Программная реализация протокола 1-wire (iButton, MicroLan) на микроконтроллерах AVR

Автор: Погребняк Дмитрий. Самара, 2013.

1-wire - это низкоскоростная шина связи, разработанная корпорацией Dallas Semiconductor (которая позже была приобретена компанией Maxim Integrated). Особенностью этой шины является то, что и ля питания подключенных устройств, и для обмена данными используется одна и та же пара проводов.
Многие микроконтроллеры не реализуют аппаратную поддержку этой шины. Выходом является установка отдельного чипа –драйвера шины. Но, поскольку протокол довольно простой, его легко можно реализовать программно, без использования внешних драйверов, путём управления портами ввода-вывода.

В этой статье речь пойдёт о программной реализации применительно к микроконтроллерам семейства AVR (ATtiny, ATmega, ATxmega) фирмы Atmel.

Принцип работы шины 1-wire

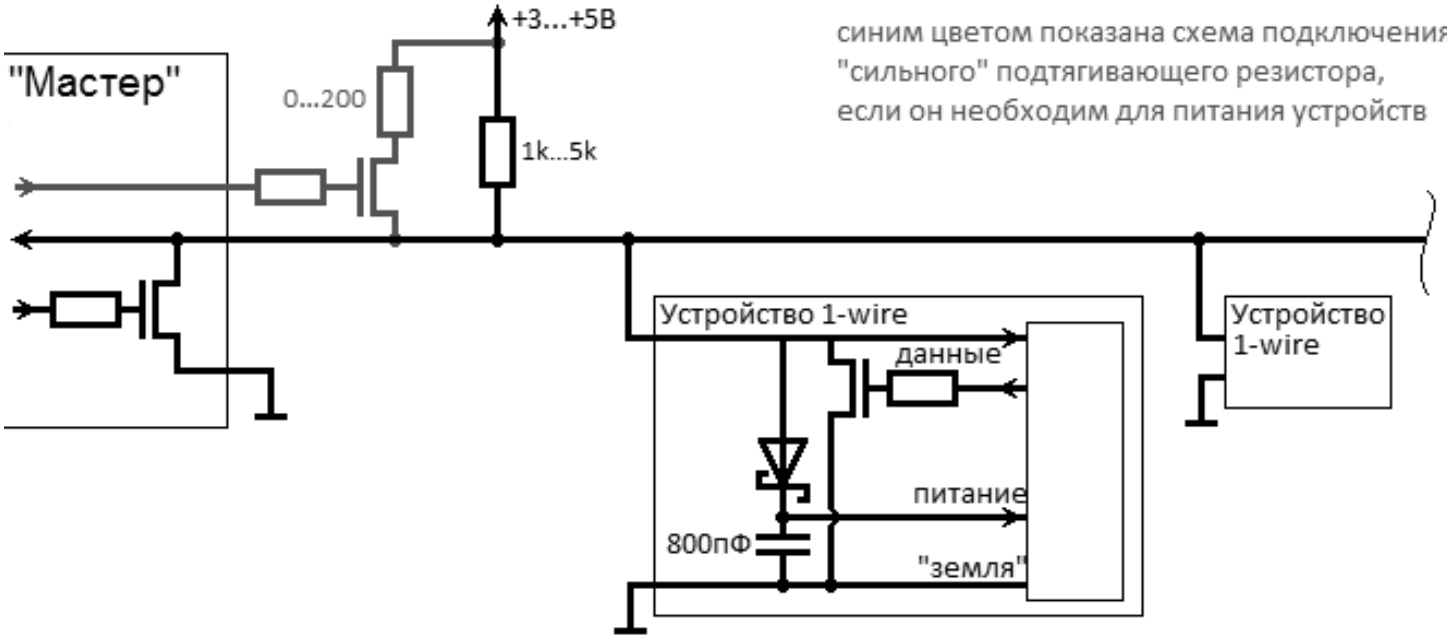


Схема подключения сети устройств 1-wire

Питание подводится по тому же проводу, что и данные (т.н. паразитное питание). Провод через подтягивающий резистор (5 кОм, но может быть и ниже, вплоть до 1 кОм) подключен к источнику напряжения 5 Вольт. Устройства реализуют схему с открытым стоком, замыкая линию на «землю». Длительность низкого уровня и определяет передаваемые данные. Для обеспечения питания схем во время передачи данных, устройства содержат внутренний конденсатор 800пФ, который заряжается, пока на шине нет активности.

На шине может быть параллельно подключено несколько ведомых устройств, но ведущее устройство («мастер») должно быть только одно. Такая сеть 1-wire устройств имеет название MicroLan. Обмен инициируется мастером (кроме некоторых специальных случаев, когда используются прерывания). Схема с открытым коллектором реализует «монтажное И», что позволяет в автоматическом режиме перечислить подключенные к шине устройства.

Подтягивающий резистор должен успевать поднять напряжение на шине до высокого логического уровня, что накладывает ограничение на общую электрическую ёмкость сети, зависящую от её длины и количества подключенных устройств. Вместе с тем, спецификация указывает минимальный ток, равный 4мА, при котором устройства должны принять при выставлении низкого уровня на шине. Таким образом, подтягивающий резистор должен быть номиналом не менее 1кОма. При таких параметрах на одну шину может быть подключено до 30 устройств, при длине шины более 200 метров.

Некоторые устройства могут требовать для совершения определённых операций более высокого значения тока на линии, чем то может обеспечить подтягивающий резистор. При наличии таких устройств, в схему должен быть введён подключаемый «сильный» подтягивающий резистор, обеспечивающий питание таким устройствам. Во время подключения такого подключения, активность на линии не допускается.

В каждом устройстве 1-wire жёстко зашиты уникальная восьмибайтная последовательность, однозначно идентифицирующая конкретный экземпляр устройства. Эта последовательность включает в себя:
- 1 байт «код семейства» определяющий тип устройства (коды с 0 по 127 зарезервированы для устройств Maxim Integrated/Dallas Semiconductor, коды 128 и выше вкупе с частью серийного номера – для сторонних производителей);
- 6 байт – серийный номер
- и 1 байт – контрольная сумма, рассчитанная на основе кода семейства и серийного номера.

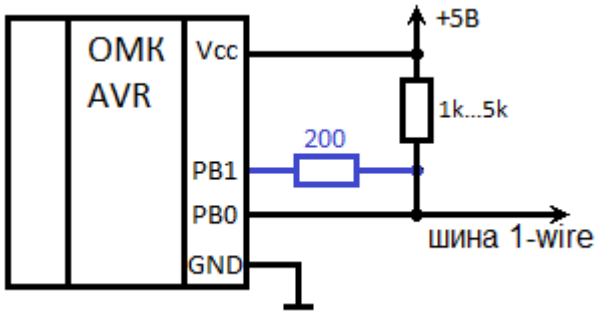
Применение 1-wire

Наибольшее применение устройства 1-wire нашли в виде круглых контактных площадок-«таблеток», называемых iButton, многим они знакомы в виде ключей от домофонов (впрочем, некоторые производители домофонов, например «Цифрал», используют иные, отличные от 1-wire, схемы, но выполненные в таком же корпусе). В наиболее простом случае микросхема, спрятанная внутри такой

«таблетки», содержит лишь уникальный код и ничего более. Хотя этот код уникальный, такой ключ несложно подделать.
В некоторых более сложных системах безопасности используются чипы со встроенной энергонезависимой памятью, с возможностью шифрования.

Другой пример использования – это различные датчики, например, датчики температуры из серии DS18B20, DS18S20, DS18S22 и им подобные. Использование всего двух проводов делает достаточно удобным подключение серии датчиков, вместе с тем, в отличие от аналоговых датчиков, показания не зависят от длины провода.

В качестве примера ниже будут рассмотрены как раз такие термодатчики.



Подключение к микроконтроллеру AVR

Подключение к микроконтроллеру AVR

В некоторых источниках рассматривается вариант подключения через последовательный порт, путём замыкания Rx и Tx. В таком варианте при использовании UART работающего на скорости 115200 бод, формируются импульсы нужной формы. Однако такое подключение не лишено ряда недостатков:
- задействованы два вывода микроконтроллера, когда можно вполне обойтись одним;
- порт UART оказывается занятым, а он часто присутствует на микроконтроллере в единственном числе и требуется для иных нужд;
- накладываются схематические ограничения, т.к. шина 1-wire должна быть подключена на строго определённые ножки микроконтроллера.

