ГУАП

КАФЕДРА № 43

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

к.т.н., доцент		А. А. Попов
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ

Программирование встроенных приложений

по курсу: Программирование встроенных приложений

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ гр. № 4131з		М. Д. Быстров
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2025

Результат выполнения работы

Раздел 2

Задание

Создайте проект. Задайте размеры стека и heap согласно формуле:

0x40+0x80x[номер варианта]_16. Переменным: a1, bl, cl типа unsigned char;

a2, b2, c2 типа unsigned short;

a4, b4, c4 типа unsigned int;

a8, b8, c8, типа unsigned long long присвоить повторяющиеся значение

0x11+0x9×[номер варианта]_16 (пример: номер варианта 26_10=1A_16, 0x11+0x9×0x1A=0xFB, a1=0xFB, ., c4=0xFBFBFBFB, ...), переменной name1 присвоить своё имя, переменной name2 фамилию в латинской транскрипции, name3 номер группы.

2) Найти в файле карты компоновки (map-файле): затраты оперативной (RAM) и постоянной (ROM) памяти МК для вашего проекта; адрес расположения и размер стека, heap; адрес расположения и размер таблицы векторов; адрес расположения и размер функции main ().

3) Проанализировать переменные a1-c8, name1-3 инструментами отладки ИСР Keil. Определить адреса переменных. По найденным адресам определить расположение в памяти. Сохранить отпечаток всей области памяти этих переменных в файл logdat.txt.

4) Оформить отчёт.

Отчёт должен содержать:

1) Текст задания согласно варианту, с указанием размера стека, значений переменных.

2) Программу проекта.

3) Выписку из файла карты компоновки: затрат оперативной и постоянной памяти проекта; адрес расположения и размер стека, heap; адрес расположения и размер таблицы векторов; адрес расположения и размер функции main.

4) Адреса расположения в памяти переменных a1, b1,-,b8, c8, name1/2/3 (подобно рис. 2.6).

5) Содержимое файла logdat. txt с отпечатком всей области памяти содержащей значения переменных al,b1,..,b8,c8, namel/2/3.

Выполнение задания

Вариант 4 – Быстров М.Д.

Номер варианта – $4_{10} = 0x4_{16}$

Размер heap, stack = 0x40+0x80*0x4=$0x240_{16}$ байт

Повторяющееся значение для переменных = 0x11+0x9*0x4=$0x35_{16}$

Текст программы:

```
int main () {
        volatile unsigned char a1 = 0x35;
        volatile unsigned char b1 = 0x35;
        volatile unsigned char c1 = 0x35;
        volatile unsigned short a2 = 0x3535;
        volatile unsigned short b2 = 0x3535;
        volatile unsigned short c2 = 0x3535;
        volatile unsigned int a4 = 0x35353535;
        volatile unsigned int b4 = 0x35353535;
        volatile unsigned int c4 = 0x35353535;
        volatile unsigned long long a8 = 0x3535353535353535;
        volatile unsigned long long b8 = 0x3535353535353535;
        volatile unsigned long long c8 = 0x3535353535353535;
        volatile char name1[] = "Maxim";
        volatile char name2[] = "Bystrov";
        volatile char name3[] = "4131z";
        for(;;){}
        return 0;
}
```

Затраты оперативной и постоянной памяти:

```
Total RO  Size (Code + RO Data)                 1112 (   1.09kB)
Total RW  Size (RW Data + ZI Data)              1248 (   1.22kB)
Total ROM Size (Code + RO Data + RW Data)        1112 (   1.09kB)
```

Адрес расположения и размер стека, heap:

```
Exec Addr    Load Addr    Size         Type   Attr      Idx    E Section Name
0x20000060       -       0x00000240   Zero   RW         12    HEAP
0x200002a0       -       0x00000240   Zero   RW         11    STACK
```

Адрес расположения и размер таблицы векторов:

