МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Информационная безопасность систем и технологий»

Отчет
по лабораторной работе №3
на тему «Работа с последовательным интерфейсомпередачиданныхUART
микроконтроллера семейства STM32»

Дисциплина: ПМК
Группа: 21ПИ1
Выполнил: Гусев Д. А.
Количество баллов:
Дата сдачи:
Принял: Хворостухин С. П .

2024

1 Цель работы: Ознакомиться с программными средствами работыпопоследовательному интерфейсу UART микроконтроллера STM32..

2 Задание на лабораторную работу.

2.1 Получить вариант задания у преподавателя.

2.2 Рассчитать значение делителя частоты и регистра BRR контроллера USART в соответствии с заданием.

2.3 Создать проект в среде Keil uVision5 для микроконтроллераSTM32F103RB.

2.4 Выбрать программные компоненты: CMSIS/Core, Device/Startup, Device/StdPeriph Drivers/Framework, Device/StdPeriph Drivers/GPIO, Device/StdPeriph Drivers/RCC; Device/StdPeriph Drivers/TIM; Device/StdPeriph Drivers/USART.

2.5 Выполнить настройку режима отладки для проекта.

2.6 Разработать программу согласно задания.

2.7 Выполнить симуляцию разработанной программы.

2.8 Зафиксировать результаты функционирования программы: настройки аппаратных средств; содержимое терминала последовательного интерфейса; параметры формируемых сигналов.

2.9 Сделать выводы по проделанной работе и оформить отчет

3 Выполнение лабораторную работы:

3.1 Был получен вариант задания – 8. Данные для варианта заданя представлены в таблице 1.

Таблица 1 — Вариант задания №8

Номер варианта	Контроллер	Скорость передачи, бод
8	USART2	2400

3.2 Был создан проект в среде Keil uVision5 для микроконтроллера STM32F103RB. Результат представлен на рисунке 1.