Вариант, предложенный в этой статье, предполагает подключение шины 1-wire на произвольную ножку микроконтроллера, которой можно управлять как портом ввода-вывода общего назначения (GPIO). Поскольку протокол связи 1-wire оставляет предостаточный

допуск на время проведения операций, то такое прямое управление портом не составит особой трудности.

Микроконтроллеры AVR способны выдавать ток до 40ма на свои порты. Для устройств, требующих подключения «сильного» подтягивающего резистора, можно использовать один из свободных портов, подключив его к шине через защитный резистор 200 Ом. Этот резистор требуется для того, чтобы ограничить ток в цепи, если в цепи произойдёт короткое замыкание, или «сильный» порт будет включен, когда одно из устройств пытается выдавать на линию сигнал. С риском для порта микроконтроллера, этот резистор можно исключить. В таком случае схему можно упростить, т.к. «сильное» подключение может обеспечиваться тем же портом, что используется для передачи данных (на схеме - PB0).

На уровне программы передача данных обеспечивается переключением значения соответствующего бита регистра, выбирающего направление порта (DDRx), пока соответствующий бит регистра, выбирающего значение на порту (PORTx) остаётся равным нулю.

Для порта «сильного» подтягивающего резистора требуется обеспечить переключение значений бит обоих регистров, чтобы в активном состоянии подключался сильный драйвер (соответствующие биты в регистрах DDRx и PORTx установлены в единицу) чтобы в неактивном состоянии он не влиял на шину (соответствующие биты в регистрах DDRx и PORTx сброшены в ноль). При переключении между этими двумя состояниями следует исключить возможность просаживания шины на «землю» (когда соответствующий бит в регистре DDRx равен единице, а в PORTx – нулю). Поэтому для включения «сильного» резистора требуется сначала установить значение (PORTx), что включит внутренний подтягивающий резистор и не изменит состояние шины, лишь затем – направление порта (DDRx). Отключение производить в обратной последовательности.

Пример кода

Код написан на языке C для компилятора avr-gcc с использованием библиотек из AVR-toolchain.
Примеры кода ниже будут использовать объявленные здесь макросы и функции, и вызванные подключения.

```
#define F_CPU 8000000UL // Объявление частоты микроконтроллера для макросов _delay_us

#include <avr/io.h>
#include <avr/pgmspace.h>
#include <util/delay.h>

#define ONEWIRE_PORT PORTB
#define ONEWIRE_DDR DDRB
#define ONEWIRE_PIN PINB
#define ONEWIRE_PIN_NUM PB0

// Устанавливает низкий уровень на шине 1-wire
inline void onewire_low() {
  ONEWIRE_DDR |= _BV(ONEWIRE_PIN_NUM);
}

// "Отпускает" шину 1-wire
inline void onewire_high() {
  ONEWIRE_DDR &= ~_BV(ONEWIRE_PIN_NUM);
}

// Читает значение уровня на шине 1-wire
inline uint8_t onewire_level() {
  return ONEWIRE_PIN & _BV(ONEWIRE_PIN_NUM);
}


// Определения и функции ниже нужны только если требуется "сильный" подтягивающий резистор
#define ONEWIRE_STRONG_DDR DDRB
#define ONEWIRE_STRONG_PORT PORTB
#define ONEWIRE_STRONG_PIN_NUM PB1
```

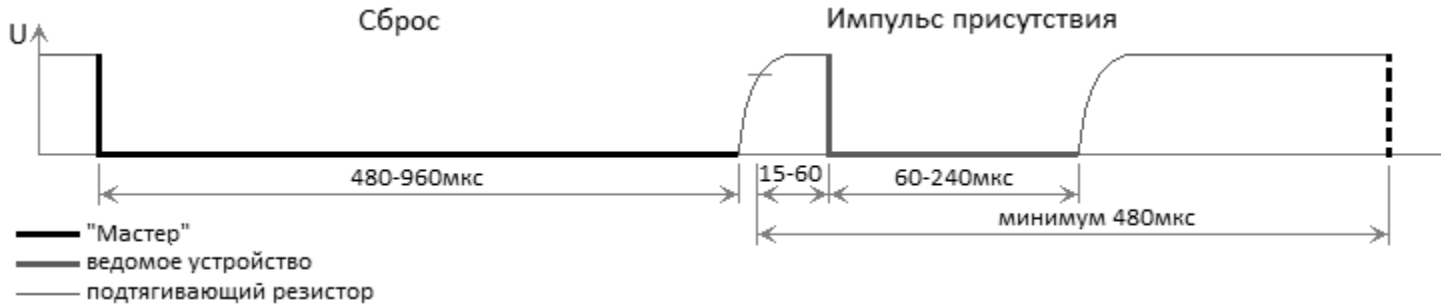
```
// включение "сильной" подтяжки
void onewire_strong_enable() {
    //  Для исключения низкого уровня на шине,
сначала изменяется регистр значения
                    ONEWIRE_STRONG_PORT          |=
_BV(ONEWIRE_STRONG_PIN_NUM);
  // затем - регистр направления
                     ONEWIRE_STRONG_DDR          |=
_BV(ONEWIRE_STRONG_PIN_NUM);
}

// отключение "сильной" подтяжки
void onewire_strong_disable() {
    //  Для исключения низкого уровня на шине,
сначала изменяется регистр направления
                     ONEWIRE_STRONG_DDR          &=
~_BV(ONEWIRE_STRONG_PIN_NUM);
  // затем - регистр значения
                    ONEWIRE_STRONG_PORT          &=
~_BV(ONEWIRE_STRONG_PIN_NUM);
}
```

Протокол связи

Сеанс связи инициируется, и передаваемые данные синхронизируется мастером. Определяющим является длительность интервала, в течение которого на шине установлен низкий уровень сигнала. Спецификация допускает достаточно широкий разброс временных параметров протокола, что облегчает реализацию протокола.

Сброс и импульс присутствия



Последовательность инициализации 1-wire: сброс и импульс присутствия

Любой обмен начинается с инициализации, во время которой мастер посылает импульс «сброс» (reset),. Этот импульс представляет собой низкий уровень на линии в течение минимум 480 микросекунд (т.к. для ведомых устройств минимальная длительность импульса сброса, на которой они реагируют, находится в диапазоне 120-480мкс). Если на линии присутствуют устройства, использующие прерывание, то импульс сброса не должен превышать 960 микросекунд (иначе на его фоне может затеряться сигнал прерывания).

За исключением случаев, когда линия подпитывается «сильной» подтяжкой, импульс сброса может быть послан в любой момент, даже если обмен данными с устройством не закончен.

Опознав импульс сброса, устройства выжидают от 15 до 60 микросекунд после его завершения (т.е. появления высокого уровня) и выдают импульс длительностью от 60 до 240 микросекунд, подтверждающий их присутствие на линии. Такой импульс называется «импульс присутствия» (presence pulse). Поскольку схема с открытым стоком реализует «монтажное И», одновременная передача импульса присутствия многими устройствами сливается в один импульс длиной до 285 микросекунд. Обнаруживая такой импульс, мастер может определить подключены ли устройства к шине.

