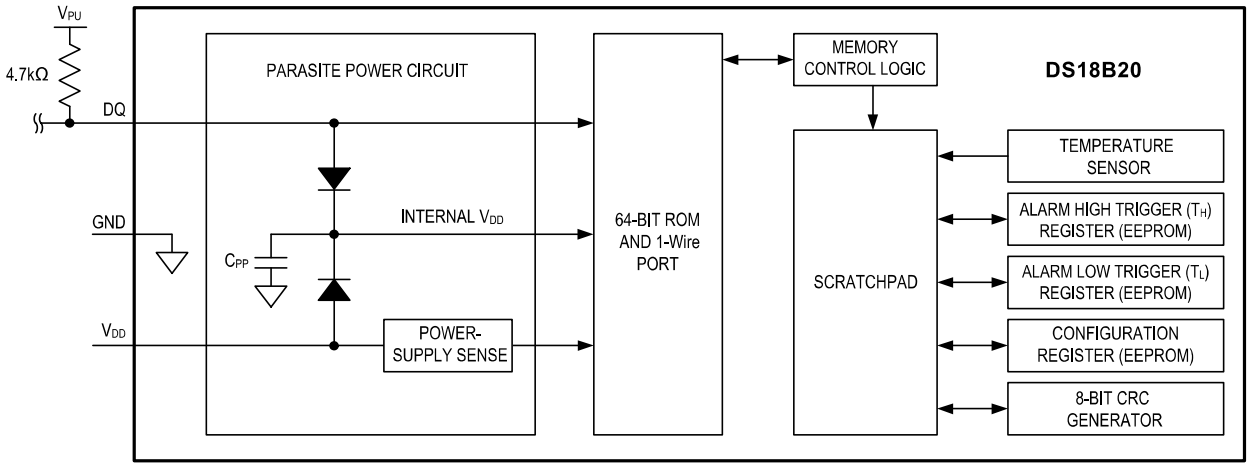
Датчик температуры DS18B20

Датчик температуры DS18B20 работает по протоколу 1-Wire (с английского «один провод»). Протокол был разработан компанией Dallas Semiconductor (в 2001 году выкуплена Maxim Integrated Products). Как понятно из названия, для передачи данных используется один провод (не считая земли), следовательно, требуется всего одна ножка GPIO (в общем случае). Более того, на один провод (который называют шиной DQ) вы можете посадить несколько устройств, и тогда обращение к конкретному устройству можно будет произвести по определенному идентификатору. В ваших часах всего один датчик на шине, поэтому мы рассмотрим работу всего с одним устройством.

Справедливости ради стоит заметить, что реализуют работу с датчиками в данном протоколе по-разному.

- «Тупой Ногодрыг» — ножка последовательно переключается из высокого состояния в низкое, тем самым формируя последовательности состояний (т. е. формируя команды для устройства). Требуется работа с GPIO и задержкой, скажем, реализованная на SysTick. Это чисто программная реализация.
- «Умный Ногодрыг» — в целях разгрузки основного цикла можно переложить работу на один из таймеров в МК (для формирования команд) и модуль прямого доступа к памяти (англ. DMA – direct memory access), который будет подсовывать нужные данные в нужный момент. Относительно нетривиальная задача.
- Нестандартное использование модуля UART — требуется две ножки, RX и TX, описан в документации на DS18B20.
- Использование специализированной микросхемы-драйвера, например, DS2480B от компании Maxim.

В режиме ожидания (англ. idle) на шине должен быть высокий уровень, что достигается подтяжкой линии к питанию через нагрузочный резистор (англ. push-up). Величина подтягивающего резистора указывается в документации на подчиненное устройство. Давайте скачаем **документацию** на датчик DS18B20 и найдем необходимую нам информацию.



Структура датчика DS18B20, изображение из документации

Кроме того, что сопротивление резистора подтяжки — 4,7 кОм, можно найти и другую важную для нас информацию, например: ножка главного устройства должна быть в режиме открытого стока. Кроме того, из описания датчика нам становится известно, что он способен работать при температурах от -55 °C до +125 °C, с погрешностью ±0.5 °C (в диапазоне от -10 °C до +85 °C). Точность измерений задается пользователем (от 9 до 12 бит). Чем больше точность — тем больше время для обработки.

