры. Для модуля ввода-вывода создан датчик типа 4 в структуре Sensor\_type[STQ] и присвоен адрес устройства в структуре Sensor\_array[SQ].

Опрос модуля ввода-вывода выполняется в едином цикле опроса датчиков. Результат опроса – два байта, считанные отдельными словами, объединяются в одно слово с типом данных uint\_16t, которое записывается в переменную Model::T\_CORR\_sensor вместо температуры.

# Программирование модуля ввода-вывода

Модуль ввода-вывода, как и датчики температуры, требует программирования адреса устройства на шине и скорости обмена по шине.

Программирование модуля ввода-вывода проводится в окне программирования датчиков для датчика типа 4.

Защита от отключения выхода. Когда подчиненная станция (модуль) обнаруживает, что главная станция (сервер) не отправляла данные более установленного времени, она автоматически сбрасывает выходной сигнал для защиты безопасности полевого оборудования. Функцию непрерывной защиты сети по умолчанию можно выбрать, установив время большее 1800 секунд.

Модуль никак не реагирует на широковещательный адрес 0xFF! Поэтому для поиска нужно будет перебирать все адреса от 1 до 247. Чтобы поиск устройства происходил быстрее, считаем, что больше двух устройств ввода-вывода не понадобится, выделяем для них адреса в начале адресного пространства, начиная с адреса 02 до 03. Адрес 01 устанавливается на заводе. Сканирование проводим по адресам от 01 до 03.

Можно сбросить модуль до заводских настроек ( адрес 01, скорость 2 (9600)) путём нажатия кнопки RESET. Кнопка расположена на плате рядом с красным светодиодом RUN. Нужно нажать и держать кнопку, включить питание, продолжать держать кнопку более 3 секунд. Индикатор RUN мигнет 3 раза с интервалом 1 секунда и загорится постоянным светом. Теперь кнопку можно отпустить. Модуль вернется к заводским настройкам после отключения питания.

На команду чтения 0x03 независимо от количества запрошенных регистров всегда выдаёт два байта от одного регистра.

Модуль поддерживает сообщения информирования клиента (мастера) ModBus об исключительных ситуациях (Exception). Формат возвращаемых пакетов полностью соответствует документу «ModBus Application Protocol Specification v1.1a». Сообщения об исключительных ситуациях возникают только на запросы, адресованные данному устройству с правильным значением LRC или CRC пакета.

Когда модуль обнаруживает одну их этих ошибок, он посылает ответное сообщение, содержащее адрес, код функции, код ошибки и контрольную сумму. Для указания на то, что ответное сообщение – это уведомление об ошибке, старший бит поля кода функции устанавливается в 1.

Запрос: 01:03:9С:43:00:02:1B:8F – запрос чтения 2-х регистров у устройства с адресом 01, начиная с регистра 9C43h (40003d).

Ответ: 01:83:02:С0:А1 – frame contains error. Мы видим, что в ответном коде команды 03 появился установленным старший бит, модуль вернул код 83, а это признак ошибки. Код исключительной ситуации – 02. Неподдерживаемый адрес регистра.

Коды исключительных ситуаций:

* + 1. 01 – неподдерживаемая команда. Возникает при запросе от устройства команды, которую устройство не поддерживает.
    2. 02 – неподдерживаемый адрес данных. Возникает только при запросе с адресом данных, которых нет в таблицах соответствия между адресами Modbus и внутренней памятью устройства.
    3. Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, снимок экрана

       Автоматически созданное описание03 – неверное количество данных. В запросе содержится значения, недопустимые для устройства. Например, запрос количества регистров более чем 120.

При записи новых адреса и скорости в регистры модуля переключение модуля на новые параметры произойдёт только после отключения питания и включения вновь. До снятия питания модуль отвечает при обращении по старому адресу на старой скорости, но при ответе при обращении по старым параметрам, модуль передает из регистров новые параметры.

# Управление модулем ввода-вывода в рабочем режиме

Значения регистра ввода модуля ввода-вывода будут сохраняться в массиве данных класса Sensor в 16-разрядном поле T (температура).

Значения регистра вывода модуля ввода-вывода будут сохраняться в массиве данных класса Sensor в поле 16-разрядном H (влажность).

Регистры ввода и регистры вывода у модуля ввода-вывода 16-разрядные, но значащим является только младший байт, поэтому перед записью младший и старший байты данных нужно соединить в одно 16-разрядное число.

