Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)
Кафедра комплексной информационной безопасности электронно-вычислительных систем (КИБЭВС)

РАБОТА С ЦИФРОВЫМ ДАТЧИКОМ ПО ИНТЕРФЕЙСУ 1-WIRE

Отчет по практической работе №5
по дисциплине
«Аппаратные средства телекоммуникационных систем»

Студент гр.739-1
_____Цыриторов Ц.Б.
__.__.2023

Руководитель
Профессор кафедры КИБЭВС, д.т.н.
_____ Аврамчук В.С.
__.__.2023

Томск 2023

Введение

Цель работы – получить практические навыки работы с интерфейсом 1-Wire на основе CMSIS (Common Microcontroller Software Interface Standard). Реализовать функции записи команд и чтения данных по интерфейсу 1-Wire с датчика температуры DS18B20.

Задание: рассмотреть и описать представленные в курсе примеры программ, использующие интерфейс 1-Wire.

Теоретические сведения

DS18B20 – цифровой датчик температуры, поддерживающий 9–12-битное преобразование температуры. Структурная схема датчика показана на рисунке 1.

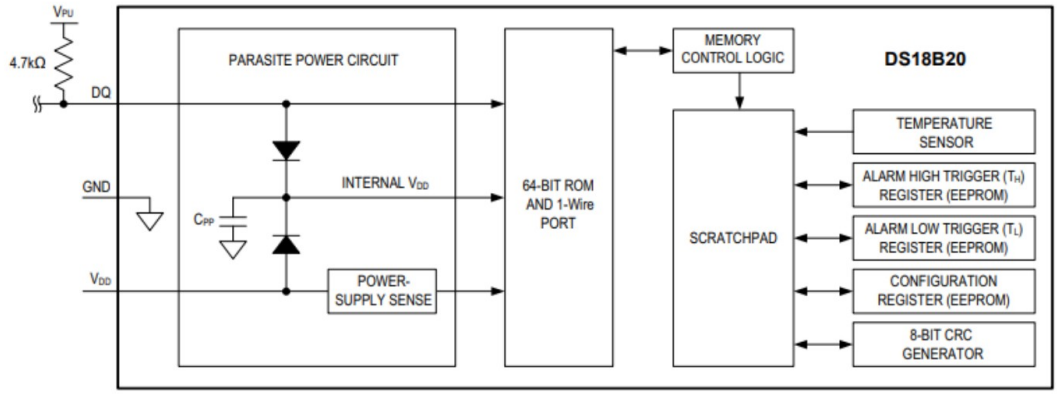


Рисунок 1 – Структурная схема

В состав датчика входит схема питания (Parasite Power Circuit), 64-битная память ROM для хранения кода (адреса) устройства, схема управления (Memory Control Logic), память для хранения данных (Scratchpad): температуры, настроек, граничных значений температуры, контрольной суммы CRC8; измеритель температуры (Temperature Sensor), модуль расчета CRC8 (8-BIT CRC Generator), регистры (EEPROM) для сохранения граничных значений температуры (Alarm High Trigger и Alarm Low Trigger), регистр для сохранения настроек (Configuration Register).

Для хранения температуры и настроек используется память размером 9 байт, структура которой представлена на рисунке 2.

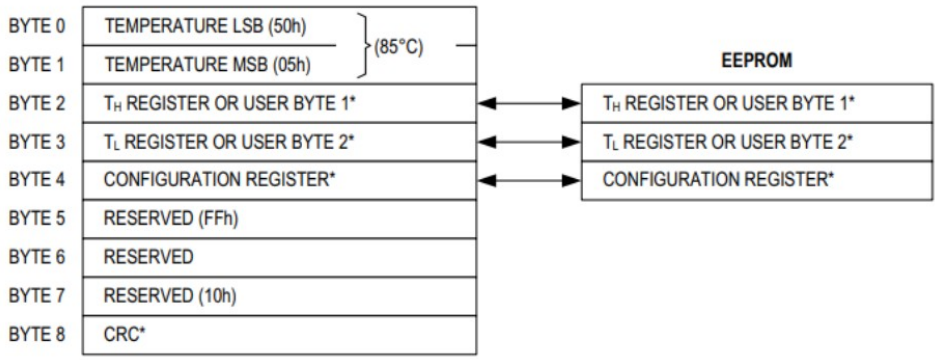


Рисунок 2 – Память для хранения температуры и настроек (Scratchpad)

Температура хранится в младших двух байтах памяти данных. Формат регистра температуры представлена на рисунке 3. Старшие 5 бит отводятся под знак, остальные 12 бит для кодирования температуры.