Сейчас будет описано, как работает данный датчик температуры, однако критически важно, чтобы вы не просто прочитали этот текст, но и сами заглянули в документацию и отследили каждое утверждение, которое мы делаем. Не поленитесь, ведь наша с вами цель — научится пользоваться документацией, где в большинстве случаев есть всё, что нам требуется. Приготовим заголовочный файл `ds18b20.h`, он будет включать в себя следующие прототипы функций и перечисление:

```
#ifndef __DS18B20_H__
#define __DS18B20_H__

#include "stm32f10x.h"
#include "utils.h"

typedef enum {
    DS18B20_Resolution_9_bit    = 0x1F,
    DS18B20_Resolution_10_bit   = 0x3F,
    DS18B20_Resolution_11_bit   = 0x5F,
    DS18B20_Resolution_12_bit   = 0x7F
} DS18B20_RESOLUTION_t;

void ds18b20_init(const DS18B20_RESOLUTION_t resolution);
void ds18b20_set_resolution(const DS18B20_RESOLUTION_t resolution);
uint16_t ds18b20_get_temperature(void);

#endif /* __DS18B20_H__ */
```

Перечисление `DS18B20_RESOLUTION_t` содержит все возможные разрядности для температуры (стр. 8 Configuration register). Функция `ds18b20_init( ... )` позволит нам инициализировать порт и ножку, к которой подключен датчик (см. **схему**). Кроме того, через эту функцию мы установим разрядность. В документации также указано, какое время занимает преобразование (для 12 бит это 750 мс — от этого числа мы и будем отталкиваться для корректировки задержки). Заметим сразу, мы не будем реализовывать «универсальный драйвер», т. к. это займет больше времени, а смысл останется тот же. Полученный код вы с легкостью сможете портировать на другой МК, поменяв пару строк.

Для работы модуля нам потеряются некоторые функции, такие как запись бита или его чтения. Создадим их прототипы внутри файла исходного кода (они не должны быть доступны извне модуля), пояснения будут позже.

```
// ds18b20.c
static void write_bit(const uint8_t bit);
static void write_byte(const uint8_t byte);
static uint8_t read_bit(void);
static uint8_t get_devider(const DS18B20_RESOLUTION_t resolution);
static uint16_t read_temperature(void);
static void reset(void);
```

Линия данных DQ подключена к ножке 1 порта A. Его необходимо настроить как открытый сток.

```
void ds18b20_init(const DS18B20_RESOLUTION_t resolution) {
    RCC->APB2ENR |= RCC_APB2ENR_IOPAEN;

    // 50MHz output open-drain
    GPIOA->CRL |= GPIO_CRL_MODE1;
    GPIOA->CRL |= GPIO_CRL_CNF1_0;
    GPIOA->CRL &= ~GPIO_CRL_CNF1_1;

    ds18b20_set_resolution(resolution);
}
```

Вторая функция, `ds18b20_set_resolution( ... )`, позволит нам менять разрядность в любом участке кода. Внимательно изучив документацию мы получим следующее:

1. отправляем команду пропуска ожидания адреса (устройство на шине одно);
2. отправляем команду, что хотим записывать в регистры;
3. отправляем название регистров;
4. устанавливаем разрешение;
5. пересчитываем время конвертации.

```
void ds18b20_set_resolution(const DS18B20_RESOLUTION_t resolution) {
    reset();                        // send reset
    write_byte(SKIP_ROM);           // work only with one device
    write_byte(WRITE_SCRATCHPAD); // set resolution
    write_byte(TH_REGISTER);        //
    write_byte(TL_REGISTER);        //
    write_byte(resolution);         //
    convert_delay = DELAY_T_CONVERT / get_devider(resolution); // calc conversation delay
}
```

Она вызывает ряд функций, разобраться в нужности которых (а также в том, как их писать) нам еще предстоит ниже. Но перед этим обратим внимание на строчку:

```
convert_delay = DELAY_T_CONVERT / get_devider(resolution);
```