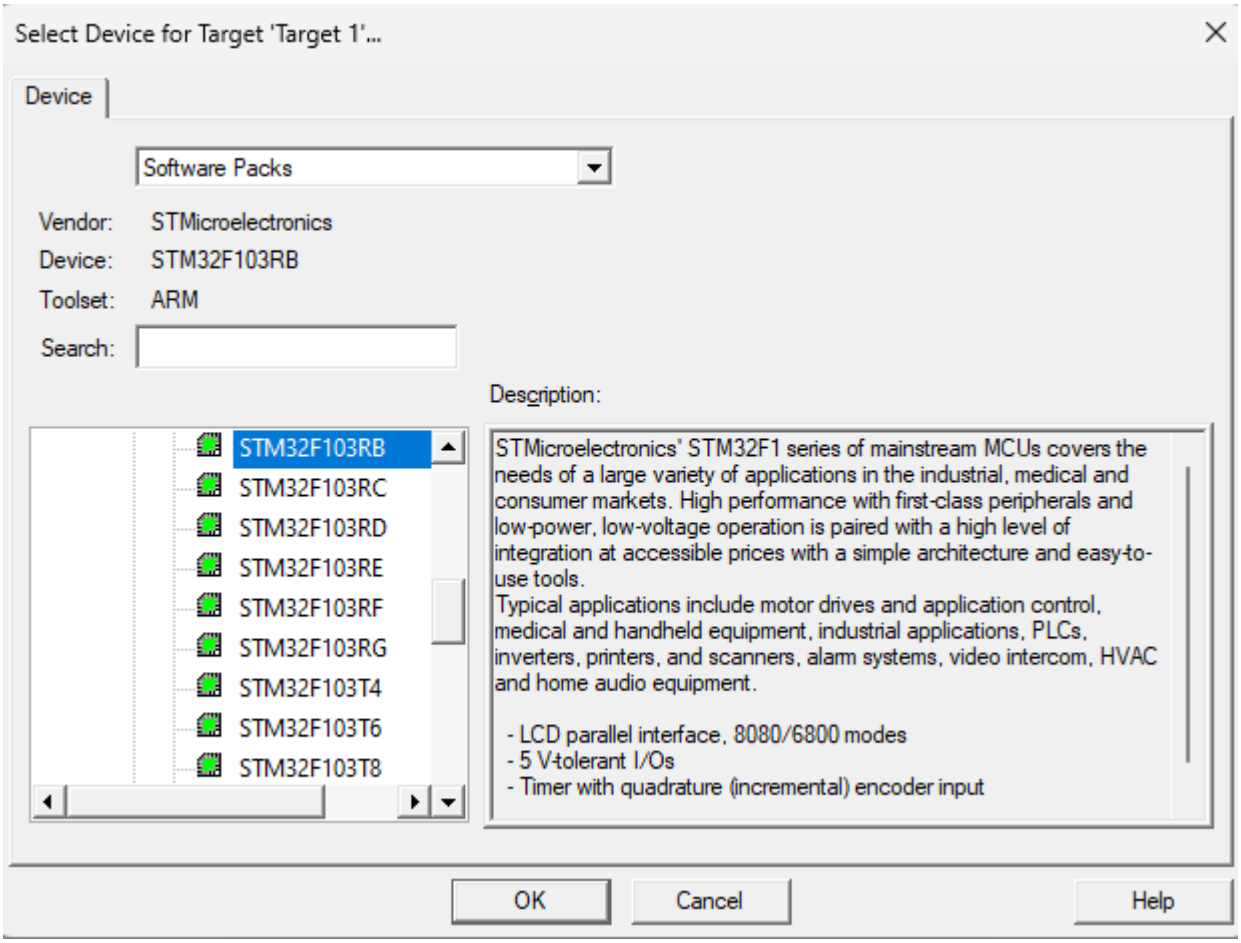

Рисунок 1 — Создание проекта для микроконтроллера STM32F103RB

3.3 Были выбраны программные компоненты: CMSIS/Core, Device/Startup, Device/StdPeriph Drivers/Framework, Device/StdPeriph Drivers/GPIO, Device/StdPeriph Drivers/RCC; Device/StdPeriph Drivers/TIM; Device/StdPeriph Drivers/USART. Результат представлен на рисунке 2.

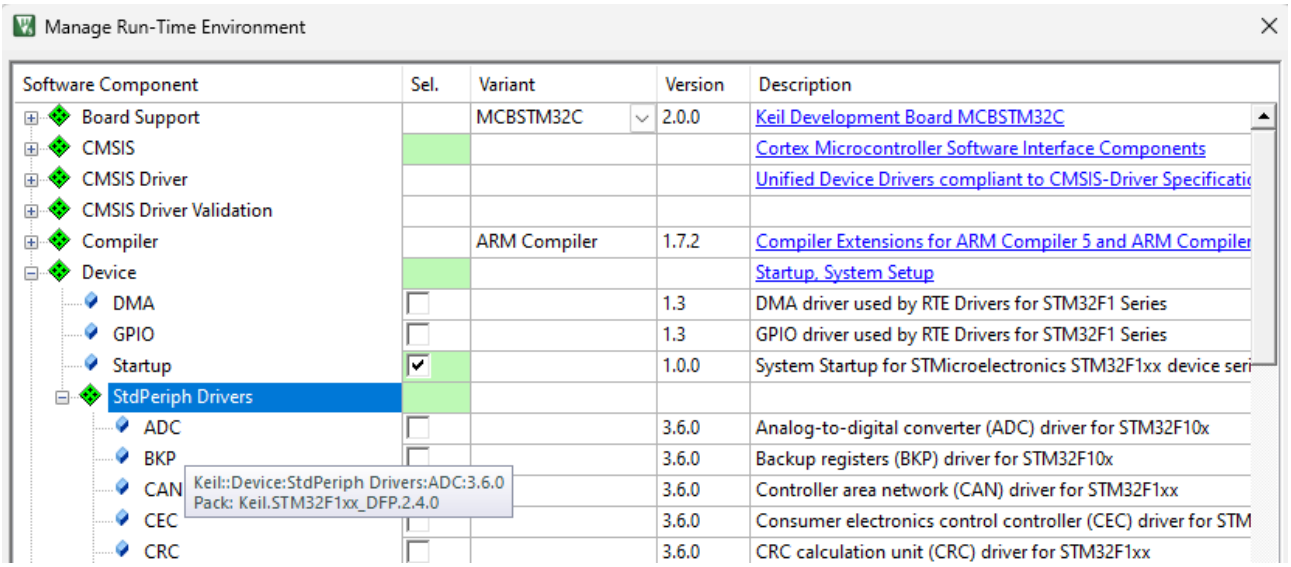

Рисунок 2 — Выбор компонентов

3.4 Была выполнена настройка режима отладки для проекта. Для этого в папке проекта был создан файл MAP.ini со следующим содержанием:

```
MAP 0x40000000, 0x47FFFFFF READ WRITE;
```

Далее была открыта вкладка «Option for Target...» затем была открыта вкладка «Debug», был включен переключатель «UseSimulator». В поле «DialogDLL» было записано DARMSTM.DLL, в поле «Parameter» было записано: - pSTM32F103RB. Был установлен путь к файлу MAP.ini в поле «InitializationFile». Результат представлен на рисунке 3.