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
LS BYTE	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
	BIT 15	BIT 14	BIT 13	BIT 12	BIT 11	BIT 10	BIT 9	BIT 8
MS BYTE	S	S	S	S	S	2^6	2^5	2^4

S = SIGN

Рисунок 3 – Формат регистра температуры

За выход за старший бит, то есть, если температура не влезает в 7 бит (8-й отвечает за знак), то в 1 устанавливается флаг alarm flag.

На рисунке 4 представлен формат регистра настроек.

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
0	R1	R0	1	1	1	1	1

Рисунок 4 – Формат регистра настроек

Также, как ранее было сказано, рассчитывается код CRC8, для контроля целостности. Рассчитывается для байтов 0–7 памяти данных на основе образующего полинома $X^8+ X^5+ X^4+1$.

Для проверки целостности данных необходимо вычислить CRC8 для принятых 8 байт память данных и сравнить полученную CRC8 с принятой CRC8 вместе с сообщением.

Если устройств в сети несколько, то необходимо либо предварительно узнать код каждого устройства (например, включая устройства по одному и читая их коды в память), либо использовать команду поиска кодов (0xF0) и специальный алгоритм считывания кодов устройств, основанный на том, что доминантным является низкий уровень сигнала на шине. Таким образом, если какие-то устройства будут выставлять на шине низкий уровень, а какие- то — высокий, то «победят» те, кто выставлял низкий уровень, и, соответственно, на шине установится низкий уровень.

Идея работы

Идея работы алгоритма поиска устройств заключается в следующем: после того, как мастер отправляет в сеть команду 0xF0, все Slave-устройства начинают обмениваться с мастером информацией по следующему алгоритму:

1) Slave передаёт мастеру младший бит ROM;

2) Slave передаёт мастеру инверсную копию младшего бита ROM и ждёт ответ от мастера;

3) Если мастер присылает в ответ тот же самый бит, который Slave посылал ему в пункте 1, то Slave повторяет шаги 1,2 для следующего бита ROM, в противном случае Slave прекращает обмен данными с мастером.

Со стороны мастера, при считывании битов от slave-устройств, возможны следующие ситуации:

– Мастер считывает с шины 01. Это означает, что все устройства, участвующие в обмене данными, послали ему биты 01 (текущий бит ROM = 0, его инверсная копия = 1);

– Мастер считывает с шины 10. Это означает, что все устройства, участвующие в обмене данными, послали ему биты 10 (аналогично первому

– Мастер считывает с шины 00. Это означает, что какие-то устройства послали ему 01, а какие-то 10 (в обоих случаях победили те, кто посылал нули). Это означает, что перед мастером находится узел двоичного дерева, то есть на этой ветви существуют такие ROM-адреса, у которых следующий бит равен нулю и такие, у которых следующий бит равен единице;

– Есть ещё вариант, когда мастер считывает с шины 11. Казалось бы, такая ситуация является невероятной, однако она тоже возможна, например, если устройства, с которыми мастер решил продолжить обмен, отключились.

Работа с датчиком по интерфейсу 1-Wire осуществляется с помощью команд. Есть представление этих команд в виде однобайтового шестнадцатеричного кода. Например, Search ROM (код F0) – производит поиск ROM-кодов подключенных устройств; Read ROM (33) – чтение ROM-кода,

если только одно устройство присутствует в сети. Полный список команд находится в документации к датчику.

К плате STM32F103RB подключается так, как показано на рисунке 5.