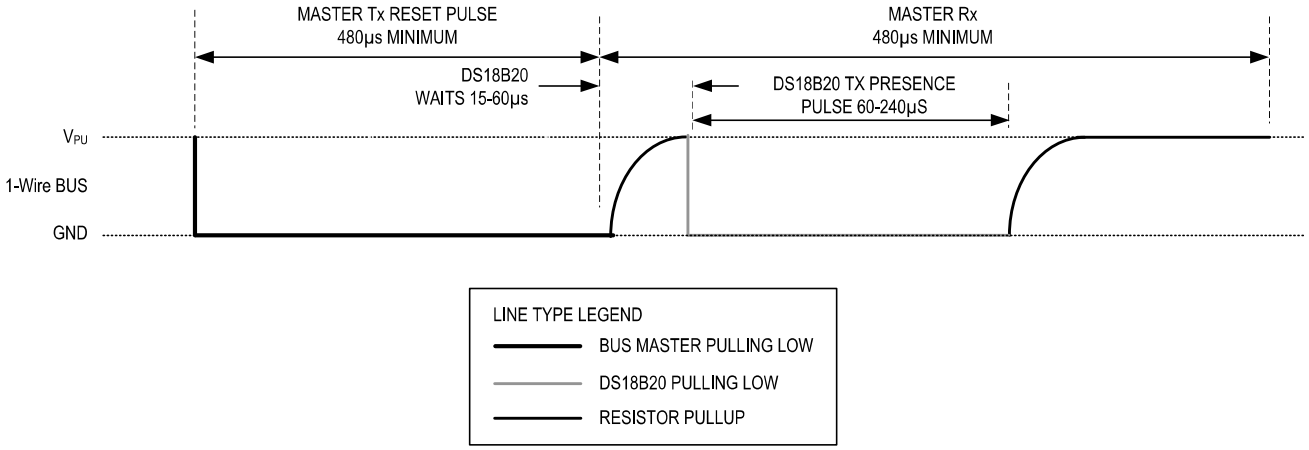
Так как мы можем задавать разную разрядность преобразования, нам нужно контролировать время задержки между началом преобразования и началом чтения данных из памяти датчика. Реализуем функцию получения делителя задержки.

```
static uint32_t convert_delay = DELAY_T_CONVERT;
// ...
uint8_t get_devider(const DS18B20_RESOLUTION_t resolution) {
    uint8_t devider;
    switch (resolution) {
    case DS18B20_Resolution_9_bit:
        devider = 8;
        break;
    case DS18B20_Resolution_10_bit:
        devider = 4;
        break;
    case DS18B20_Resolution_11_bit:
        devider = 2;
        break;
```

```
    case DS18B20_Resolution_12_bit:
    default:
        devider = 1;
        break;
    }

    return devider;
}
```

Теперь приступим к описанию работы самого протокола. Обмен всегда ведется по инициативе главного устройства (в нашем случае МК) и начинается с импульса сброса (англ. reset). Главное устройство подтягивает линию к логическому нулю на время не менее 480 мкс, а затем переключается в режим входа. При переключении линия через резистор подтяжки кратковременно подтягивается к высокому уровню, и подчиненное устройство, выждав 15-60 мкс, отправляет импульс приветствия (англ. presence), подтягивая линию к низкому уровню на длительность от 60 до 240 мкс.



