Реализация протокола MODBUS RTU на микроконтроллерах семейства STM32

А. Ю. Савинков, email: savinkov_a_yu@sc.vsu.ru

Федеральное государственное бюджетное образовательное учреждение высшего образования «Воронежский государственный университет» (ФГБОУ ВО «ВГУ»)

Аннотация. *В данной работе предложена эффективная реализация протокола MODBUS RTU на микроконтроллере семейства STM32 с использованием расширенных аппаратных возможностей приемопередатчика UART микроконтроллера.*

Ключевые слова: *протокол MODBUS, MODBUS RTU, UART, RS-485, микроконтроллер, STM32.*

Введение

MODBUS – коммуникационный протокол для связи между электронными устройствами. Первая версия спецификаций MODBUS была опубликована в 1979 году. Протокол был разработан американской компанией Modicon (MOdular DIgital CONtroller – модульные цифровые контроллеры), созданной в 1968 году специально для разработки электронных модулей управления гидравлической трансмиссией GM Hydramatic. С 1996 года компания Modicon принадлежит французской корпорации Schneider Electric.

В настоящее время протокол MODBUS широко применяется в промышленности и находит все более широкое применение в системах умного дома и интернета вещей, в связи с чем часто возникает необходимость реализации протокола MODBUS в микроконтроллере. Наибольшее распространение получила разновидность протокола MODBUS RTU, реализующая бинарную передачу данных по линии RS-485 с использованием UART микроконтроллера, поэтому в дальнейшем будем иметь в виду именно эту разновидность протокола.

Ввиду архитектурных и аппаратных различий (разная периферия) применяемых микроконтроллеров невозможно сделать единую унифицированную реализацию MODBUS RTU, поэтому разработчику программного обеспечения (ПО) приходится делать собственную реализацию для каждого частного случая. Для упрощения этой задачи разработано множество библиотек MODBUS, например [2. 1, 2. 2, 2. 3]. Идея таких библиотек состоит в абстрагировании от аппаратных особенностей UART микроконтроллера с помощью специального драйвера и реализации протокола MODBUS на абстрактном интерфейсе

UART. Теперь для поддержки MODBUS достаточно реализовать только драйвер UART для используемого микроконтроллера. Преимущества такого подхода состоит в его универсальности, но ему также свойственен ряд существенных недостатков:

– для расширения применимости библиотеки интерфейс абстрактного UART предельно минимизируется, в результате теряется возможность использования аппаратных возможностей конкретного микроконтроллера;

– отказ от использования аппаратных возможностей UART микроконтроллера, например от возможности аппаратного обнаружения таймаута передачи, увеличивает объем и вычислительную сложность программы, что может быть критично в условиях ограниченности доступного объема памяти и быстродействия микроконтроллера;

– необходимость использовать драйвер UART увеличивает объем программного кода, что может быть критично в условиях ограниченного объема памяти микроконтроллера.

В докладе предлагается вариант реализации протокола MODBUS RTU для ведомого устройства на основе микроконтроллера семейства STM32, не требующий драйвера и максимально использующий аппаратные возможности микроконтроллера. Предложенный подход применим только для микроконтроллеров с поддержкой прямого доступа к памяти (DMA) приемопередатчика UART, аппаратного управления приемопередатчиком RS-485 и аппаратного детектирования отсутствия принимаемых данных на интерфейсе UART в течение заданного времени. Микроконтроллеры семейства STM32 получили очень широкое применение в системах автоматики и интернета вещей и многие из них предоставляют необходимые аппаратные возможности, поэтому предлагаемое решение обладает достаточной общностью и может быть широко использовано.

1. Общие сведения о протоколе MODBUS

Протокол MODBUS в настоящее время развивается Modbus Organization [2. 4], которая позиционирует себя как группу независимых пользователей и поставщиков устройств автоматизации. На официальном сайте Modbus Organization можно скачать последнюю версию спецификацию протокола MODBUS [2. 5], а также спецификации для различных физических каналов передачи, в том числе для MODBUS RTU через RS-485 [2. 6].

Протокол MODBUS построен на основе модели ведущий-ведомый. При использовании MODBUS RTU ведомые устройства имеют уникальные адреса на линии RS-485 в диапазоне 1…247. Ведущее устройство адреса не имеет. Ведомое устройство представлено как

набор адресуемых регистров четырех типов (см. таблицу), которые могут читаться или записываться ведущим устройством.