Следующий импульс сброса должен посылаться не ранее чем через 480 микросекунд после завершения предыдущего.

```
// Выдаёт импульс сброса (reset), ожидает ответный импульс присутствия.
```

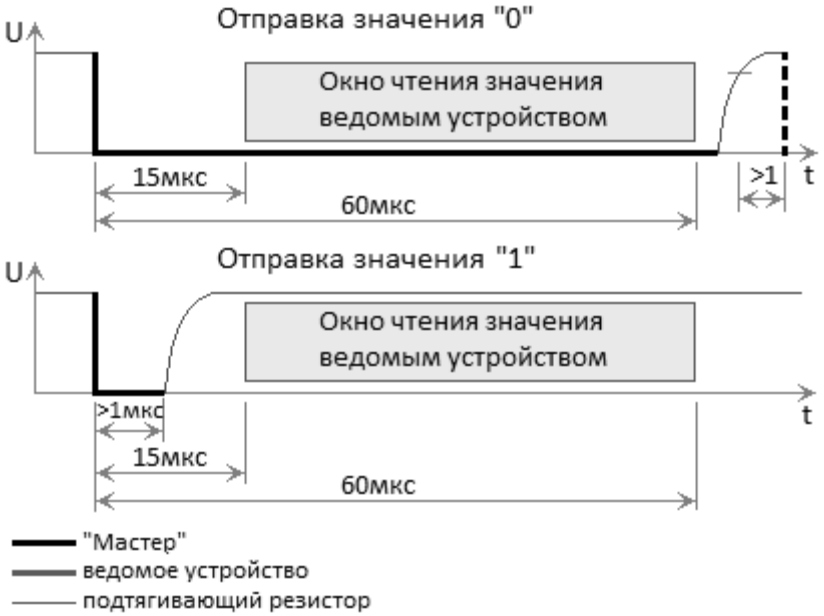
```
// Если импульс присутствия получен, дожидается его завершения и возвращает 1, иначе возвращает 0
uint8_t onewire_reset()
{
  onewire_low();
  _delay_us(640); // Пауза 480..960 мкс
  onewire_high();
       _delay_us(2);    //    Время    необходимое подтягивающему резистору, чтобы вернуть высокий уровень на шине
   // Ждём не менее 60 мс до появления импульса присутствия;
  for (uint8_t c = 80; c; c--) {
    if (!onewire_level()) {
       // Если обнаружен импульс присутствия, ждём его окончания
       while (!onewire_level()) { } // Ждём конца сигнала присутствия
      return 1;
    }
    _delay_us(1);
  }
  return 0;
}
```

Передача и приём данных

Данные передаются побитно, и для приёма и для передачи инициатором является мастер, выставляя низкий уровень на линии длительностью минимум 1 микросекунду. Для передачи одного бита выделяется интервал времени, называемый «тайм-слот» (time slot), составляющий от 60 до 120 микросекунд. Между соседними тайм-слотами на шине должен присутствовать высокий уровень в течение минимум 1 микросекунды.

Передача от мастера к ведомому устройству

Если ведомые устройства готовы к приёму данных, они ожидают импульса низкого уровня, после которого выжидают от 15 до 60 микросекунд и проводят замер уровня на шине. Высокий уровень соответствует единице, низкий – нулю.



Отправка значений от мастера к ведомому устройству 1-wire

Таким образом, для передачи единицы мастер должен вернуть высокий уровень на линии до истечения 15 микросекунд после появления низкого уровня. Следует учесть, что в зависимости от электрической ёмкости сети, подтягивающему резистору может потребоваться значительное время (до нескольких микросекунд), чтобы вытянуть напряжения на линии до высокого логического уровня. Поэтому мастер должен прекращать удерживать линию на низком уровне загодя.

Для передачи нуля мастер должен удерживать линию на низком уровне как минимум 60 микросекунд, но не дольше 120 микросекунд (т.к. иначе такой сигнал может быть воспринят некоторыми устройствами как импульс сброса). После возвращения высокого уровня на линии, следует выждать минимум 1 микросекунду до следующей активности.

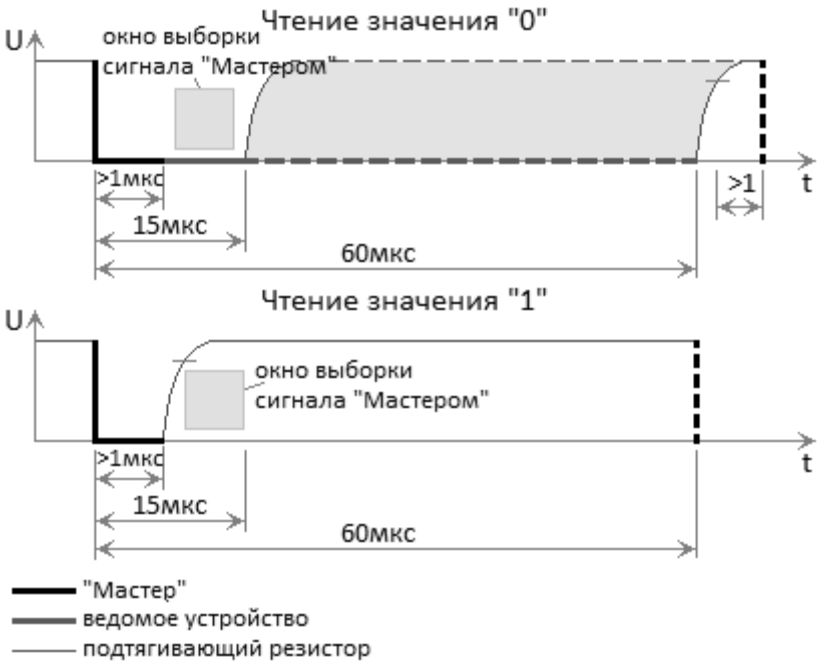
```
// Отправляет один бит
// bit отправляемое значение, 0 - ноль, любое другое значение - единица
void onewire_send_bit(uint8_t bit) {
  onewire_low();
  if (bit) {
     _delay_us(5); // Низкий импульс, от 1 до 15 мкс (с учётом времени восстановления уровня)
```

```
    onewire_high();
       _delay_us(90); // Ожидание до завершения таймслота (не менее 60 мкс)
  } else {
       _delay_us(90); // Низкий уровень на весь таймслот (не менее 60 мкс, не более 120 мкс)
    onewire_high();
    _delay_us(5); // Время восстановления высокого уровеня на шине + 1 мс (минимум)
  }
}

// Отправляет один байт, восемь подряд бит, младший бит вперёд
// b - отправляемое значение
void onewire_send(uint8_t b) {
  for (uint8_t p = 8; p; p--) {
    onewire_send_bit(b & 1);
    b >>= 1;
  }
}
```

Приём от ведомого устройства мастером

Когда подразумевается ответ ведомого устройства, мастер также инициирует передачу очередного бита кратковременным (не менее 1 микросекунды) выставлением низкого уровня на шине. Распознав срез, активное ведомое устройство для передачи нуля продолжает удерживать низкий уровень на линии в течение 15-60 мкс, а для передачи единицы не предпринимает никаких действий, позволяя подтягивающему резистору вернуть высокий уровень на линии.



Приём значений мастером от ведомого устройства 1-wire

Из вышесказанного следует, что замер уровня (сэмплирование) мастером должно быть произведено до истечения 15 микросекунд после установки низкого уровня, но при этом спустя достаточное время после освобождения шины, чтобы подтягивающий резистор гарантировано смог вернуть высокий уровень сигнала.

В случае если несколько устройств на шине передают данные одновременно, то, согласно правилу «монтажного И», передача неодинаковых данных будет распознана мастером как ноль.

```
// читает значение бита, передаваемое уйстройством.
// Возвращает 0 - если передан 0, отличное от нуля значение - если передана единица
uint8_t onewire_read_bit() {
  onewire_low();
   _delay_us(2); // Длительность низкого уровня, минимум 1 мкс
  onewire_high();
  _delay_us(8); // Пауза до момента сэмплирования, всего не более 15 мкс
  uint8_t r = onewire_level();
   _delay_us(80); // Ожидание до следующего тайм-слота, минимум 60 мкс с начала низкого уровня
  return r;
}

// Читает один байт, переданный устройством, младший бит вперёд, возвращает прочитанное значение
uint8_t onewire_read() {
  uint8_t r = 0;
  for (uint8_t p = 8; p; p--) {
    r >>= 1;
    if (onewire_read_bit())
      r |= 0x80;
  }
```

```
  return r;
}
```

Прерывания

Некоторые устройства (например, DS1994) могут сигнализировать о возникновении ситуаций требующих реакции путём посылки прерываний. Прерывание представляет собой импульс низкого уровня на шине длительностью от 960 до 3840 микросекунд.

Сразу по возникновению события импульс может передаваться после процедуры инициализации (сброса и импульса присутствия) но до начала любой другой активности на шине. Если событие возникло в момент импульса присутствия, то устройство сначала дожидается появления высокого уровня на шине (окончания импульса присутствия).

Если же событие возникло после начала обмена данными и до импульса сброса, то устройство будет дожидаться очередного импульса сброса, и после него будет продолжать удерживать линию в низком уровне 960 до 3840 микросекунд. Если событие возникло во время импульса сброса, то до 3840 микросекунд с момента события.