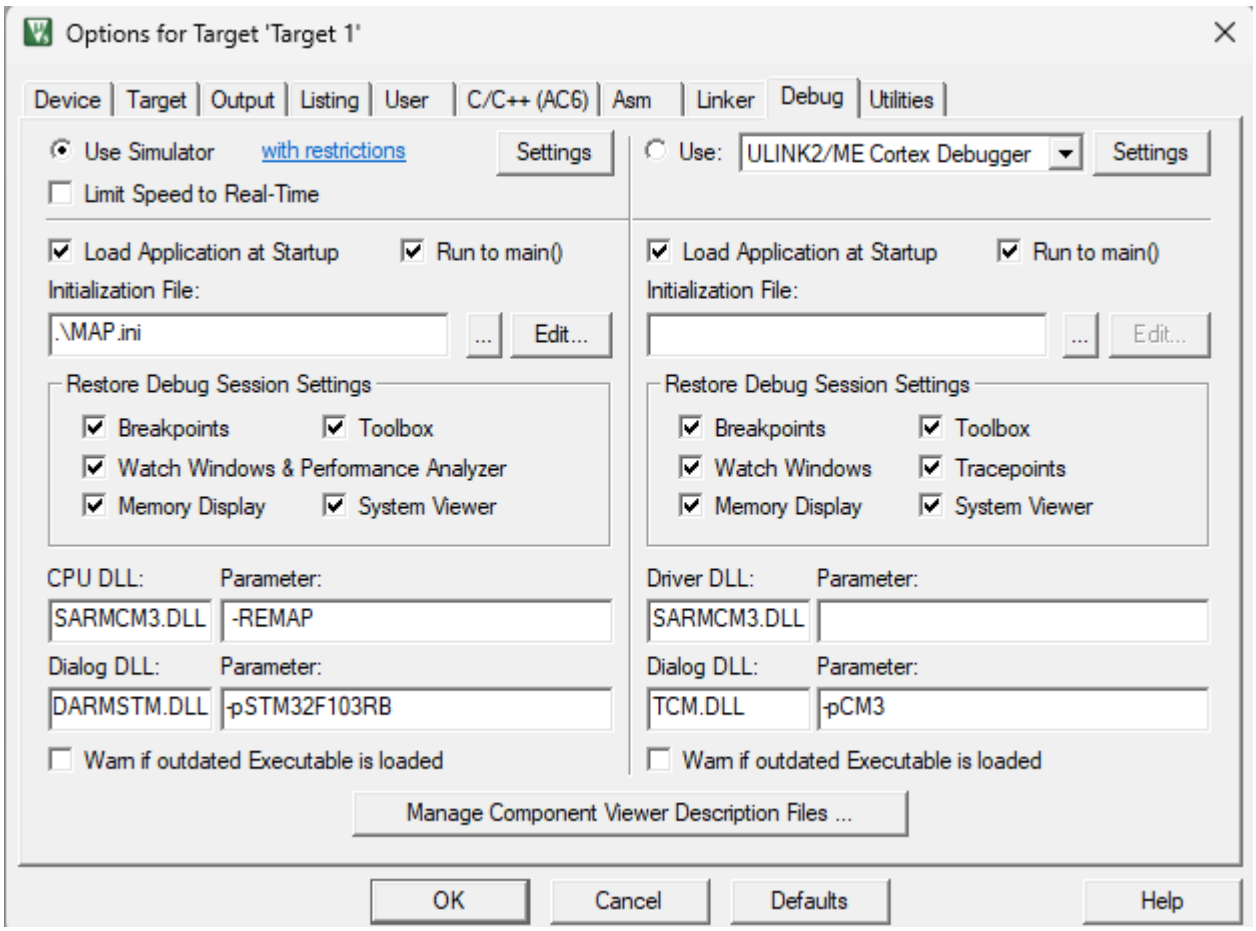

Рисунок 3 — Настройка проекта

3.5 Была разработана программа, реализующая: инициализацию портов ввода-вывода; инициализацию таймера в режиме захвата сигнала; инициализацию таймера в режиме широтно-импульсной модуляции; обработку прерываний таймеров. Код программы представлен в Приложении А.