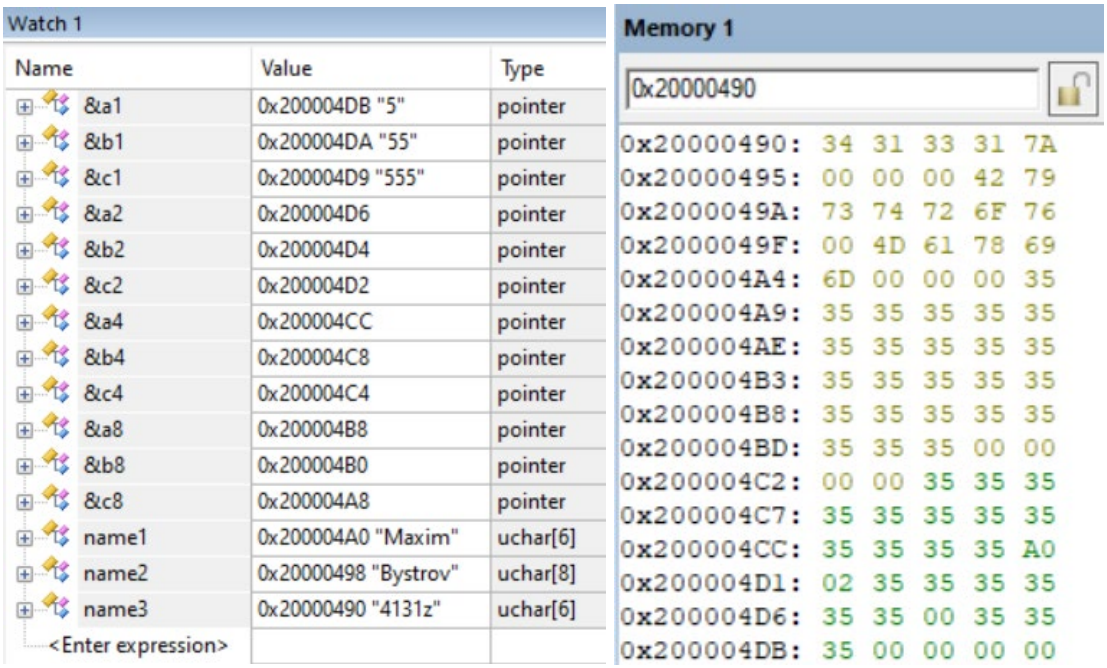
```
Exec Addr    Load Addr    Size         Type   Attr      Idx    E Section Name        Object
```

0x08000000 0x08000000 0x000000ec Data RO 13 RESET startup_stm32f10x_md.o

Адрес расположения и размер функции main:

Exec Addr	Load Addr	Size	Type	Attr	Idx	E Section Name	Object
0x080003bc	0x080003bc	0x00000078	Code	RO	2	.text.main	main.o
0x08000434	0x08000434	0x00000014	Data	RO	4	.rodata.str1.1	main.o

Адреса расположения в памяти переменных a1, b1,-,b8, c8, name1/2/3



Содержимое файла logdat.txt с отпечатком всей области памяти содержащей значения переменных al,b1,..,b8,c8, namel/2/3:

```
0x20000490  34 31 33 31 7A 00 00 00 - 42 79 73 74 72 6F 76 00 4131z...Bystrov.
0x200004A0  4D 61 78 69 6D 00 00 00 - 35 35 35 35 35 35 35 35 Maxim...55555555
0x200004B0  35 35 35 35 35 35 35 35 - 35 35 35 35 35 35 35 35 5555555555555555
0x200004C0  00 00 00 00 35 35 35 35 - 35 35 35 35 35 35 35 35 ....555555555555
0x200004D0  A0 02 35 35 35 35 35 35 - 00 35 35 35             ..555555.555
```

Раздел 3

Задание

В соответствии с вариантом (табл. 3.1) нужно написать на языке «си» программу и отладить её работу по переключению уровня сигнала на двух линиях в/в микроконтроллера STM32F103C8T6. Значение частоты измерять инструментом отладчика Logic Analyzer. В программе один из выводов настраивать через регистры

без использования библиотеки CMSIS. Значение счётчика задержки, под заданную вариантом частоту переключения линии в/в, подбирать вручную или рассчитать (тактовая частота 72МГц). Сохранить эпюры сигналов в отчёт.

Отчет должен содержать:

1. Номер варианта с заданием. Исходный код программы.
2. Таблицу трассировки двух заданных выводов STM32F103x8.
3. Таблицу используемых регистров STM32F103x8 с расчетом адресов (с указанием на документацию) и управляемые биты.
4. Две эпюры сигналов на линиях в/в, по образцу рисунка 3.3. Снимок окна Watch со значением OSC. И таблица с характеристиками сигналов (частоты и периода).