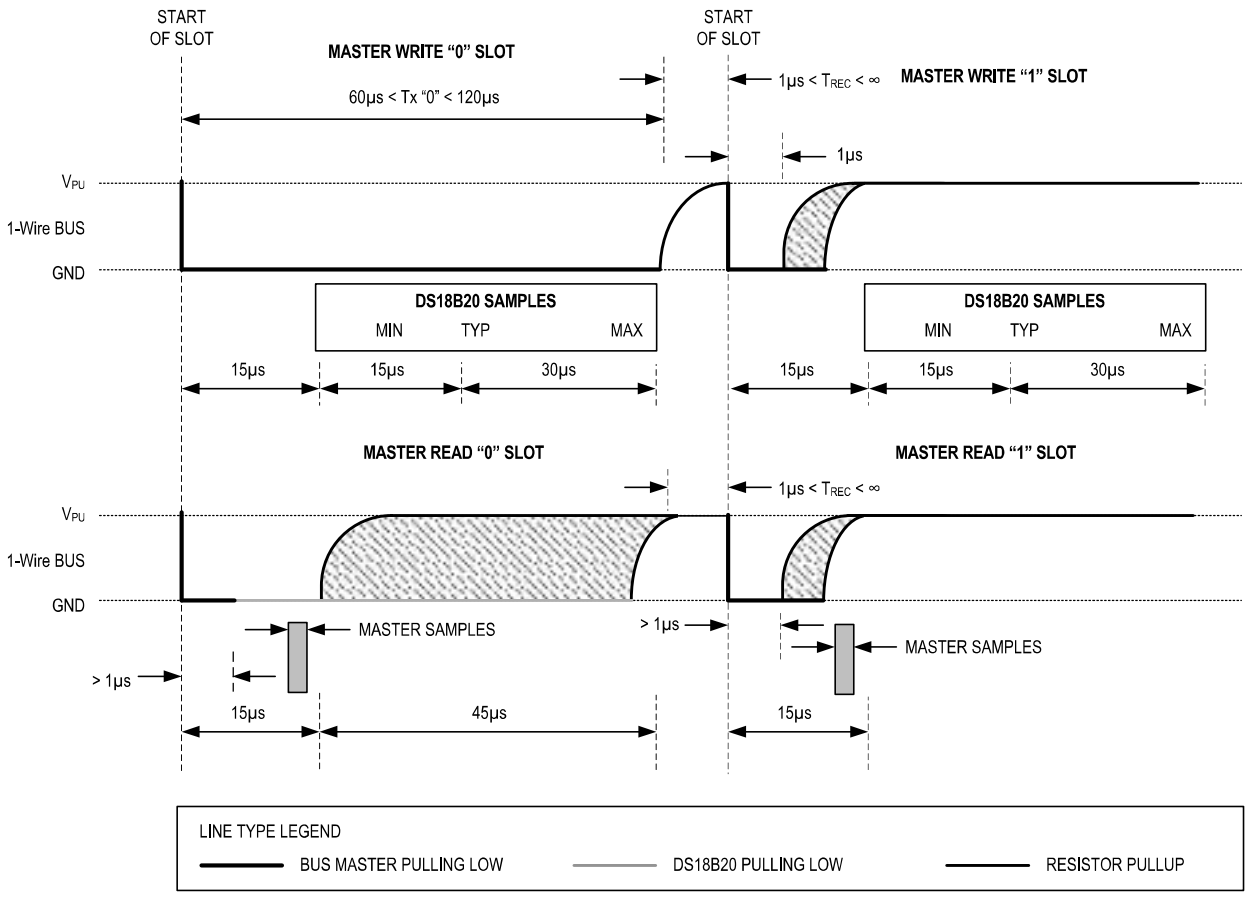
Импульс приветствия, изображение из документации

```
void reset(void) {
    GPIOA->ODR &= ~GPIO_ODR_ODR1;
    delay(DELAY_RESET);
    GPIOA->ODR |= GPIO_ODR_ODR1;
    delay(DELAY_RESET);
}
```

Таким образом, появление на шине DQ хотя бы одного импульса PRESENCE однозначно говорит о том, что на линии находится по крайней мере одно устройство.

Обмен данными по шине происходит последовательно, начиная с младшего бита. Передача/прием одного бита выполняется за определенное время, которое называется временным слотом (англ. time-slot). Сам временной слот находится в интервале от 60 до 120 мкс, а между ними обязательно должен быть предусмотрен разделяющий интервал не менее 1 мкс. Всего их может быть 4 вида: прием логической «1»; прием логического «0»; отправка логической «1»; отправка логического «0».

Любой временной слот начинается с перевода шины главным устройством в низкий логический уровень.



Временые слоты записи и чтения, изображение из документации

Чтобы передать «0», главное устройство должно установить на шине низкий уровень в течение всего временного слота. Для передачи «1» главное устройство устанавливает низкий уровень на шине на период от 1 до 15 мкс, а затем отпускает её (что ведет к установлению высокого уровня) и держит паузу до конца временного слота.

DS18B20 — подчиненное устройство, а значит, он может передавать данные только тогда, когда ему это позволят (соответствующей командой) и сформируют для него временной слот. Главное устройство может получить данные от подчиненного про прошествии 15 мкс после начала слота. Главное устройство просаживает линию к низкому уровню на 1 мкс и отпускает её. Если подчиненное устройство хочет передать «0», то оно оставляет линию в низком уровне, если хочет передать «1» — то держит 1 до конца слота.