3.6 Была выполнена симуляция разработанной программы с использованием функций отладки, а также зафиксированы результаты функционирования программы. Результаты представлены на рисунке 4.

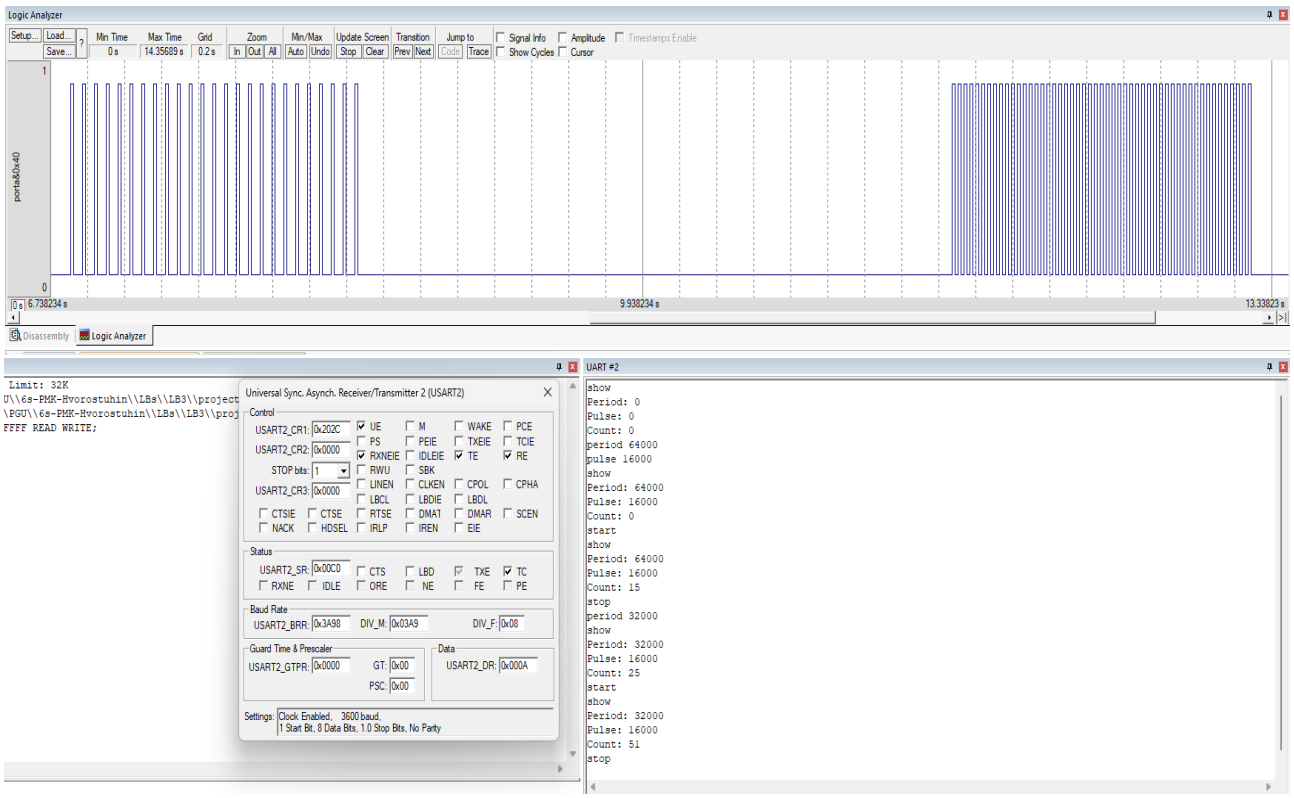

Рисунок 4 - Симуляция

4 Вывод: было выполнено ознакомление с программными средствами работы по последовательному интерфейсу UART микроконтроллера STM32.