Тип регистра	Разрядность регистра, бит	Вид доступа
Дискретный вход (Discrete Input)	1	Только чтение
Флаг (Coil)	1	Чтение и запись
Регистр ввода (Input Register)	16	Только чтение
Регистр хранения (Holding Register)	16	Чтение и запись

При необходимости, для представления данных ведомого устройства могут использоваться несколько регистров, например, если датчик устройства (например, термометр) предоставляет результаты в виде float32, то для чтения данных измерения может использоваться пара регистров ввода, при этом размещение числа float32 в двух регистрах (порядок байт) не регламентируется стандартом. Чтение и запись регистров может выполняться как индивидуально (по одному), так и блоками смежных регистров, начиная с заданного. Ввиду ограничений формата данных MODBUS RTU за одну операцию может быть прочитано (записано) не более 2000 (1968) однобитных регистров и не более 125 (123) 16-битных регистров.

Формат пакета данных MODBUS RTU показан на рис. 1.

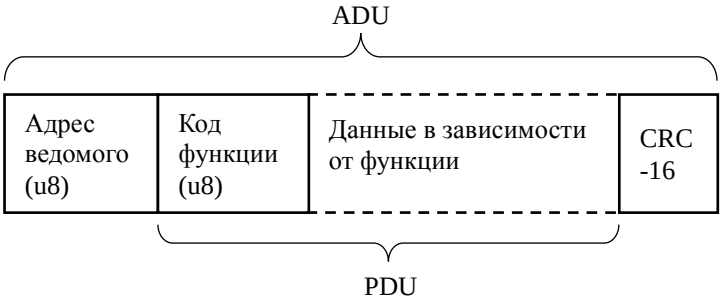

Рис. 1. Структура кадра MODBUS RTU.

Полный пакет, передаваемый по физическому каналу, включающий адрес ведомого и CRC, называют ADU – Application Data Unit. ADU инкапсулирует PDU – Protocol Data Unit – данные MODBUS, не зависящие от среды передачи. Формат и размер PDU определяется

кодом функции и числом регистров, которые должны быть записаны или прочитаны.

Текущая версия спецификации MODBUS определяет 21 функцию ведомого устройства. Обязательная реализация всех функций не требуется, устройство может реализовать только часть функций, требуемых для его работы. Наиболее часто реализуются следующие функции доступа к регистрам:

– чтение группы флагов (Read Coils) / код функции 1;

– чтение группы дискретных входов (Read Discrete Inputs) / код функции 2;

– чтение группы регистров хранения (Read Holding Registers) / код функции 3;

– чтение группы регистров ввода (Read Input Registers) / код функции 4;

– запись одного флага (Write Single Coil) / код функции 5;

– запись одного регистра хранения (Write Single Register) / код функции 6;

– запись группы флагов (Write Multiple Coils) / код функции 15 (0x0f);

– запись группы регистров хранения (Write Multiple Registers) / код функции 16 (0x10).

Формат PDU запроса и ответа для функций чтения регистров (функции 1-4) показан на рис. 2.

Запрос ведущего устройства:

Код функции (u8)	Адрес первого регистра: от 0 до 65535 (u16)	Число регистров: от 1 до 2000 для однобитных или до 125 для 16-битных регистров (u16)

Ответ ведомого устройства:

Код функции (u8)	N: от 1 до 125 (u8)	Данные ответа: N байт для однобитных или 2N байт для 16-битных регистров

Рис. 2. PDU запроса и ответа для функций чтения регистров.

Данные однобитных регистров побитно упаковываются в байты ответа в порядке от младшего бита к старшему. Первый байт ответа содержит значения первых 8 регистров из диапазона запроса, второй байт – следующие 8 регистров и т.д. Данные 16-битных регистров передаются с использованием двух байт, первый байт ответа содержит 8 старших битов первого регистра из диапазона запроса, второй байт – оставшиеся 8 младших битов регистра и т.д.

Формат PDU запроса для функций записи одного регистра (функции 5, 6) показан на рис. 3.

Код функции (u8)	Адрес регистра: от 0 до 65535 (u16)	Код 0 или 0xff00 для однобитного регистра или значение от 0 до 65536 для 16-битного регистра (u16)

Рис. 3. PDU запроса для функций записи одного регистра.

Для очистки флага используется код 0, а для установки – 0xff00. Значение регистра хранения записывается явно в виде 16-битного числа. Ответ ведомого устройства является копией запроса.

Формат PDU запроса и ответа для функций записи группы регистров (функции 15, 16) показан на рис. 4.

Устанавливаемые состояния флагов побитно упаковываются в байты запроса в порядке от младшего бита к старшему. Первый байт запроса содержит установки первых 8 флагов из диапазона запроса, второй байт – следующие 8 флагов и т.д. Данные для записи в16-битные регистры хранения передаются с использованием двух байт, первый байт ответа содержит 8 старших битов данных для первого регистра из диапазона запроса, второй байт – оставшиеся 8 младших битов данных и т.д.