Сразу после длительного импульса низкого уровня, спустя 15-60 микросекунд посылается импульс присутствия, как если бы сигнал прерывания был сигналом сброса. Такой же импульс присутствия будет послан и другими устройствами на шине, которые распознают импульс прерывания как импульс сброса.

Сетевой протокол

Обмен данными с устройствами начинается с процедуры инициализации (импульса сброса и ответного импульса присутствия) , после которого идут команды адресации, определяющие устройство, или несколько устройств, с которыми будет происходить дальнейший обмен.

Передача данных осуществляется младшими битами вперёд.

Все устройства 1-wire обладают зашитым в ПЗУ уникальным 64-разрядным кодом, который позволяет однозначно определить устройство. Этот код и используется для адресации.

Младший байт кода (передаваемый первым) кодирует семейство. Коды с 1 по 127 зарезервированы Dallas Semiconductor/Maxim Integrated и фактически определяют тип устройства. Таблицу с кодами семейств и поддерживаемыми командами можно найти вот здесь: http://owfs.sourceforge.net/family.html

Следующие шесть байт – это уникальный серийный номер, в паре с кодом семейства, он позволяет однозначно отличить конкретный экземпляр оборудования.

И, наконец, старший, восьмой байт – это контрольная сумма, вычисляемая на основе первых сими байт.

Расчёт контрольной суммы

Контрольная сумма представляет собой циклический избыточный код (Cyclic redundancy checkCRC) с полиномом $X^8 + X^5 + X^4 + 1$, инициализируется значением 0 и использует не инвертированную форму.

Контрольную сумму желательно использовать всегда, где это возможно, для проверки передаваемых данных.

Обновление контрольной суммы байтом собственного значения вернёт ноль. На практике это означает, что расчёт контрольной суммы для всех восьми байт адреса устройства, включая байт контрольной суммы, должен вернуть ноль. Эту особенность удобно применять при проверке полученных данных.

Для побитного расчёта можно воспользоваться функцией:

```
// Обновляет значение контольной суммы crc применением всех бит байта b.
// Возвращает обновлённое значение контрольной суммы
```

```
uint8_t onewire_crc_update(uint8_t crc, uint8_t b)
{
//  return pgm_read_byte(&onewire_crc_table[crc ^
b]);
  for (uint8_t p = 8; p; p--) {
      crc = ((crc ^ b) & 1) ? (crc >> 1) ^
0b10001100 : (crc >> 1);
    b >>= 1;
  }
  return crc;
}
```

Поскольку формула довольно проста, побитный расчёт требует около 90 тактов процессорного времени на каждый байт данных. Если же требуется ускорить расчёт, то можно использовать табличную функцию. Она выполняется значительно быстрее, но требует хранения 256-байтной таблицы:

```
const uint8_t PROGMEM onewire_crc_table[] = {
  0x00, 0x5e, 0xbc, 0xe2, 0x61, 0x3f, 0xdd, 0x83, 0xc2, 0x9c, 0x7e, 0x20, 0xa3, 0xfd, 0x1f, 0x41,
  0x9d, 0xc3, 0x21, 0x7f, 0xfc, 0xa2, 0x40, 0x1e, 0x5f, 0x01, 0xe3, 0xbd, 0x3e, 0x60, 0x82, 0xdc,
  0x23, 0x7d, 0x9f, 0xc1, 0x42, 0x1c, 0xfe, 0xa0, 0xe1, 0xbf, 0x5d, 0x03, 0x80, 0xde, 0x3c, 0x62,
  0xbe, 0xe0, 0x02, 0x5c, 0xdf, 0x81, 0x63, 0x3d, 0x7c, 0x22, 0xc0, 0x9e, 0x1d, 0x43, 0xa1, 0xff,
  0x46, 0x18, 0xfa, 0xa4, 0x27, 0x79, 0x9b, 0xc5, 0x84, 0xda, 0x38, 0x66, 0xe5, 0xbb, 0x59, 0x07,
  0xdb, 0x85, 0x67, 0x39, 0xba, 0xe4, 0x06, 0x58, 0x19, 0x47, 0xa5, 0xfb, 0x78, 0x26, 0xc4, 0x9a,
  0x65, 0x3b, 0xd9, 0x87, 0x04, 0x5a, 0xb8, 0xe6, 0xa7, 0xf9, 0x1b, 0x45, 0xc6, 0x98, 0x7a, 0x24,
  0xf8, 0xa6, 0x44, 0x1a, 0x99, 0xc7, 0x25, 0x7b, 0x3a, 0x64, 0x86, 0xd8, 0x5b, 0x05, 0xe7, 0xb9,
  0x8c, 0xd2, 0x30, 0x6e, 0xed, 0xb3, 0x51, 0x0f, 0x4e, 0x10, 0xf2, 0xac, 0x2f, 0x71, 0x93, 0xcd,
  0x11, 0x4f, 0xad, 0xf3, 0x70, 0x2e, 0xcc, 0x92, 0xd3, 0x8d, 0x6f, 0x31, 0xb2, 0xec, 0x0e, 0x50,
  0xaf, 0xf1, 0x13, 0x4d, 0xce, 0x90, 0x72, 0x2c, 0x6d, 0x33, 0xd1, 0x8f, 0x0c, 0x52, 0xb0, 0xee,
  0x32, 0x6c, 0x8e, 0xd0, 0x53, 0x0d, 0xef, 0xb1, 0xf0, 0xae, 0x4c, 0x12, 0x91, 0xcf, 0x2d, 0x73,
  0xca, 0x94, 0x76, 0x28, 0xab, 0xf5, 0x17, 0x49, 0x08, 0x56, 0xb4, 0xea, 0x69, 0x37, 0xd5, 0x8b,
  0x57, 0x09, 0xeb, 0xb5, 0x36, 0x68, 0x8a, 0xd4, 0x95, 0xcb, 0x29, 0x77, 0xf4, 0xaa, 0x48, 0x16,
  0xe9, 0xb7, 0x55, 0x0b, 0x88, 0xd6, 0x34, 0x6a, 0x2b, 0x75, 0x97, 0xc9, 0x4a, 0x14, 0xf6, 0xa8,
  0x74, 0x2a, 0xc8, 0x96, 0x15, 0x4b, 0xa9, 0xf7, 0xb6, 0xe8, 0x0a, 0x54, 0xd7, 0x89, 0x6b, 0x35
};

// Обновляет значение контольной суммы crc
применением всех бит байта b.
// Возвращает обновлённое значение контрольной
суммы
inline uint8_t onewire_crc_update(uint8_t crc,
uint8_t b) {
    return pgm_read_byte(&onewire_crc_table[crc ^
b]);
}
```

В подавляющем большинстве случаев скорости работы функции, вычисляющей сумму побитно – вполне достаточно.

Команда 0xCC SKIP ROM (игнорирование адресации)

Если достоверно известно, что на шине присутствует лишь одно устройство (т.н. схема подключения single-drop), то мастер может пропустить процедуру адресации, выслав байт 0xCC. Получив эту команду, ведомое устройство сразу перейдёт в активный режим, ожидая управляющих команд. В случае если на шине оказалось несколько устройств, то все они будут ожидать команд и реагировать на них одновременно, возможно, конфликтуя друг с другом.

```
// Выполняет последовательность инициализации (reset + ожидает импульс присутствия)
// Если импульс присутствия получен, выполняет команду SKIP ROM
// Возвращает 1, если импульс присутствия получен, 0 - если нет
uint8_t onewire_skip() {
  if (!onewire_reset())
    return 0;
  onewire_send(0xCC);
  return 1;
}
```

Команда 0x33 READ ROM (Чтение ПЗУ)

Если на шине присутствует лишь одно устройство, то послав 0x33, и обеспечив 64 тайм-слота чтения, мастер может прочитать его уникальный код. Если же на шине присутствует несколько устройств, то они будут отвечать одновременно, код будет сформирован по принципу «монтажного И», и контрольная сумма не сойдётся. Таким образом можно определить что на шину подключено несколько устройств.

После получения этой команды, устройство передаёт содержимое своего 64-битного ПЗУ, хранящего адрес. Сначала идёт младший бит.