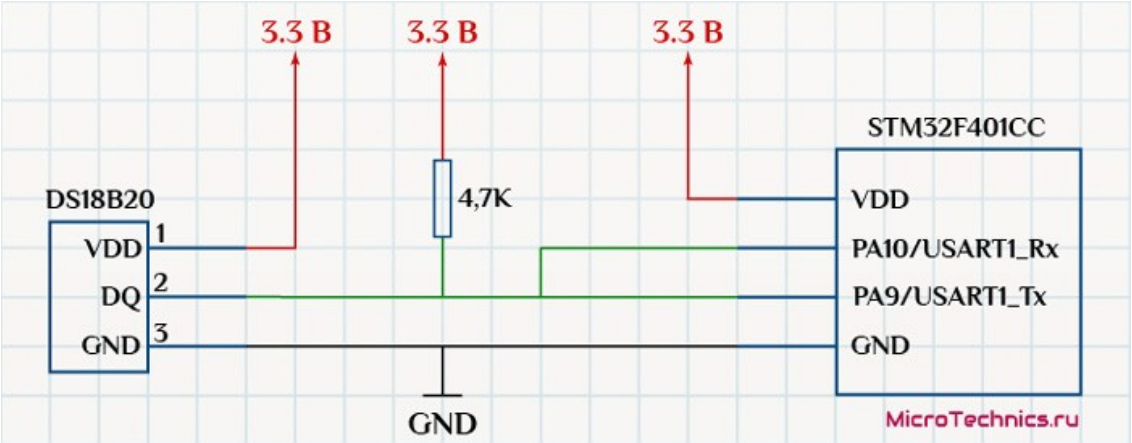


Рисунок 5 – Присоединение к плате

В данном случае приведена плата F401, но сути не меняет. К тому же, на плате F103RB можно переназначить любые пины на Wire.

1 ХОД РАБОТЫ

На рисунках 6 – 7 представлен код первой программы примера.

```
#include "stm32f10x.h"
#include "ds18b20.h"

int16_t raw_temp;
float temp;
uint8_t device_reset;

static void GPIO_Init(void)
{
    SET_BIT(RCC->APB2ENR, RCC_APB2ENR_IOPAEN);
    GPIOA->CRL &= ~((15ul << 4 * 5));
    GPIOA->CRL |= ((1ul << 4 * 5));
}

void LED_blinking(uint32_t delay_mcs)
{
    GPIOA->BSRR = 1 << 5;
    DelayMicro(delay_mcs);
    GPIOA->BSRR = 1 << 21;
    DelayMicro(delay_mcs);
}

int main(void)
{
    uint8_t scratchpad_data[9] = {0};
    uint8_t rom_data[8] = {0};
    uint8_t ds18b20_addr[8] = {0x28, 0xA0, 0x3F, 0xC1, 0x0B, 0x00, 0x00, 0xDE}; // адрес датчика (ROM-код)
    //uint8_t ds18b20_addr [8] = {0x28,0xEB,0xBB,0x08,0x0C,0x00,0x00,0x57};
    uint8_t crc8_rom = 0x0, crc8_rom_error = 1;
    uint8_t crc8_data = 0x0, crc8_data_error = 0;

    SysTick_Config(SystemCoreClock / 1000000); // SysTick настройка прерываний каждую микросекунду

    GPIO_Init(); // Инициализация порта A.5 для индикации светодиодом

    ds18b20_PortInit(); // Инициализация порта B.11 для передачи данных
    while (ds18b20_Reset()) ; // Если DS18B20 существует, продолжаем

    while (crc8_rom_error)
    { // Чтение ROM-кода до корректного значения
        ds18b20_ReadROM(rom_data); // Чтение ROM-кода для получения адреса датчика
        crc8_rom = Compute_CRC8(rom_data, 7); // Вычисление CRC для ROM-кода
        crc8_rom_error = Compute_CRC8(rom_data, 8) == 0 ? 0 : 1; // Сигнал об ошибке CRC
    }
```