2. Реализация протокола MODBUS

Реализация содержит ядро, выполняющее прием и передачу данных через UART, проверку и контрольной суммы, проверку адреса, извлечение управляющих данных запросов (адреса и число регистров), вызов внешних функций работы с данными (зависят от устройства, реализуются как функции обратного вызова) и отправку ответа

ведущему устройству. Предполагается использование STM32 HAL и операционной системы FreeRTOS.

Запрос ведущего устройства:

Код функции (u8)	Адрес первого регистра: от 0 до 65535 (u16)	Число регистров: от 1 до 1968 для однобитных или до 123 для 16-битных регистров (u16)	N (u8)	Данные для записи: N байт для однобитных или 2N байт для 16-битных регистров

Ответ ведомого устройства:

Код функции (u8)	Адрес первого регистра: от 0 до 65535 (u16)	Число регистров: от 1 до 1968 для однобитных или до 123 для 16-битных регистров (u16)

Рис. 4. PDU запроса и ответа для функций записи группы регистров.

Интерфейс ядра MODBUS приведен в листинге 1.

Листинг 1

```
/* Контекст устройства и функция инициализации */
typedef struct {
  UART_HandleTypeDef* huart; /* UART микроконтроллера */
  uint8_t address; /* адрес данного устройства */
  MODBUS_RegAccess_Proc_t read_coils_proc;
  MODBUS_RegAccess_Proc_t read_discrete_inputs_proc;
  MODBUS_RegAccess_Proc_t read_holding_registers_proc;
  MODBUS_RegAccess_Proc_t read_input_registers_proc;
  MODBUS_RegAccess_Proc_t write_coils_proc;
  MODBUS_RegAccess_Proc_t write_holding_registers_proc;
} MODBUS_Context_t;
void MODBUS_Init(MODBUS_Context_t* context);

/* Главная функция ядра MODBUS
MODBUS_Status_t MODBUS_ProcRequest();
```

Таким образом, для реализации ведомого устройства MODBUSA RTU на микроконтроллере достаточно определить функции обратного вызова для реализации функций MODBUS, инициализировать UART,

заполнить контекст (экземпляр структуры MODBUS_Context_t) и вызвать функцию инициализации ядра MODBUS. Нет необходимости устанавливать все функции обратного вызова. При попытке использовать функцию MODBUS, для которой не установлена функция обратного вызова, ядро MODBUS автоматически отправит ответ ведущему устройству с кодом ошибки Illegal Function.

Функция MODBUS_Init() устанавливает обработчики событий UART для STM32 HAL и настраивает обработку таймаута при приеме данных, чтобы определить окончание передачи пакета MODBUS. Согласно спецификации MODBUS, при передаче пакета данных не должно быть пауз, длительностью более одного символа, а соседние пакеты должны разделяться промежутком не менее 3.5 символов. Код функций обработки событий UART приведен в листинге 2.

Листинг 2

```
/* Функции обратного вызова UART */
void MODBUS_RxEventCallback(UART_HandleTypeDef *huart,
  uint16_t Size) {
  MODBUS_receiver_counter += Size;
  HAL_UARTEx_ReceiveToIdle_DMA(MODBUS_context->huart,
    MODBUS_adu_buffer + MODBUS_receiver_counter,
    MODBUS_MAXIMUM_ADU_SIZE - MODBUS_receiver_counter);
}
void MODBUS_TransmitCpltCallback(UART_HandleTypeDef *huart) {
  UNUSED(huart);
  osSignalSet(MODBUS_execution_thread,
    MODBUS_THREAD_WAKEUP_SIGNAL);
}
void MODBUS_ErrorCallback(UART_HandleTypeDef *huart) {
  SET_BIT(huart->Instance->ICR,
    USART_ICR_PECF | USART_ICR_FECF | USART_ICR_NCF |
    USART_ICR_ORECF | USART_ICR_RTOCF);
  HAL_UART_AbortReceive(huart);
  osSignalSet(MODBUS_execution_thread,
    MODBUS_THREAD_WAKEUP_SIGNAL);
}
```

Вся функциональность ядра MODBUS реализуется функцией MODBUS_ProcRequest(), которая должна вызываться в цикле, желательно в отдельной задаче FreeRTOS, например, как показано в листинге 3.