Приложение А

Код программы

```
#include "stm32f10x.h"       // Подключение библиотеки общих определений для STM32F10x
#include "stm32f10x_gpio.h"  // Подключение библиотеки для работы с GPIO
#include "stm32f10x_rcc.h"   // Подключение библиотеки для работы с RCC
#include "stm32f10x_tim.h"   // Подключение библиотеки для работы с таймерами
#include "stm32f10x_usart.h" // Подключение библиотеки для работы с USART

/* Объявление структуры для инициализации канала таймера */
static TIM_OCInitTypeDef ChObj;
/* Объявление структуры для инициализации таймера */
static TIM_TimeBaseInitTypeDef TimObj;

static char buffer;      // Буфер для хранения принятых данных
static char xbuffer[64]; // Буфер для хранения массива данных
static uint8_t xlen = 0; // Длина буфера
static uint32_t counter = 0;

// Функция вывода массива байтов
static void sendArr(char *data)
{
    uint64_t i = 0;
    while (data[i])
    {
        USART_SendData(USART2, data[i]);
        while (!USART_GetFlagStatus(USART2, USART_FLAG_TXE))
        {
        }
        i++;
    }
}

// Функция вывода байта
static void sendChar(char data)
{
```

```
    USART_SendData(USART2, data);
    while (!USART_GetFlagStatus(USART2, USART_FLAG_TXE))
    {
    }
}

// Функция очистки буфера
static void clearXBuff(void)
{
    uint8_t i;
    for (i = 0; i <= xlen; i++)
        xbuffer[i] = 0;
    xlen = 0;
    buffer = 0;
}
static uint16_t getArg()
{
    uint8_t i;
    uint8_t begin = 0;
    uint16_t arg = 0;

    for (i = 0; i < xlen; i++)
    {
        if (xbuffer[i] == ' ')
            begin = i + 1;
    }
    if (!begin)
        return arg;

    for (i = begin; i < xlen; i++)
        arg = arg * 10 + (xbuffer[i] - '0');
    return arg;
}

// Функция сравнения массива байтов с буфером массива байтов
static int isXBuff(char *data)
{
```

```
    uint8_t i;
    for (i = 0; i < sizeof(data); i++)
    {
        if (data[i] != xbuffer[i])
            return 0;
    }
    return 1;
}

static void initGPIO(void) // Функция инициализации GPIO
{
    GPIO_InitTypeDef PortObj;                              // Объявление структуры для инициализации GPIO
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE); // Включение тактирования порта A

    /* Настройка порта PA9 */
    PortObj.GPIO_Pin = GPIO_Pin_2;          // Настройка пина 9
    PortObj.GPIO_Speed = GPIO_Speed_50MHz; // Установка скорости GPIO
    PortObj.GPIO_Mode = GPIO_Mode_AF_PP;   // Установка режима альтернативной функции push-pull
    GPIO_Init(GPIOA, &PortObj);            // Инициализация GPIOA с использованием структуры Obj

    /* Настройка порта PA10 */
    PortObj.GPIO_Pin = GPIO_Pin_3;              // Настройка пина 10
    PortObj.GPIO_Mode = GPIO_Mode_IN_FLOATING; // Установка режима входа с подтяжкой к "плавающему" уровню
    GPIO_Init(GPIOA, &PortObj);                // Инициализация GPIOA с использованием структуры Obj

    /* Настройка порта PA3 */
    PortObj.GPIO_Pin = GPIO_Pin_6;        /* Настройка пина 3 */
    PortObj.GPIO_Mode = GPIO_Mode_AF_PP;  /* Режим альтернативной функции, push-pull */
    PortObj.GPIO_Speed = GPIO_Speed_2MHz; /* Скорость порта 2 МГц */
    GPIO_Init(GPIOA, &PortObj);           /* Применение настроек к порту A */
}

static void initUSART(void) // Функция инициализации USART
```

```
{
    USART_InitTypeDef UsartObj;                                 // Объявление структуры для
инициализации USART

    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE); // Включение тактирования
USART2


    UsartObj.USART_BaudRate = 2400;                                         // Установка
скорости передачи данных

    UsartObj.USART_WordLength = USART_WordLength_8b;                        // Установка
длины слова

    UsartObj.USART_StopBits = USART_StopBits_1;                             // Установка
количества стоп-битов

    UsartObj.USART_Parity = USART_Parity_No;                                // Установка
контроля четности