Таким образом мы можем записать все необходимые задержки (с небольшим запасом):

```
#define DELAY_RESET           500
#define DELAY_WRITE_0         60
#define DELAY_WRITE_0_PAUSE   10
#define DELAY_WRITE_1         10
#define DELAY_WRITE_1_PAUSE   60
#define DELAY_READ_SLOT       10
#define DELAY_BUS_RELAX       10
#define DELAY_READ_PAUSE      50
#define DELAY_T_CONVERT       760000
#define DELAY_RELAXATION      5
```

Мы рассмотрели, что необходимо сделать в общем случае. Теперь нам предстоит получить от датчика значение температуры. Последовательность будет следующая:

- посылка RESET;
- ожидание PRESENCE (опустим получение отклика, т. к. выход Open-Drain ничего страшного если импульс придет на выход – мы даже сможем прочитать это значение в регистре `IDR`);
- работаем с единственным датчиком на шине, отправляя команду `SKIP_ROM`;
- запускаем преобразование температуры командой `CONVERT_T` и выжидаем время;
- читаем из ОЗУ датчика значение температуры командой `READ_SCRATCHPAD`.

Команды представлены ниже:

```
typedef enum {
    SKIP_ROM          = 0xCC,
    CONVERT_T         = 0x44,
    READ_SCRATCHPAD   = 0xBE,
    WRITE_SCRATCHPAD  = 0x4E,
    TH_REGISTER       = 0x4B,
    TL_REGISTER       = 0x46,
} COMMANDS_t;
```

Самая важная функция — `ds18b20_get_temperature()` — вернет нам целочисленное значение температуры (домноженное на 10 — это нужно для простой обработки данных).

```
uint16_t ds18b20_get_temperature() {
    reset();
    write_byte(SKIP_ROM);
    write_byte(CONVERT_T);
    delay(convert_delay);

    reset();
    write_byte(SKIP_ROM);
    write_byte(READ_SCRATCHPAD);

    return read_temperature();
}
```

Для записи разрядности последовательность примерно такая же (см. функцию `ds18b20_set_resolution( ... )`):

- посылка `RESET`;
- работаем с единственным датчиком на шине, отправляя команду `SKIP_ROM`;
- посылаем команду `WRITE_SCRACHPAD` для получения временных слотов для записи;
- последовательно записываем данные о нижнем и верхнем пороге срабатывания предупреждения (мы их не используем, но записать все равно придется) и разрядность.

Функции записи:

```
void write_bit(uint8_t bit) {
    GPIOA->ODR &= ~GPIO_ODR_ODR1;
    delay(bit ? DELAY_WRITE_1 : DELAY_WRITE_0);
    GPIOA->ODR |= GPIO_ODR_ODR1;
    delay(bit ? DELAY_WRITE_1_PAUSE : DELAY_WRITE_0_PAUSE);
}

void write_byte(uint8_t data) {
    for (uint8_t i = 0; i < 8; i++) {
        write_bit(data >> i & 1);
        delay(DELAY_RELAXATION);
    }
}
```

Функции чтения:

```
uint8_t read_bit(void) {
    uint8_t bit = 0;
    GPIOA->ODR &= ~GPIO_ODR_ODR1;
    delay(DELAY_READ_SLOT);
    GPIOA->ODR |= GPIO_ODR_ODR1;
    // ... switch to INPUT
    delay(DELAY_BUS_RELAX);
    bit = (GPIOA->IDR & GPIO_IDR_IDR1 ? 1 : 0);
    delay(DELAY_READ_PAUSE);
    // ... switch to OUTPUT
    return bit;
}

uint16_t read_temperature(void) {
    uint16_t data = 0;
    for (uint8_t i = 0; i < 16; i++)
        data += (uint16_t) read_bit() << i;
    return (uint16_t)(((float) data / 16.0f) * 10.0f);
}
```

Таким образом мы написали драйвер для работы с датчиком температуры DS18B20. Всё, что остается сделать — просто вызвать пару функций:

```
ds18b20_init(DS18B20_Resolution_11_bit);
// ...
result = ds18b20_get_temperature();
```

Пока мы не умеем выводить данные на дисплей, поэтому проверьте получившийся результат через режим отладки, прочитав значение переменной.

Код урока можно найти на github: **CMSIS**.

Назад | **Оглавление** | **Дальше**