Рисунок 6 – Код программы №1. Часть 1

```c
int main(void)
{
    uint8_t scratchpad_data[9] = {0};
    uint8_t rom_data[8] = {0};
    uint8_t ds18b20_addr[8] = {0x28, 0xA0, 0x3F, 0xC1, 0x0B, 0x00, 0x00, 0xDE}; // адрес датчика (ROM-код)
    //uint8_t ds18b20_addr [8] = {0x28,0xEB,0xBB,0x08,0x0C,0x00,0x00,0x57};
    uint8_t crc8_rom = 0x0, crc8_rom_error = 1;
    uint8_t crc8_data = 0x0, crc8_data_error = 0;

    SysTick_Config(SystemCoreClock / 1000000); // SysTick настройка прерываний каждую микросекунду

    GPIO_Init(); // Инициализация порта A.5 для индикации светодиодом

    ds18b20_PortInit(); // Инициализация порта B.11 для передачи данных
    while (ds18b20_Reset()) ; // Если DS18B20 существует, продолжаем

    while (crc8_rom_error)
    { // Чтение ROM-кода до корректного значения
        ds18b20_ReadROM(rom_data); // Чтение ROM-кода для получения адреса датчика
        crc8_rom = Compute_CRC8(rom_data, 7); // Вычисление CRC для ROM-кода
        crc8_rom_error = Compute_CRC8(rom_data, 8) == 0 ? 0 : 1; // Сигнал об ошибке CRC
    }

    ds18b20_Init(1, rom_data); // Инициализация и запись конфигурации в датчик

    while (1)
    {
        if (!crc8_data_error)
        {
            ds18b20_ConvertTemp(1, 0, rom_data); // Измерение температуры (отправка команды на преобразование температуры)
        }
        DelayMicro(750000); // Задержка для преобразования (максимум 750 мс для 12-битного результата)
        ds18b20_ReadStratchpad(1, scratchpad_data, 0, rom_data); // Чтение измеренной температуры
        crc8_data = Compute_CRC8(scratchpad_data, 8); // Вычисление CRC для данных
        crc8_data_error = Compute_CRC8(scratchpad_data, 9) == 0 ? 0 : 1; // Сигнал об ошибке CRC
        if (!crc8_data_error)
        { // Проверка корректности данных
            raw_temp = ((uint16_t)scratchpad_data[1] << 8) | scratchpad_data[0]; // Получение 16-битной температуры со знаком
            temp = raw_temp * 0.0625; // Преобразование в тип float
        }
        DelayMicro(100000);
    }
}

```

Рисунок 7 – Код программы №1. Часть 2

В коде выполняется следующее:

1. Импортируются необходимые заголовочные файлы, включая "stm32f10x.h" и "ds18b20.h".

2. Объявляются переменные raw_temp (для хранения сырого значения температуры), temp (для хранения преобразованного значения температуры) и device_reset (для обозначения состояния сброса устройства).

3. Определяется функция GPIO_Init(), которая инициализирует порт GPIOA и устанавливает пин A.5 в качестве выхода для индикации светодиодом.

4. Определяется функция LED_blinking(), которая мигает светодиодом путем установки и сброса соответствующего пина GPIOA с помощью задержки.

5. В функции main() объявляются массивы scratchpad_data, rom_data и ds18b20_addr для хранения данных о температуре, ROM-коде датчика и адресе датчика соответственно.

6. Задается переменная crc8_rom для хранения значения CRC для ROM-кода, а crc8_rom_error инициализируется значением 1 для обозначения ошибки CRC.

7. Задаются переменные crc8_data для хранения значения CRC для данных, а crc8_data_error инициализируется значением 0 для обозначения отсутствия ошибки CRC.

8. Настройка SysTick для генерации прерываний каждую микросекунду.

9. Инициализация порта GPIOA с помощью функции GPIO_Init().

10. Инициализация порта B.11 для передачи данных с помощью функции ds18b20_PortInit().

11. Проверка наличия DS18B20 путем вызова функции ds18b20_Reset(). Если DS18B20 присутствует, программа продолжает выполнение.

12. В цикле while происходит чтение ROM-кода до тех пор, пока не будет получено корректное значение. ROM-код считывается с помощью функции ds18b20_ReadROM(), а затем вычисляются значения CRC для ROM-кода с помощью функции Compute_CRC8(). Если CRC-значение равно 0, считается, что CRC прошла успешно, и значение crc8_rom_error устанавливается в 0.