В общем цикле опроса датчиков выполняем три запроса к модулю ввода-вывода: сначала запись в регистр состояния выходных реле, а затем считывание входных данных и считывание данных из регистра состояния выходных реле.

Сначала выполняем запись в регистр состояния выходных реле.

Для считывания входных данных и данных из регистра выходных реле делаем два запроса на считывание. Считанные байты записываем в соответствующие поля массива данных класса Sensor. Входные данные – в поле T, выходные данные (управление реле) – в поле H.

При запросе на считывание однобитовых портов устройство сначала передаёт содержимое младших разрядов, а затем – старших.

# Программирование управления периферийными устройствами

Регистр аппаратного управления устройствами организован в Modul.hpp.

Созданы три регистра:

* + 1. DFR – регистр аппаратного управления устройствами,
    2. DFR\_current – текущее состояние регистра управления устройствами, отображаемое на экране контроллера,
    3. DFR\_chng\_flag – регистр флагов изменения состояния устройств. Бит, соответствующий устройству, установится в «1», когда состояние устройства в регистре аппаратного управления изменится относительно текущего состояния устройства, выводимого на экран контроллера. Регистр нужен для обновления отображения на экране.

Все регистры имеют битовую структуру и определяются типом DFR\_REGISTERS\_t.

Для управления устройствами в программе производится обращение к конкретному именованному биту, выделенному для управления конкретным устройством.

Для передачи регистра управления в модуль ввода-вывода используется 16-разрядная переменная RelayRegister, в которую загружается число, являющееся двоичной копией регистра управления DFR.

Загрузка регистра DFR в модуль ввода-вывода производится только если есть изменения регистра по сравнению с текущим состоянием DFR\_current в каждом основном цикле опроса датчиков.

Состояние DFR сравнивается с текущим состоянием устройства ввода-вывода, которое содержит регистр DFR\_current, и изменившиеся биты DFR устанавливают в единицу биты регистра флагов изменения состояния устройств DFR\_chng\_flag.

Изменения флагов регистра DFR\_chng\_flag обрабатывает процедура VisualizationPresenter::ValUpdatePresenter() при открытии окна Visualization. Процедура ValUpdatePresenter() запускает процедуры изменения отображения устройств на экране контроллера в VisualizationView.cpp.

# Ручное управление устройствами дефростера

Ручное управление дефростером требуется только для диагностики оборудования. Использование дефростера в ручном режиме категорически не рекомендуется, поскольку дефростируемый продукт с высокой вероятностью будет испорчен.

При ручном управлении в регистр аппаратного управления устройствами DFR записывается информация из регистра ручного управления DFR\_manual.

Биты состояния оборудования в регистре ручного управления устанавливаются переключателями вручную с экрана контроллера.

Ручное управление начинается с общего переключателя режима работы – автоматическое и ручное управление. При включении ручного управления устанавливается флаг Model::Flag\_DFR\_manual, который переключает источник данных для загрузки регистра аппаратного управления устройствами DFR.

При включении ручного режима становятся доступны управление воротами и клапаном подачи воды в форсунки увлажнителя. Также становится доступным управление вентиляторами и тэнами на следующей странице меню контроллера.

Включение любого тэна станет возможным только после включения соответствующего вентилятора.

Нажатие кнопок включения устройств на экране контроллера приводит к установке в единицу бита управления соответствующим устройством в регистре DFR\_manual.

# Управление роллерными воротами

Управление роллерными воротами осуществляется по трёхкнопочной схеме:

* + 1. Вверх.
    2. Вниз.
    3. Стоп.

Нажатием на кнопку запускается процесс подъёма, опускания или остановка движения ворот. Движение ворот производится либо до срабатывания концевого выключателя, либо до нажатия на кнопку СТОП.

Для каждой кнопки движения ворот подсвечиваются две лампы: лампа команды движения и лампа конечного положения. Нажатие на кнопку движения устанавливает бит команды движения и зажигает лампу движения белого цвета. Достижение воротами конечного положения устанавливает бит состояния концевого выключателя и зажигает лампу конечного положения зелёного цвета.

При одновременно установленных битах движения и конечного положения команда движения обнуляется, лампа движения гаснет. Ворота останавливаются. Лампа конечного положения остаётся зажжённой.

Из верхнего конечного положения ворота могут двигаться только вниз при нажатии на кнопку ВНИЗ. Из нижнего конечного положения ворота могут двигаться только вверх при нажатии на кнопку ВВЕРХ. В конечных положениях соответствующие положению кнопки заблокированы и не реагируют на нажатия.