Младший байт ПЗУ кодирует семейство устройства, затем 6 байт – уникальный серийный номер и последний байт – контрольная сумма.

```
// Выполняет последовательность инициализации (reset + ожидает импульс присутствия)
// Если импульс присутствия получен, выполняет команду READ ROM, затем читает 8-байтовый код устройства
//     и сохраняет его в буфер по указателю buf, начиная с младшего байта
// Возвращает 1, если код прочитан, 0 - если импульс присутствия не получен
uint8_t onewire_read_rom(uint8_t * buf) {
  if (!onewire_reset())
    return 0;
  onewire_send(0x33);
  for (uint8_t p = 0; p < 8; p++) {
    *(buf++) = onewire_read();
  }
  return 1;
}
```

Команда 0x55 MATCH ROM (Совпадение ПЗУ)

Эта команда нужна для выбора конкретного устройства, чей код достоверно известен.

После передачи 0x55 мастер передаёт полный 8-байтный код (включая код семейства, серийный номер и контрольную сумму) адресуемого устройства, младшими разрядами вперёд. Устройство, у которого зашитый в ПЗУ код соответствует переданному, переходит в активный режим, ожидая дальнейших команд.

```
// Выполняет последовательность инициализации (reset + ожидает импульс присутствия)
// Если импульс присутствия получен, выполняет команду MATCH ROM, и пересылает 8-байтовый код
//  по указателю data (младший байт вперёд)
```

```
// Возвращает 1, если импульс присутствия получен,
0 - если нет
uint8_t onewire_match(uint8_t * data) {
  if (!onewire_reset())
    return 0;
  onewire_send(0x55);
  for (uint8_t p = 0; p < 8; p++) {
    onewire_send(*(data++));
  }
  return 1;
}
```

Команда 0xF0 SEARCH ROM (Поиск ПЗУ)
С помощью команды 0xF0 можно получить коды всех устройств, подключенных к шине.

Сразу за этой командой идут 64 группы (на каждый бит зашитого в ПЗУ кода) состоящие из двух тайм-слотов чтения и одного тайм-слота передачи.

Получив эту команду, все устройства передают младший бит своего кода (соответствующий младшему биту кода семейства) , затем они передают инвертированное значение этого бита, и затем принимают значение бита от мастера. Если принятое значение соответствует значению бита в адресе, то подобные действия совершаются для следующего бита адреса, и так для всех 64 бит. В противном случае устройство переходит в неактивный режим, не участвуя в обмене до следующего импульса сброса.

Высылав эту команду, мастер проводит два тайм-слота чтения, во время которых каждое из ведомых устройств на шине передаёт бит своего адреса и его инвертированное значение.
Если значения этого бита адреса у всех устройств совпадают, то мастер получит последовательность 0 1 или 1 0 (для нуля и единицы, соответственно).
Если значения этого бита у адресов устройств различаются, то, используя принцип «монтажного И», оба значения будут прочитаны

мастером как нули. Мастер должен выбрать одно из значений, в соответствии с процедурой перебора адресов.

Если оба бита приняты мастером как единицы, это может означать отсутствие устройств. Такая ситуация возможна, если устройства не распознали команду, или на линии имеются иные проблемы. В этом случае мастер может попробовать послать команду заново, или прекратить процедуру перечисления.

Выбрав интересующее значение бита адреса, мастер передаёт его. В дальнейшем в процедуре поиска продолжают участвовать только те устройства, у которых значение бита адреса равно переданному мастером.

Процедура повторяется для всех 64 бит, в конце остаётся только одно устройство, которое продолжает оставаться в активном режиме для получения следующих команд.

Алгоритм перебора всех подключенных устройств

Идея процедуры вычисления номеров всех устройств выглядит следующим образом:

В памяти сохраняется последний вычисленный адрес (изначально все нули) и последняя позиция (изначально – больше максимально возможной), на которой возникла неоднозначность, и для которой было выбрано значение ноль. На очередном шаге для всех бит адреса, где возникает неоднозначность:

- для идущих раньше этой позиции – выбирается значение из сохранённого (последнего найденного) адреса.
- для этой позиции – выбирается единица
- для позиций правее – выбирается ноль.

Соответственно, сохраняется максимальный номер неоднозначной позиции, где был выбран ноль, для использования на следующем шаге.

Поиск заканчивается, если на всех неоднозначных позициях переданы единицы.

```
// Переменные для хранения промежуточного результата поиска
uint8_t onewire_enum[8]; // найденный восьмибайтовый адрес
uint8_t onewire_enum_fork_bit; // последний нулевой бит, где была неоднозначность (нумеруя с единицы)

// Инициализирует процедуру поиска адресов устройств
void onewire_enum_init() {
  for (uint8_t p = 0; p < 8; p++) {
    onewire_enum[p] = 0;
  }
  onewire_enum_fork_bit = 65; // правее правого
}

// Перечисляет устройства на шине 1-wire и получает очередной адрес.
// Возвращает указатель на буфер, содержащий восьмибайтовое значение адреса, либо NULL, если поиск завешён
uint8_t * onewire_enum_next() {
    if (!onewire_enum_fork_bit) { // Если на предыдущем шаге уже не было разногласий
      return 0; // то просто выходим ничего не возвращая
  }
  if (!onewire_reset()) {
    return 0;
  }
  uint8_t bp = 8;
  uint8_t * pprev = &onewire_enum[0];
  uint8_t prev = *pprev;
  uint8_t next = 0;

  uint8_t p = 1;
  onewire_send(0xF0);
```

```
  uint8_t newfork = 0;
  for(;;) {
    uint8_t not0 = onewire_read_bit();
    uint8_t not1 = onewire_read_bit();
     if (!not0) { // Если присутствует в адресах
бит ноль
      if (!not1) { // Но также присустствует бит 1
(вилка)
          if (p < onewire_enum_fork_bit) { // Если
мы левее прошлого правого конфликтного бита,
           if (prev & 1) {
              next |= 0x80; // то копируем значение
бита из прошлого прохода
           } else {
             newfork = p; // если ноль, то запомним
конфликтное место
           }
        } else if (p == onewire_enum_fork_bit) {
            next |= 0x80; // если на этом месте в
прошлый раз был правый конфликт с нулём, выведем 1
        } else {
          newfork = p; // правее - передаём ноль и
запоминаем конфликтное место
        }
      } // в противном случае идём, выбирая ноль в
адресе
    } else {
      if (!not1) { // Присутствует единица
        next |= 0x80;
         } else { // Нет ни нулей ни единиц -
ошибочная ситуация
        return 0;
      }
    }
    onewire_send_bit(next & 0x80);
    bp--;
    if (!bp) {
      *pprev = next;
```

```
      if (p >= 64)
        break;
      next = 0;
      pprev++;
      prev = *pprev;
      bp = 8;
    } else {
      if (p >= 64)
        break;
      prev >>= 1;
      next >>= 1;
    }
    p++;
  }
  onewire_enum_fork_bit = newfork;
  return &onewire_enum[0];
}

// Выполняет инициализацию (сброс) и, если импульс присутствия получен,
// выполняет MATCH_ROM для последнего найденного методом onewire_enum_next адреса
// Возвращает 0, если импульс присутствия не был получен, 1 - в ином случае
uint8_t onewire_match_last() {
  return onewire_match(&onewire_enum[0]);
}
```

Опрос температурных датчиков DS18B20, DS18S20, DS1822

Одним из наиболее частых применений протокола 1-wire является взаимодействие с температурными датчиками DS18B20, DS18S20, DS18S22.

Как предписано стандартом 1-wire, все датчики содержат 8-байтовый код, состоящий из кода семейства, уникального 48-битного серийного номера и контрольной суммы.

Датчики выпускаются в разных корпусах - трёхвыводные to92, для поверхностного монтажа - soic, а также в разнообразных вариантах герметичного исполнения.

Все они обеспечивают измерение температуры в пределах от -55°C до 125°C с точностью до 1/16 градуса. Время замера зависит от разрядности и достигает 0,75с.
Датчики могут работать как от внешнего источника питания, так и в режиме "паразитного питания" от линии данных, используя всего 2 провода.
Во втором случае вход внешнего питания должен быть соединён с общим выводом, и, так как устройства потребляют до 1,5 мА, на время замера температуры, шина данных должна подтягиваться "сильным" резистором, обеспечивая напряжение питания датчиков не менее 3 Вольт.