13. Вызывается функция ds18b20_Init(), которая инициализирует датчик с заданными параметрами и записывает конфигурацию в датчик.

14. Внутри бесконечного цикла while (1) выполняются следующие действия:

a. Если crc8_data_error равно 0 (отсутствует ошибка CRC), то вызывается функция ds18b20_ConvertTemp() для запуска процесса измерения температуры. Параметры функции указывают на использование режима питания, передачу сигнала сброса и ROM-код датчика.

b. Затем следует задержка DelayMicro(750000) для ожидания окончания процесса преобразования температуры (максимальное время преобразования для разрешения 12 бит составляет 750 мс).

c. Вызывается функция ds18b20_ReadStratchpad(), чтобы считать измеренную температуру из Scratchpad. Параметры функции указывают на

номер датчика, буфер для хранения данных Scratchpad, использование режима питания и ROM-код датчика.

d. Вычисляется значение CRC для считанных данных Scratchpad с помощью функции Compute_CRC8(), и результат сохраняется в переменной crc8_data.

e. Проверяется значение crc8_data_error, и если оно равно 0 (отсутствует ошибка CRC), выполняются следующие действия:

– Сырые данные температуры извлекаются из буфера Scratchpad и сохраняются в переменной raw_temp.

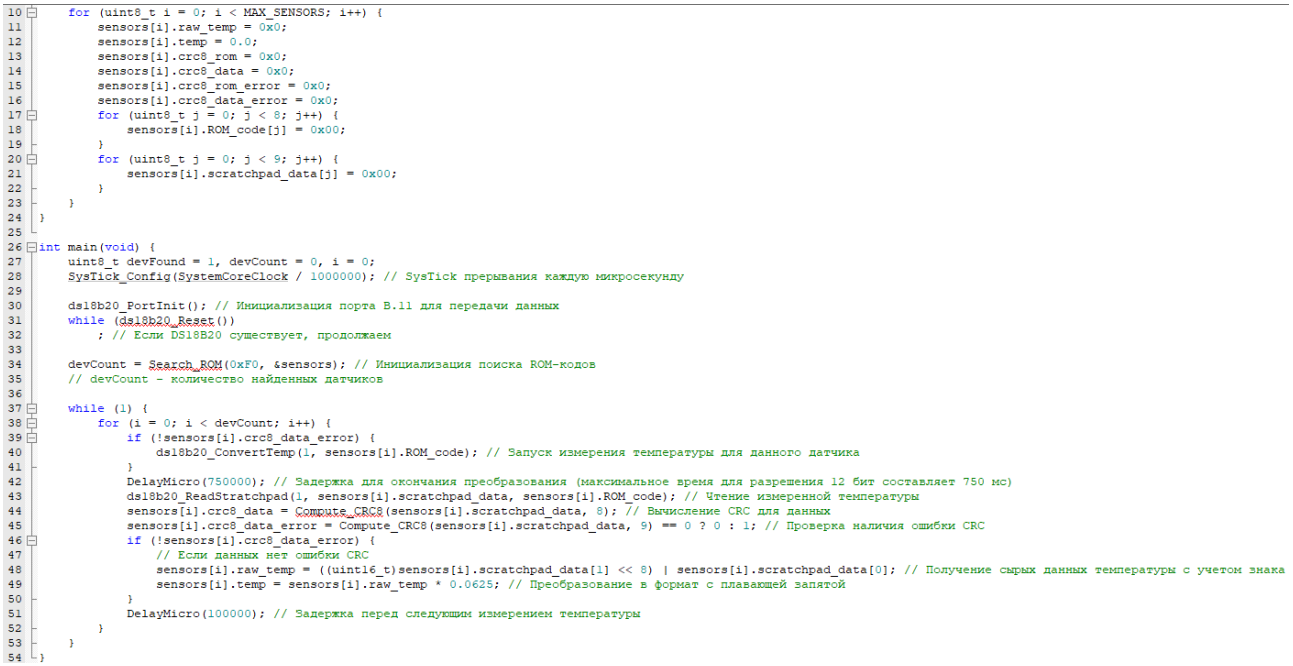
– Производится преобразование сырых данных температуры в формат с плавающей запятой, умножая их на 0.0625, и результат сохраняется в переменной temp.

f. Следует задержка DelayMicro(100000) перед следующей итерацией цикла для создания интервала между измерениями температуры.