Кнопка СТОП не имеет лампы движения. Есть только лампа остановки красного цвета, которая зажигается при нажатии на кнопку СТОП, в этот момент движение ворот останавливается. Состояние СТОП можно разблокировать нажатием на одну из кнопок ВВЕРХ или ВНИЗ.

При поступлении на модуль ввода-вывода сигнала срабатывания аварийной кнопки ворота переходят в состояние СТОП, аналогично нажатию на кнопку СТОП. Обработка аварийной остановки производится как в режиме ручного, так и автоматического управления.

Поступление сигнала срабатывания аварийной кнопки сопровождается зажиганием лампы «Аварийная кнопка» на экране контроллера. Кнопки управления воротами блокируются до снятия аварийного сигнала.

При поступлении на модуль ввода-вывода сигнала аварийного открытия ворот ворота открываются в режиме, аналогичном нажатию на кнопку ВВЕРХ. Обработка аварийного открытия ворот производится как в режиме ручного, так и автоматического управления.

Поступление сигнала аварийного открытия ворот сопровождается зажиганием лампы «Аварийное открытие» на экране контроллера. Кнопки управления воротами блокируются до снятия аварийного сигнала.

Сигнал аварийного открытия ворот подаётся кнопкой изнутри дефростера или датчиком препятствия. Датчик препятствия может быть:

* + 1. Механическим, срабатывающим от натяжения шнура, натянутого вдоль кромки ворот,
    2. Оптическим, срабатывающим при пересечении посторонним предметом светового луча вдоль кромки ворот,
    3. Электронным, срабатывающим от превышения порогового значения потребляемого тока электродвигателем привода ворот.

При срабатывании автоматического выключателя питания модуля управления воротами на экране контроллера зажигается лампа «Авария электропитания», кнопки управления воротами блокируются до снятия сигнала аварии.

Состояние концевых выключателей считывается модулем ввода-вывода в переменную Model::DFR\_gate\_flags.

# Сигналы управления немецким дефростером

Все сигналы управления, формируемые микроконтроллером и подаваемые на микроконтроллер, имеют напряжение 24 В.

Входные сигналы на микроконтроллер:

* + 1. Дополнительные секции концевых выключателей дают сигнал микроконтроллеру о достижении воротами конечного положения. На схеме модуль М100, Endschalter Rolltor Auf/Zu, 7/1, 7/2, X12:2, X12:3. E124:0, E124:1.
    2. Сигнал промежуточной остановки движения ворот по кнопке. На схеме модуль М110, Endschalter wrasenklappe Auf/Zu, 7/5, 7/6, X12:6, X12:7. E124:4, E124:5.
    3. Сигнал аварийного открывания ворот по аварийному шнуру двери. На схеме Notöffnung Rolltor Auf/Zu, 7/7, X12:8. E124:6.
    4. Работа нагревателей (ТЭНов), формируется контактами реле включения нагревателей К10, К20, К30, К40. Heizung Betrieb 1…4, 8/1…8/4, X11:22…X11:24. E125:0…E125:3.
    5. Сигнал неисправности ворот формируется дополнительным контактом автомата питания модуля М100 управления воротами. Rolltor störung, 9/1, X12:12. E126:0.
    6. Работа вытяжного вентилятора, формируется контактом реле включения вентилятора, Wresenablufter Betrieb, 9/2, X12:13. E126:1.
    7. Работа циркуляционных вентиляторов, формируется контактами реле включения вентиляторов, Ventilator Betrieb 1…4, 9/3, X12:14… X12:17. E126:2…E126:5.
    8. Кнопка СТАРТ, 9/7, X12:18. E126:6.
    9. Кнопка СТОП, 9/8, X12:19. E126:7.

Выходные сигналы микроконтроллера:

* + 1. Включить вентилятор вытяжки. Wresenablufter Ein, 10/1, реле К200.
    2. Включить вентилятор 1…4. Ventilator 1…4, 10/2, реле К201…К204.
    3. Включить водные форсунки, Magnetventil spruh wasser, 10/6, реле К205.
    4. Закрыть защитную заслонку вытяжного вентилятора. Wrasenklappe schlieben, 10/8, реле К207.
    5. Подать звуковой сигнал. Signal horn, 11/1, реле К208.
    6. Открыть ворота. Rolltor aufmachen, 11/2, реле К209.
    7. Остановить движение ворот. Rolltor stop, 11/3, реле К210.
    8. Закрыть ворота. Rolltor zumachen, 11/4, реле К211.
    9. Включить ТЭН. Heizung 1…4, 11/6, реле К212…К214.
    10. Зеленая лампа аварийной работы ворот Notöffnung Rolltor, 13/1, X3:11, X3:12.
    11. Зеленая лампа операции размораживания, auftauhen betrieb, 13/2.
    12. Красная лампа нарушения размораживания, auftauhen störung, 13/3.