Листинг 3

```
void StartRtosTask(void const * argument) {
  static MODBUS_Context_t mb_context = {
    .huart = &huart1,
    .read_holding_registers_proc = ReadHoldingRegisters,
    .write_holding_registers_proc = WriteHoldingRegisters,
```

```
    .read_input_registers_proc= ReadInputRegisters
  };
  mb_context.address =100;
  MODBUS_Init(&mb_context);

  for(;;) {
    MODBUS_ProcRequest();
  }
}
```

Фрагмент кода функции MODBUS_ProcRequest() приведен в листинге 4.

Листинг 4

```
/* обрабатывает принятый пакет и отправляет ответ */
MODBUS_Status_t MODBUS_ProcRequest() {
…
  /* очищаем счетчик принятых данных и флаги ошибок UART */
  MODBUS_receiver_counter = 0;
  SET_BIT(MODBUS_context->huart->Instance->ICR,
    USART_ICR_PECF | USART_ICR_FECF | USART_ICR_NCF |
    USART_ICR_ORECF | USART_ICR_RTOCF);

  /* очищаем регистр принятых данных UART */
  uint8_t rd = (uint8_t)READ_REG(
    MODBUS_context->huart->Instance->RDR); UNUSED(rd);

  /* прием данных через DMA до конца пакета (определяется по
таймауту приемника), но не более максимально возможного
размера пакета */
  HAL_UARTEx_ReceiveToIdle_DMA(MODBUS_context->huart,
    MODBUS_adu_buffer, MODBUS_MAXIMUM_ADU_SIZE);

  /* блокируем вызывающий поток до приема пакета */
  while (osSignalWait(MODBUS_THREAD_WAKEUP_SIGNAL,
    osWaitForever).status != osEventSignal) {}

  /* проверка принятого пакета */
  if ((READ_REG(MODBUS_context->huart->Instance->ISR) &
    (USART_ISR_PE | USART_ISR_FE | USART_ISR_NE |
    USART_ISR_ORE)) != 0) {
    return MODBUS_Status_UartError;
  }
  if (MODBUS_Crc16(MODBUS_adu_buffer,
    MODBUS_receiver_counter) != 0) {
    return MODBUS_Status_CrcError;
  }
  if (MODBUS_adu_buffer[0] != MODBUS_context->address) {
    return MODBUS_Status_AddressMismatch;
  }

  /* селектор функций */
```

```
  switch (MODBUS_adu_buffer[1]) {

  /* функции чтения регистров */
  case MODBUS_FUNCTION_READ_COILS:
  case MODBUS_FUNCTION_READ_DISCRETE_INPUTS:
  case MODBUS_FUNCTION_READ_HOLDING_REGISTERS:
  case MODBUS_FUNCTION_READ_INPUT_REGISTERS:
  {
…
  }
  break;

  /* функции записи одного регистра */
  case MODBUS_FUNCTION_WRITE_SINGLE_COILS:
  case MODBUS_FUNCTION_WRITE_SINGLE_REGISTER:
  {
…
  }
  break;

  /* функции записи группы регистров */
  case MODBUS_FUNCTION_WRITE_MULTIPLE_COILS:
  case MODBUS_FUNCTION_WRITE_MULTIPLE_REGISTERS:
  {
…
  }
  break;

  /* все остальные функции не поддерживаются */
  default:
    SendException(MODBUS_Exception_IllegalFunction);
    return MODBUS_Status_FunctionCallbacNotDefined;
  }

  return MODBUS_Status_UnknownError;
}
```

Заключение

Предложенная модель реализации протокола MODBUS RTU на микроконтроллере семейства STM32 позволяет максимально использовать аппаратные возможности UART микроконтроллера и довольно экономична в использовании ресурсов. Ответ ведущему устройству формируется непосредственно в буфере запроса, а благодаря использованию DMA удается избежать лишних копирований данных и использования промежуточных буферов. Предложенное решение использовалось в реальных проектах, в том числе с использованием контроллера STM32F030K6Tx с объемом ОЗУ всего 4 кБ и объёмом памяти программ 32 кБ.

Список литературы

1. Библиотека MODBUS [Электронный ресурс] : сайт разработчика – Режим доступа: https://libmodbus.org/

2. Библиотека MODBUS [Электронный ресурс] : облачное хранилище – Режим доступа: https://github.com/alejoseb/Modbus-STM32-HAL-FreeRTOS

3. Библиотека MODBUS [Электронный ресурс] : облачное хранилище – Режим доступа: https://sourceforge.net/projects/freemodbus.berlios/

4. Официальный сайт организации Modbus Organization. – Режим доступа: https://modbus.org/

5. Спецификация протокола MODBUS [Электронный ресурс] : – Режим доступа: https://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf

6. Спецификация MODBUS RTU [Электронный ресурс] : – Режим доступа: https://modbus.org/docs/Modbus_over_serial_line_V1_02.pdf