На рисунках 8 – 9 представлен код второй программы примера.

```c
#include "stm32f10x.h"
#include "ds18b20.h"

#define MAX_SENSORS 2

Sensor sensors[MAX_SENSORS]; // массив структур для хранения данных сенсоров

void Init_Sensors() {
    // сброс всех значений в структурах сенсоров
    for (uint8_t i = 0; i < MAX_SENSORS; i++) {
        sensors[i].raw_temp = 0x0;
        sensors[i].temp = 0.0;
        sensors[i].crc8_rom = 0x0;
        sensors[i].crc8_data = 0x0;
        sensors[i].crc8_rom_error = 0x0;
        sensors[i].crc8_data_error = 0x0;
        for (uint8_t j = 0; j < 8; j++) {
            sensors[i].ROM_code[j] = 0x00;
        }
        for (uint8_t j = 0; j < 9; j++) {
            sensors[i].scratchpad_data[j] = 0x00;
        }
    }
}

int main(void) {
    uint8_t devFound = 1, devCount = 0, i = 0;
    SysTick_Config(SystemCoreClock / 1000000); // SysTick прерывания каждую микросекунду

    ds18b20_PortInit(); // Инициализация порта B.11 для передачи данных
    while (ds18b20_Reset())
        ; // Если DS18B20 существует, продолжаем

    devCount = Search_ROM(0xF0, &sensors); // Инициализация поиска ROM-кодов
    // devCount - количество найденных датчиков

    while (1) {
        for (i = 0; i < devCount; i++) {
            if (!sensors[i].crc8_data_error) {
                ds18b20_ConvertTemp(1, sensors[i].ROM_code); // Запуск измерения температуры для данного датчика
            }
            DelayMicro(750000); // Задержка для окончания преобразования (максимальное время для разрешения 12 бит составляет 750 мс)
            ds18b20_ReadStratchpad(1, sensors[i].scratchpad_data, sensors[i].ROM_code); // Чтение измеренной температуры
            sensors[i].crc8_data = Compute_CRC8(sensors[i].scratchpad_data, 8); // Вычисление CRC для данных
            sensors[i].crc8_data_error = Compute_CRC8(sensors[i].scratchpad_data, 9) == 0 ? 0 : 1; // Проверка наличия ошибки CRC
```

Рисунок 8 – Код программы №2. Часть 1



```
    for (uint8_t i = 0; i < MAX_SENSORS; i++) {
        sensors[i].raw_temp = 0x0;
        sensors[i].temp = 0.0;
        sensors[i].crc8_rom = 0x0;
        sensors[i].crc8_data = 0x0;
        sensors[i].crc8_rom_error = 0x0;
        sensors[i].crc8_data_error = 0x0;
        for (uint8_t j = 0; j < 8; j++) {
            sensors[i].ROM_code[j] = 0x00;
        }
        for (uint8_t j = 0; j < 9; j++) {
            sensors[i].scratchpad_data[j] = 0x00;
        }
    }
}

int main(void) {
    uint8_t devFound = 1, devCount = 0, i = 0;
    SysTick_Config(SystemCoreClock / 1000000); // SysTick прерывания каждую микросекунду

    ds18b20_PortInit(); // Инициализация порта B.11 для передачи данных
    while (ds18b20_Reset())
        ; // Если DS18B20 существует, продолжаем

    devCount = Search_ROM(0xF0, &sensors); // Инициализация поиска ROM-кодов
    // devCount - количество найденных датчиков

    while (1) {
        for (i = 0; i < devCount; i++) {
            if (!sensors[i].crc8_data_error) {
                ds18b20_ConvertTemp(1, sensors[i].ROM_code); // Запуск измерения температуры для данного датчика
            }
            DelayMicro(750000); // Задержка для окончания преобразования (максимальное время для разрешения 12 бит составляет 750 мс)
            ds18b20_ReadStratchpad(1, sensors[i].scratchpad_data, sensors[i].ROM_code); // Чтение измеренной температуры
            sensors[i].crc8_data = Compute_CRC8(sensors[i].scratchpad_data, 8); // Вычисление CRC для данных
            sensors[i].crc8_data_error = Compute_CRC8(sensors[i].scratchpad_data, 9) == 0 ? 0 : 1; // Проверка наличия ошибки CRC
            if (!sensors[i].crc8_data_error) {
                // Если данных нет ошибки CRC
                sensors[i].raw_temp = ((uint16_t)sensors[i].scratchpad_data[1] << 8) | sensors[i].scratchpad_data[0]; // Получение сырых данных температуры с учетом знака
                sensors[i].temp = sensors[i].raw_temp * 0.0625; // Преобразование в формат с плавающей запятой
            }
            DelayMicro(100000); // Задержка перед следующим измерением температуры
        }
    }
}
```