# Передача данных на сервер

Передача данных осуществляются по сети TCP/IP через модуль-конвертер ModBus – TCP/IP Elfin EW11. Особенностью настроек конвертера является то, что он осуществляет «прозрачную» передачу всей информации с шины ModBus в TCP/IP и обратно. Конвертер настроен на режим client в TCP/IP, в настройках устанавливается адрес и порт сервера. На стороне UART настройки такие: 19200, 8, 1, None. Protocol None.

В сервер транслируется информация с шины программирования датчиков. Контроллер настраивается так, чтобы в рабочем режиме информация для передачи на сервер транслировалась на шину программирования.

Для трансляции на шину запускается отдельная задача – поток TX\_To\_Server. В потоке работает в бесконечном цикле функция TX\_ToServer ().

Собранные с датчиков данные собираются в массив данных и передаются из потока ReadData в поток TX\_To\_Server посредством очереди Data\_Queue.

Чтобы поток TX\_To\_Server не работал впустую и не расходовал зря ресурсы процессора, будем использовать параметр osWaitForever. Функция получения сообщения из очереди будет ожидать пополнения очереди бесконечно долго, всё время до этого события поток TX\_To\_Server будет стоять, и только после пополнения очереди продолжит работу и прочитает данные из очереди.

Собственно передачей данных занимается функция WriteToServer(), которая находится в файле ModBus.cpp, потому что в этом файле собраны все функции, работающие с ModBus.

# Функция HAL\_StatusTypeDef HAL\_UARTEx\_ReceiveToIdle\_DMA

HAL\_StatusTypeDef HAL\_UARTEx\_ReceiveToIdle\_DMA(UART\_HandleTypeDef \*huart, uint8\_t \*pData, uint16\_t Size)

/\*\*

\* @brief Receive an amount of data in DMA mode till either the expected number of data is received or an IDLE event occurs.

\* @note Reception is initiated by this function call. Further progress of reception is achieved thanks to DMA services, transferring automatically received data elements in user reception buffer and calling registered callbacks at half/end of reception. UART IDLE events are also used to consider reception phase as ended. In all cases, callback execution will indicate number of received data elements.

\* @note When the UART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position).

\* @note When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M = 01), the received data is handled as a set of uint16\_t. In this case, Size must indicate the number of uint16\_t available through pData.

\* @param huart UART handle.

\* @param pData Pointer to data buffer (uint8\_t or uint16\_t data elements).

\* @param Size Amount of data elements (uint8\_t or uint16\_t) to be received.

\* @retval HAL status

\*/

/\*\* @brief Принимает объем данных в режиме DMA до тех пор, пока не будет получено ожидаемое количество данных или не произойдет событие ПРОСТОЯ.

\* @примечание Прием данных инициируется вызовом этой функции. Дальнейший прогресс в приеме достигается благодаря сервисам DMA, которые автоматически переносят полученные элементы данных в пользовательский буфер приема и вызывают зарегистрированные обратные вызовы в половине/конце приема. События простоя UART также используются для того, чтобы считать фазу приема завершенной. Во всех случаях при выполнении обратного вызова будет указано количество полученных элементов данных.

\* @примечание: когда функция контроля четности UART включена (PCE = 1), принимаемые данные содержат бит контроля четности (позицию MSB).

\* @примечание: когда четность UART не включена (PCE = 0), а длина слова настроена на 9 бит (M = 01), полученные данные обрабатываются как набор uint16\_t. В этом случае размер должен указывать на количество uint16\_t, доступных через pData.

\* @param\_art дескриптор UART.

\* @param\_data Указатель на буфер данных (элементы данных uint8\_t или uint16\_t).

\* @param Size Количество элементов данных (uint8\_t или uint16\_t), которые должны быть получены.

\* @retval Статус проверки

\*/

HAL\_UARTEx\_ReceiveToIdle\_DMA(&huart2,modbustcp->PROTOCOL\_BUFFER,MODBUSTCP\_MAX\_RECV\_SIZE); //Связь по ETH MODBUSTCP  
\_\_HAL\_DMA\_DISABLE\_IT(&hdma\_usart2\_rx, DMA\_IT\_HT); //Отключаем срабатывание на полузаполнения массива