    UsartObj.USART_HardwareFlowControl = USART_HardwareFlowControl_None; // Установка
аппаратного управления потоком

    UsartObj.USART_Mode = USART_Mode_Tx | USART_Mode_Rx;                    // Установка
режима передачи и приема

    USART_Init(USART2, &UsartObj);                                          //
Инициализация USART2 с использованием структуры Obj


    USART_ITConfig(USART2, USART_IT_RXNE, ENABLE); // Включение прерывания по приему
данных

    USART_Cmd(USART2, ENABLE);                      // Включение USART2

    NVIC_EnableIRQ(USART2_IRQn);

    NVIC_SetPriority(USART2_IRQn, 1);

}


void USART2_IRQHandler(void) // Обработчик прерывания USART2

{

    if (USART_GetFlagStatus(USART2, USART_FLAG_RXNE)) // Если флаг RXNE установлен

    {

        USART_ClearITPendingBit(USART2, USART_FLAG_RXNE); // Очистка флага прерывания
RXNE

        buffer = USART_ReceiveData(USART2);              // Чтение данных из регистра
приема USART2


        if (xlen + 1 >= sizeof(xbuffer))

        {

            sendArr("\nOverflow buffer\n");

            clearXBuff();

            return;
```

```
}

if (buffer != 0x0D)
{
    xbuffer[xlen] = buffer;
    sendChar(buffer);
    xlen++;
    return;
}

if (isXBuff("show"))
{
    clearXBuff();
    sprintf(xbuffer, "%d", TimObj.TIM_Period);
    sendArr("\nPeriod: ");
    sendArr(xbuffer);

    clearXBuff();
    sprintf(xbuffer, "%d", ChObj.TIM_Pulse);
    sendArr("\nPulse: ");
    sendArr(xbuffer);

    clearXBuff();
    sprintf(xbuffer, "%d", counter);
    sendArr("\nCount: ");
    sendArr(xbuffer);

    clearXBuff();
    sendArr("\n");
    return;
}

if (isXBuff("start"))
{
    if (TimObj.TIM_Period & ChObj.TIM_Pulse)
        TIM_Cmd(TIM3, ENABLE);
    clearXBuff();
```

```
    sendArr("\n");
    return;
}

if (isXBuff("stop"))
{
    TIM_Cmd(TIM3, DISABLE);
    clearXBuff();
    sendArr("\n");
    return;
}

if (isXBuff("period"))
{
    TimObj.TIM_Period = getArg();
    TIM_TimeBaseInit(TIM3, &TimObj);
    TIM_OC1Init(TIM3, &ChObj);
    counter--; // так как инициализация вызывает прерывание IT_Update
    sendArr("\n");
    clearXBuff();
    return;
}

if (isXBuff("pulse"))
{
    ChObj.TIM_Pulse = getArg();
    TIM_TimeBaseInit(TIM3, &TimObj);
    TIM_OC1Init(TIM3, &ChObj);
    counter--; // так как инициализация вызывает прерывание IT_Update
    clearXBuff();
    sendArr("\n");
    return;
}

clearXBuff();
sendArr("\nInvalid command\n");
return;
```

```
    }
}
static void initTIM3(void)
{
    /* Включение тактирования таймера TIM3 */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);

    /* Настройка периода и прескеллера TIM3 */
    TimObj.TIM_Period = 0;
    TimObj.TIM_Prescaler = 107;

    /* Настройка делителя частоты, режим счета таймера TIM3 */
    TimObj.TIM_ClockDivision = 0;                /* Делитель частоты таймера */
    TimObj.TIM_CounterMode = TIM_CounterMode_Up; /* Режим счета вверх */

    /* Настройка канала 1 в режиме PWM */
    ChObj.TIM_OCMode = TIM_OCMode_PWM1;             /* Режим работы канала - PWM1 */
    ChObj.TIM_OutputState = TIM_OutputState_Enable; /* Включение выхода канала */
    ChObj.TIM_OCPolarity = TIM_OCPolarity_High;     /* Полярность выходного сигнала -
прямая */
    ChObj.TIM_Pulse = TimObj.TIM_Period / 2;

    TIM_OC1Init(TIM3, &ChObj);                 /* Применение настроек к каналу 1
таймера TIM3 */
    TIM_ITConfig(TIM3, TIM_IT_Update, ENABLE); /* Включение прерывания на канале 1 */

    NVIC_EnableIRQ(TIM3_IRQn);
    NVIC_SetPriority(TIM3_IRQn, 2);
}

void TIM3_IRQHandler(void)
{
    /* Если произошло прерывание по событию захвата на канале 4 */
    if (TIM_GetITStatus(TIM3, TIM_IT_Update) != RESET)
    {
        TIM_ClearITPendingBit(TIM3, TIM_IT_Update); /* Сброс флага прерывания */
        counter++;
    }
```

```
}

int main(void)
{
    initGPIO();  // Инициализация GPIO
    initUSART(); // Инициализация USART
    initTIM3();  // Инициализация TIM3
    while (1)
    {
    }
}
```