Рисунок 9 – Код программы №2. Часть 2

Ниже описано пошагово, что выполняется в коде:

1. Включение необходимых библиотек stm32f10x.h и ds18b20.h, которые содержат определения функций и структур для работы с микроконтроллером STM32F10x и датчиком DS18B20.

2. Определение максимального количества поддерживаемых датчиков MAX_SENSORS для установки размера массива структур датчиков.

3. Определение структуры Sensor, которая содержит переменные для хранения информации о датчике, такие как сырые данные температуры, преобразованная температура, значения CRC и т.д.

4. Объявление массива структур sensors, который будет использоваться для хранения данных о датчиках. Размер массива определяется константой MAX_SENSORS.

5. Определение функции Init_Sensors, которая инициализирует значения структур датчиков. В данном случае, все значения переменных структур устанавливаются в ноль.

6. Объявление переменных devFound, devCount и i, которые будут использоваться для управления поиском и обработкой датчиков.

7. Настройка прерывания SysTick с частотой 1 микросекунда. SysTick используется для создания микропауз в программе.

8. Вызов функции ds18b20_PortInit, которая инициализирует порт B.11 для передачи данных между микроконтроллером и датчиком DS18B20.

9. Цикл while для проверки наличия датчика DS18B20. Функция ds18b20_Reset вызывается до тех пор, пока датчик не будет обнаружен. Когда датчик обнаружен, цикл завершается и выполняется следующий шаг.

10. Вызов функции Search_ROM, которая инициализирует процесс поиска ROM-кодов датчиков. Первый аргумент функции указывает тип поиска, а второй аргумент - указатель на массив структур датчиков sensors, куда будут сохранены найденные ROM-коды.

11. Цикл while (1) - бесконечный цикл, в котором выполняется основная логика программы для обработки данных с датчиков.

12. Цикл for для обработки каждого датчика в массиве sensors. Переменная i используется для индексации датчиков в массиве.

13. Проверка флага crc8_data_error для каждого датчика. Если флаг равен 0, значит данные температуры корректны и можно выполнить преобразование температуры.

14. Вызов функции ds18b20_ConvertTemp для каждого датчика. Функция отправляет команду на датчик для начала процесса измерения температуры.

15. Задержка с помощью функции DelayMicro для ожидания завершения преобразования температуры. Длительность задержки составляет 750 миллисекунд, что является максимальным временем для преобразования 12-битного значения температуры.

16. Вызов функции ds18b20_ReadStratchpad для каждого датчика. Функция считывает данные о температуре из датчика в массив scratchpad_data для каждого датчика.

17. Расчет значения CRC для данных каждого датчика с помощью функции Compute_CRC8.

18. Проверка флага crc8_data_error для каждого датчика. Если флаг равен 0, значит CRC данных температуры корректно вычислено и можно преобразовать сырые данные в температуру.

19. Задержка с помощью функции DelayMicro перед следующим измерением температуры.

Заключение

В ходе лабораторной работы были получены практические навыки работы с интерфейсом 1-Wire на основе CMSIS (Common Microcontroller Software Interface Standard). Были рассмотрены программы примеры, представленные в курсе.

Отчет оформлен согласно ОС ТУСУР 01-2021.