Получение значений температуры с датчиков выполняется в два этапа:
1) Запрос на измерение температуры (команда 0x44). Для устройств с паразитным питанием в течение 10мкс после этого должен быть включен сильный подтягивающий резистор на всё время замера (до 750мс).
2) Запрос на вычитывание измеренных значений (команда 0xBE).

Любому запросу к устройству обязательно предшествует сигнал сброса и одна из команд выбора устройства, в соответствии с сетевым протоколом 1-wire.
Результаты измерений сохраняются в устройстве в буферной памяти размером 9 байт, называемой скратчпад (scratchpad). У всех трёх рассматриваемых моделей термодатчиков структура скратчпада схожа между собой:

Позиция	**Значение**
0	младший байт значения температуры
1	старший байт значения температуры
2,3	граничные контрольные значения, или пользовательские данные
4	регистр конфигурации (DS18B20, DS1822) /зарезервирован (принимает значение 0xFF для DS18S20)
5	зарезервирован (принимает значение 0xFF)
6	зарезервирован (DS18B20, DS1822)/остаток COUNT_REMAIN (DS18S20)
7	зарезервирован (принимает значение 0x10 для DS18B20,

	DS1822)/множитель = отсчётов на градус (всегда 0x10 = 16 для DS18S20)
8	контрольная сумма, рассчитывается для всего скратчпада по тем же правилам что и для адреса 1-wire

Измеренное значение температуры сохраняется в виде знакового 16-битного целого в первых двух байтах скратчпада, сначала младший байт.
Датчики DS18B20 и DS1822 имеют программируемое разрешение от 9 до 12 бит, и сохраняют результат всегда в 1/16 градусах Цельсия. Таким образом значения 0x50 0x05 соответствуют +85°C, 0xA2 0x00 – +10,125°C, 0x6F 0xFE – -25,0625°C. По умолчанию разрешение датчиков задано 12 бит. При выборе меньшего разрешения (в этой статье не будет рассматриваться) сокращается время замера температуры, при этом недостающее число младших разрядов результата будет содержать неопределённые данные.
В отличие от них, DS18S20 сохраняет результат с разрешением в половину градуса. То есть 0xAA 0x00 соответствует +85°C, 0x32 0x00 – +25°C, 0xCE 0xFF – -25°C. Однако, можно вычислить значение с точностью до 1/16 градуса, используя значение COUNT_REMAIN, сохраняемое в позиции 6 скратчпада по следующей формуле:
ТЕМПЕРАТУРА = ПРОЧИТАННАЯ ТЕМПЕРАТУРА – 0,25°C + COUNT_REMAIN / 16,
Где 16 – это количество отсчётов на градус, значение, возвращаемое в позиции 7 скратчпада и всегда равное 16 для DS18S20.

Пример измерения температуры

Пример ниже собирает показания с датчиков DS18S20, DS18B20 и DS1822, подключенных к шине 1-wire с «сильным» подтягивающим резистором, как описано выше, и пересылает результаты измерений по UART. Используется процедуры и функции для работы с сетью 1-wire, описанные выше.

```
// Отправляет один байт в UART
void uart_write(uint8_t data) {
```

```
   while (!(UCSRA & (1 << UDRE))); // Ожидание освобождения буфера отправки
  UDR = data; // записывает байт в буфер, отправка начинается автоматически.
}

// Отсылает по UART одну цифру, являющуюся результатом деления нацело dig на sub.
// Возвращает остаток этого деления
uint16_t uart_digit(uint16_t dig, uint16_t sub) {
  char c = '0';
  while (dig >= sub) {
    dig -= sub;
    c++;
  }
  uart_write(c);
  return dig;
}

// Отсылает в UART десятичное представление числа с фиксированной точкой,
// где дробная часть представлена младшими 4 разрядами
void uart_num(int16_t num) {
  uint16_t unum; // число без знака
  if (num < 0) { // отрицательное число. Отсылает знак
    unum = -num;
    uart_write('-');
  } else {
    unum = num;
  }
   uint16_t snum = unum >> 4; // отбрасывает дробную часть
  if (snum >= 10) {
    if (snum >= 100) {
      if (snum >= 1000) {
```

```
            snum = uart_digit(snum, 1000); // 4й
разряд
      }
      snum = uart_digit(snum, 100); // 3й разряд
    }
    snum = uart_digit(snum, 10); // 2й разряд
  }
  uart_digit(snum, 1); // 1й разряд
  uart_write('.'); // десятичный разделитель
  uart_digit((((uint8_t)(unum & 0x0F)) * 10) >> 4,
1); // дробная часть
}

// Отсылает в UART шестнадцатиричное двузначное
представление числа
void uart_hex(uint8_t hexdig) {
   uart_write((hexdig >> 4) + (((hexdig >> 4) >=
10) ? ('A' - 10) : '0'));
   uart_write((hexdig & 0x0F) + (((hexdig & 0x0F)
>= 10) ? ('A' - 10) : '0'));
}

// Отсылает в UART символы перевода строки
void uart_newline() {
  uart_write('\r');
  uart_write('\n');
}

int main(void)
{
  // Инициализация портов
  ONEWIRE_PORT &= ~_BV(ONEWIRE_PIN_NUM);
  onewire_high();
  onewire_strong_disable();

  // Инициализация UART
   UCSRA &= ~(1 << U2X); // одинарная скорость
передачи
```

```
   UCSRB = (1 << TXEN); // Включает передатчик USART
    UCSRC = 0b1000110; // Асинхронный режим, контроль чётности - отключен, 1 стоп бит, 8 бит данных
  UBRRH = 0;
  UBRRL = 51; // 9600 бод

  while(1)
  {
     if (onewire_skip()) { // Если у нас на шине кто-то присутствует,...
       onewire_send(0x44); // ...запускается замер температуры на всех термодатчиках
          onewire_strong_enable(); // Включается усиленная подтяжка
      _delay_ms(900); // Минимум на 750 мс.
         onewire_strong_disable(); // Обязательно выключить подтяжку для дальнейшего обмена

       onewire_enum_init(); // Начало перечисления устройств
      for(;;) {
          uint8_t * p = onewire_enum_next(); // Очередной адрес
        if (!p)
          break;
        // Вывод шестнадцатиричной записи адреса в UART и рассчёт CRC
        uint8_t d = *(p++);
        uint8_t crc = 0;
          uint8_t family_code = d; // Сохранение первого байта (код семейства)
        for (uint8_t i = 0; i < 8; i++) {
          uart_hex(d);
          uart_write(' ');
          crc = onewire_crc_update(crc, d);
          d = *(p++);
```

```
        }
        if (crc) {
           // в итоге должен получиться ноль. Если
не так, вывод сообщения об ошибке
          uart_write('C');
          uart_write('R');
          uart_write('C');
        } else {
                  if ((family_code == 0x28) ||
(family_code == 0x22) || (family_code == 0x10)) {
               // Если код семейства соответствует
одному из известных...
                // 0x10 - DS18S20, 0x28 - DS18B20,
0x22 - DS1822
                 // проведём запрос scratchpad'a,
считая по ходу crc
            onewire_send(0xBE);
            uint8_t scratchpad[8];
            crc = 0;
            for (uint8_t i = 0; i < 8; i++) {
              uint8_t b = onewire_read();
              scratchpad[i] = b;
              crc = onewire_crc_update(crc, b);
            }
            if (onewire_read() != crc) {
              // Если контрольная сумма скретчпада
не совпала.
              uart_write('~');
              uart_write('C');
              uart_write('R');
              uart_write('C');
              uart_hex(crc);
            } else {
               int16_t t = (scratchpad[1] << 8) |
scratchpad[0];
              if (family_code == 0x10) { // 0x10 -
DS18S20
```

```
                    // у DS18S20 значение температуры
хранит 1 разряд в дробной части.
                     // повысить точность показаний
можно считав байт 6 (COUNT_REMAIN) из scratchpad
                   t <<= 3;
                     if (scratchpad[7] == 0x10) { //
проверка на всякий случай
                     t &= 0xFFF0;
                     t += 12 - scratchpad[6];
                   }
                  } // для DS18B20 DS1822 значение по
умолчанию 4 бита в дробной части
                // Вывод температуры
                uart_num(t);
              }
            } else {
              // Неизвестное устройство
              uart_write('?');
            }
          }
          uart_newline();
        }
        uart_write('.');
      } else {
        uart_write('-');
      }
      uart_newline();
      _delay_ms(2000); // небольшая задержка
    }
}
```

Проект для AtmelStudio 6

Исходный код проекта доступен для скачивания здесь: zip-файл, 42 кБ.

Проект написан на языке C и реализован для микроконтроллеров ATmega8, работающих на частоте 8МГц, но может быть легко

адаптирован для других AVR микроконтроллеров с флеш-памятью 2 кБ и более. Схема подключения такая же, как на рисунке выше.
Архив также включает скомпилированный код для микроконтроллера ATmega8, в форматах hex и elf.

Помещённый здесь код, является свободным. То есть, допускается его свободное использование для любых целей, включая коммерческие, при условии указания ссылки на автора (Погребняк Дмитрий, http://aterlux.ru/).

```
/*
 * OneWire.c
 *
 * Проект, демонстрирующий работу с 1-wire сетью в режиме "мастера".
 * Реализует опрос датчиков температуры DS18B20, DS18S20, DS1822
 *
 * Author: Погребняк Дмитрий, г. Самара, 2013
 *
 * Помещённый здесь код является свободным. Т.е. допускается его свободное использование для любых целей, включая коммерческие, при условии указания ссылки на автора (Погребняк Дмитрий, http://aterlux.ru/).
 */


#include <avr/io.h>

#define F_CPU 8000000UL // Объявление частоты микроконтроллера для макросов _delay_us

#include <avr/io.h>
#include <avr/pgmspace.h>
#include <util/delay.h>

#define ONEWIRE_PORT PORTB
#define ONEWIRE_DDR DDRB
#define ONEWIRE_PIN PINB
#define ONEWIRE_PIN_NUM PB0

// Устанавливает низкий уровень на шине 1-wire
inline void onewire_low() {
  ONEWIRE_DDR |= _BV(ONEWIRE_PIN_NUM);
}

// "Отпускает" шину 1-wire
```

```
inline void onewire_high() {
  ONEWIRE_DDR &= ~_BV(ONEWIRE_PIN_NUM);
}

// Читает значение уровня на шине 1-wire
inline uint8_t onewire_level() {
  return ONEWIRE_PIN & _BV(ONEWIRE_PIN_NUM);
}


// Определения и функции ниже нужны только если требуется "сильный" подтягивающий резистор
#define ONEWIRE_STRONG_DDR DDRB
#define ONEWIRE_STRONG_PORT PORTB
#define ONEWIRE_STRONG_PIN_NUM PB1

// включение "сильной" подтяжки
void onewire_strong_enable() {
    //  Для исключения низкого уровня на шине, сначала изменяется регистр значения
                    ONEWIRE_STRONG_PORT          |= _BV(ONEWIRE_STRONG_PIN_NUM);
  // затем - регистр направления
                     ONEWIRE_STRONG_DDR          |= _BV(ONEWIRE_STRONG_PIN_NUM);
}

// отключение "сильной" подтяжки
void onewire_strong_disable() {
    //  Для исключения низкого уровня на шине, сначала изменяется регистр направления
                     ONEWIRE_STRONG_DDR          &= ~_BV(ONEWIRE_STRONG_PIN_NUM);
  // затем - регистр значения
                    ONEWIRE_STRONG_PORT          &= ~_BV(ONEWIRE_STRONG_PIN_NUM);
}
```

```
// Выдаёт импульс сброса (reset), ожидает ответный импульс присутствия.
// Если импульс присутствия получен, дожидается его завершения и возвращает 1, иначе возвращает 0
uint8_t onewire_reset()
{
  onewire_low();
  _delay_us(640); // Пауза 480..960 мкс
  onewire_high();
        _delay_us(2);    //    Время    необходимое подтягивающему резистору, чтобы вернуть высокий уровень на шине
   // Ждём не менее 60 мс до появления импульса присутствия;
  for (uint8_t c = 80; c; c--) {
    if (!onewire_level()) {
       // Если обнаружен импульс присутствия, ждём его окончания
       while (!onewire_level()) { } // Ждём конца сигнала присутствия
      return 1;
    }
    _delay_us(1);
  }
  return 0;
}


// Отправляет один бит
//  bit  отправляемое  значение,  0  -  ноль,  любое другое значение - единица
void onewire_send_bit(uint8_t bit) {
  onewire_low();
  if (bit) {
     _delay_us(5); // Низкий импульс, от 1 до 15 мкс (с учётом времени восстановления уровня)
    onewire_high();
```

```
      _delay_us(90); // Ожидание до завершения таймслота (не менее 60 мкс)
  } else {
      _delay_us(90); // Высокий уровень на весь таймслот (не менее 60 мкс, не более 120 мкс)
    onewire_high();
    _delay_us(5); // Время восстановления высокого уровеня на шине + 1 мс (минимум)
  }
}

// Отправляет один байт, восемь подряд бит, младший бит вперёд
// b - отправляемое значение
void onewire_send(uint8_t b) {
  for (uint8_t p = 8; p; p--) {
    onewire_send_bit(b & 1);
    b >>= 1;
  }
}


// читает значение бита, передаваемое уйстройством.
// Возвращает 0 - если передан 0, отличное от нуля значение - если передана единица
uint8_t onewire_read_bit() {
  onewire_low();
   _delay_us(2); // Длительность низкого уровня, минимум 1 мкс
  onewire_high();
  _delay_us(8); // Пауза до момента сэмплирования, всего не более 15 мкс
  uint8_t r = onewire_level();
   _delay_us(80); // Ожидание до следующего тайм-слота, минимум 60 мкс с начала низкого уровня
  return r;
}
```

```
// Читает один байт, переданный устройством, младший бит вперёд, возвращает прочитанное значение
uint8_t onewire_read() {
  uint8_t r = 0;
  for (uint8_t p = 8; p; p--) {
    r >>= 1;
    if (onewire_read_bit())
      r |= 0x80;
  }
  return r;
}


// Обновляет значение контольной суммы crc применением всех бит байта b.
// Возвращает обновлённое значение контрольной суммы
uint8_t onewire_crc_update(uint8_t crc, uint8_t b)
{
//  return pgm_read_byte(&onewire_crc_table[crc ^ b]);
  for (uint8_t p = 8; p; p--) {
       crc = ((crc ^ b) & 1) ? (crc >> 1) ^ 0b10001100 : (crc >> 1);
    b >>= 1;
  }
  return crc;
}


// Выполняет последовательность инициализации (reset + ожидает импульс присутствия)
// Если импульс присутствия получен, выполняет команду SKIP ROM
// Возвращает 1, если импульс присутствия получен, 0 - если нет
```

```
uint8_t onewire_skip() {
  if (!onewire_reset())
    return 0;
  onewire_send(0xCC);
  return 1;
}


//  Выполняет последовательность инициализации (reset + ожидает импульс присутствия)
//  Если импульс присутствия получен, выполняет команду READ ROM, затем читает 8-байтовый код устройства
//     и сохраняет его в буфер по указателю buf, начиная с младшего байта
//  Возвращает 1, если код прочитан, 0 - если импульс присутствия не получен
uint8_t onewire_read_rom(uint8_t * buf) {
  if (!onewire_reset())
    return 0;
  onewire_send(0x33);
  for (uint8_t p = 0; p < 8; p++) {
    *(buf++) = onewire_read();
  }
  return 1;
}


//  Выполняет последовательность инициализации (reset + ожидает импульс присутствия)
//  Если импульс присутствия получен, выполняет команду MATCH ROM, и пересылает 8-байтовый код
//   по указателю data (младший байт вперёд)
// Возвращает 1, если импульс присутствия получен, 0 - если нет
uint8_t onewire_match(uint8_t * data) {
  if (!onewire_reset())
    return 0;
```

```
  onewire_send(0x55);
  for (uint8_t p = 0; p < 8; p++) {
    onewire_send(*(data++));
  }
  return 1;
}


// Переменные для хранения промежуточного результата поиска
uint8_t onewire_enum[8]; // найденный восьмибайтовый адрес
uint8_t onewire_enum_fork_bit; // последний нулевой бит, где была неоднозначность (нумеруя с единицы)

// Инициализирует процедуру поиска адресов устройств
void onewire_enum_init() {
  for (uint8_t p = 0; p < 8; p++) {
    onewire_enum[p] = 0;
  }
  onewire_enum_fork_bit = 65; // правее правого
}

// Перечисляет устройства на шине 1-wire и получает очередной адрес.
// Возвращает указатель на буфер, содержащий восьмибайтовое значение адреса, либо NULL, если поиск завешён
uint8_t * onewire_enum_next() {
    if (!onewire_enum_fork_bit) { // Если на предыдущем шаге уже не было разногласий
      return 0; // то просто выходим ничего не возвращая
  }
  if (!onewire_reset()) {
    return 0;
```

```
  }
  uint8_t bp = 8;
  uint8_t * pprev = &onewire_enum[0];
  uint8_t prev = *pprev;
  uint8_t next = 0;

  uint8_t p = 1;
  onewire_send(0xF0);
  uint8_t newfork = 0;
  for(;;) {
    uint8_t not0 = onewire_read_bit();
    uint8_t not1 = onewire_read_bit();
     if (!not0) { // Если присутствует в адресах бит ноль
      if (!not1) { // Но также присустствует бит 1 (вилка)
         if (p < onewire_enum_fork_bit) { // Если мы левее прошлого правого конфликтного бита,
          if (prev & 1) {
             next |= 0x80; // то копируем значение бита из прошлого прохода
          } else {
            newfork = p; // если ноль, то запомним конфликтное место
          }
        } else if (p == onewire_enum_fork_bit) {
            next |= 0x80; // если на этом месте в прошлый раз был правый конфликт с нулём, выведем 1
        } else {
          newfork = p; // правее - передаём ноль и запоминаем конфликтное место
        }
      } // в противном случае идём, выбирая ноль в адресе
    } else {
      if (!not1) { // Присутствует единица
        next |= 0x80;
```

```
        } else { // Нет ни нулей ни единиц - ошибочная ситуация
          return 0;
        }
      }
      onewire_send_bit(next & 0x80);
      bp--;
      if (!bp) {
        *pprev = next;
        if (p >= 64)
          break;
        next = 0;
        pprev++;
        prev = *pprev;
        bp = 8;
      } else {
        if (p >= 64)
          break;
        prev >>= 1;
        next >>= 1;
      }
      p++;
    }
    onewire_enum_fork_bit = newfork;
    return &onewire_enum[0];
}

// Выполняет инициализацию (сброс) и, если импульс присутствия получен,
// выполняет MATCH_ROM для последнего найденного методом onewire_enum_next адреса
// Возвращает 0, если импульс присутствия не был получен, 1 - в ином случае
uint8_t onewire_match_last() {
    return onewire_match(&onewire_enum[0]);
}
```

```
//////////////////// РАБОТА С UART ////////////////////

// Отправляет один байт в UART
void uart_write(uint8_t data) {
   while (!(UCSRA & (1 << UDRE))); // Ожидание освобождения буфера отправки
  UDR = data; // записывает байт в буфер, отправка начинается автоматически.
}

// Отсылает по UART одну цифру, являющуюся результатом деления нацело dig на sub.
// Возвращает остаток этого деления
uint16_t uart_digit(uint16_t dig, uint16_t sub) {
  char c = '0';
  while (dig >= sub) {
    dig -= sub;
    c++;
  }
  uart_write(c);
  return dig;
}

// Отсылает в UART десятичное представление числа с фиксированной точкой,
//  где дробная часть представлена младшими 4 разрядами
void uart_num(int16_t num) {
  uint16_t unum; // число без знака
   if (num < 0) { // отрицательное число. Отсылает знак
    unum = -num;
    uart_write('-');
  } else {
    unum = num;
  }
```

```
   uint16_t snum =   unum >> 4; // отбрасывает
дробную часть
  if (snum >= 10) {
    if (snum >= 100) {
      if (snum >= 1000) {
            snum = uart_digit(snum, 1000); // 4й
разряд
      }
      snum = uart_digit(snum, 100); // 3й разряд
    }
    snum = uart_digit(snum, 10); // 2й разряд
  }
  uart_digit(snum, 1); // 1й разряд
  uart_write('.'); // десятичный разделитель
  uart_digit((((uint8_t)(unum & 0x0F)) * 10) >> 4,
1); // дробная часть
}

// Отсылает в UART шестнадцатиричное двузначное
представление числа
void uart_hex(uint8_t hexdig) {
   uart_write((hexdig >> 4) + (((hexdig >> 4) >=
10) ? ('A' - 10) : '0'));
   uart_write((hexdig & 0x0F) + (((hexdig & 0x0F)
>= 10) ? ('A' - 10) : '0'));
}

// Отсылает в UART символы перевода строки
void uart_newline() {
  uart_write('\r');
  uart_write('\n');
}

int main(void)
{
  // Инициализация портов
  ONEWIRE_PORT &= ~_BV(ONEWIRE_PIN_NUM);
  onewire_high();
```

```
  onewire_strong_disable();

  // Инициализация UART
   UCSRA &= ~(1 << U2X); // одинарная скорость передачи
   UCSRB = (1 << TXEN); // Включает передатчик USART
    UCSRC = 0b1000110; // Асинхронный режим, контроль чётности - отключен, 1 стоп бит, 8 бит данных
  UBRRH = 0;
  UBRRL = 51; // 9600 бод

  while(1)
  {
     if (onewire_skip()) { // Если у нас на шине кто-то присутствует,...
       onewire_send(0x44); // ...запускается замер температуры на всех термодатчиках
          onewire_strong_enable(); // Включается усиленная подтяжка
       _delay_ms(900); // Минимум на 750 мс.
         onewire_strong_disable(); // Обязательно выключить подтяжку для дальнейшего обмена

       onewire_enum_init(); // Начало перечисления устройств
      for(;;) {
           uint8_t * p = onewire_enum_next(); // Очередной адрес
        if (!p)
          break;
        // Вывод шестнадцатиричной записи адреса в UART и рассчёт CRC
        uint8_t d = *(p++);
        uint8_t crc = 0;
          uint8_t family_code = d; // Сохранение первого байта (код семейства)
```

```
        for (uint8_t i = 0; i < 8; i++) {
          uart_hex(d);
          uart_write(' ');
          crc = onewire_crc_update(crc, d);
          d = *(p++);
        }
        if (crc) {
           // в итоге должен получиться ноль. Если не так, вывод сообщения об ошибке
          uart_write('C');
          uart_write('R');
          uart_write('C');
        } else {
                  if ((family_code == 0x28) || (family_code == 0x22) || (family_code == 0x10)) {
                // Если код семейства соответствует одному из известных...
                 // 0x10 - DS18S20, 0x28 - DS18B20, 0x22 - DS1822
                   // проведём запрос scratchpad'a, считая по ходу crc
            onewire_send(0xBE);
            uint8_t scratchpad[8];
            crc = 0;
            for (uint8_t i = 0; i < 8; i++) {
              uint8_t b = onewire_read();
              scratchpad[i] = b;
              crc = onewire_crc_update(crc, b);
            }
            if (onewire_read() != crc) {
              // Если контрольная сумма скретчпада не совпала.
              uart_write('~');
              uart_write('C');
              uart_write('R');
              uart_write('C');
              uart_hex(crc);
            } else {
```

```
                int16_t t = (scratchpad[1] << 8) |
scratchpad[0];
              if (family_code == 0x10) { // 0x10 -
DS18S20
                // у DS18S20 значение температуры
хранит 1 разряд в дробной части.
                  // повысить точность показаний
можно считав байт 6 (COUNT_REMAIN) из scratchpad
                t <<= 3;
                  if (scratchpad[7] == 0x10) { //
проверка на всякий случай
                    t &= 0xFFF0;
                    t += 12 - scratchpad[6];
                  }
                } // для DS18B20 DS1822 значение по
умолчанию 4 бита в дробной части
                // Вывод температуры
                uart_num(t);
              }
            } else {
              // Неизвестное устройство
              uart_write('?');
            }
          }
          uart_newline();
        }
        uart_write('.');
      } else {
        uart_write('-');
      }
      uart_newline();
      _delay_ms(2000); // небольшая задержка
    }
}
```