1. Номер варианта с заданием

Вариант	4
Номера выводов	15, 31
Частота переключения	640 Гц; 1280 Гц

Исходный код программы:

```
#include "RTE_Components.h"
#include CMSIS_device_header

void delay () {
        volatile uint32_t count = 4323;
        while (count--)
        __NOP();
}

int main() {
        // PA5:640Hz, PA10:1280Hz

        // addresses of registers for PA5
        uint32_t* _APB2ENR = (uint32_t*) (0x40021018);
        uint32_t* _GPIOA_CRL = (uint32_t*) (0x40010800);
        uint32_t* _GPIOA_BSRR = (uint32_t*) (0x40010810);
        uint32_t* _GPIOA_BRR = (uint32_t*) (0x40010814);

        // enable GPIOA (2th bit)
        *_APB2ENR |= 1 << 2;

        // setup PA5 in configuration register low
        *_GPIOA_CRL &= 0xFF0FFFFF;
        *_GPIOA_CRL |= 0x00200000;
```

```
        // setup PA10 in configuration register high
        GPIOA->CRH &= ~(GPIO_CRH_MODE10 | GPIO_CRH_CNF10);
        SET_BIT(GPIOA->CRH, GPIO_CRH_MODE10_1);


        // bitmask of PA5 for BRR/BSRR
        uint32_t PA5_BIT = 1 << 5;

        for (;;) {
                // PA5 = 1; PA10 = 1
                * _GPIOA_BSRR = PA5_BIT;
                GPIOA->BSRR = GPIO_BSRR_BS10;
                delay();

                // PA5 = 1; PA10 = 0
                * _GPIOA_BSRR = PA5_BIT;
                GPIOA->BRR = GPIO_BRR_BR10;
                delay();

                // PA5 = 0; PA10 = 1
                * _GPIOA_BRR = PA5_BIT;
                GPIOA->BSRR = GPIO_BSRR_BS10;
                delay();

                // PA5 = 0; PA10 = 0
                * _GPIOA_BRR = PA5_BIT;
                GPIOA->BRR = GPIO_BRR_BR10;
                delay();
        }
        return 0;
}
```

2. Таблица трассировки выводов STM32F103x8

Номера выводов	Обозначение согласно DS5319	Номера разъемов и выводов на отладочной плате
15	PA5	Разъем номер 10 на однорядном штыревом разъеме P3
31	PA10	Разъем номер 14 на однорядном штыревом разъеме P4

3. Таблица используемых регистров STM32F103x8 с расчетом адресов и управляемые биты.

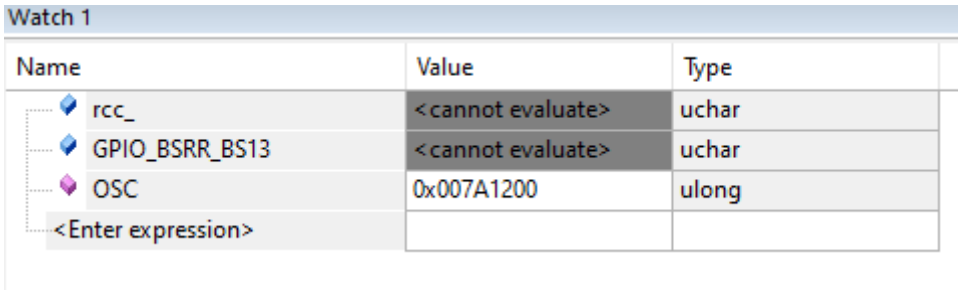
Подсистема/регистр	Расчет адреса и ссылки на	Биты и их назначение

	документацию	
RCC_APB2ENR	Адрес регистра RCC_APB2ENR: 0x40021000+0x18=0x40021018 RM0008 3.3 RM0008 8.3.7	Бит 2 IOPAEN: включает ПВВ A (GPIOA)
GPIOA_CRL	Адрес регистра GPIOA_CRL: 0x40010800+0x00=0x40010800 RM0008 3.3 RM0008 9.2.1	Конфигурация линии №5 ПВВ A. Поля бит: 21:20 CNF5[1:0] 23:22 MODE5[1:0]
GPIOA_CRH	Адрес регистра GPIOA_CRH: 0x40010800+0x04=0x40010804 RM0008 3.3 RM0008 9.2.2	Конфигурация линии №10 ПВВ A. Поля бит: 9:8 CNF10[1:0] 11:10 MODE10[1:0]
GPIOA_BSRR	Адрес регистра GPIOA_BSRR: 0x40010800+0x10=0x40010810 RM0008 9.2.5	Бит 5 BS5: устанавливает единицу на линии №5 ПВВ A (5й бит регистра GPIOA_ODR) Бит 10 BS10: устанавливает единицу на линии №10 ПВВ A (10й бит регистра GPIOA_ODR)
GPIOA_BRR	Адрес регистра GPIOA_BRR: 0x40010800+0x14=0x40010814 RM0008 9.2.6	Бит 5 BS5: устанавливает нуль на линии №5 ПВВ A (5й

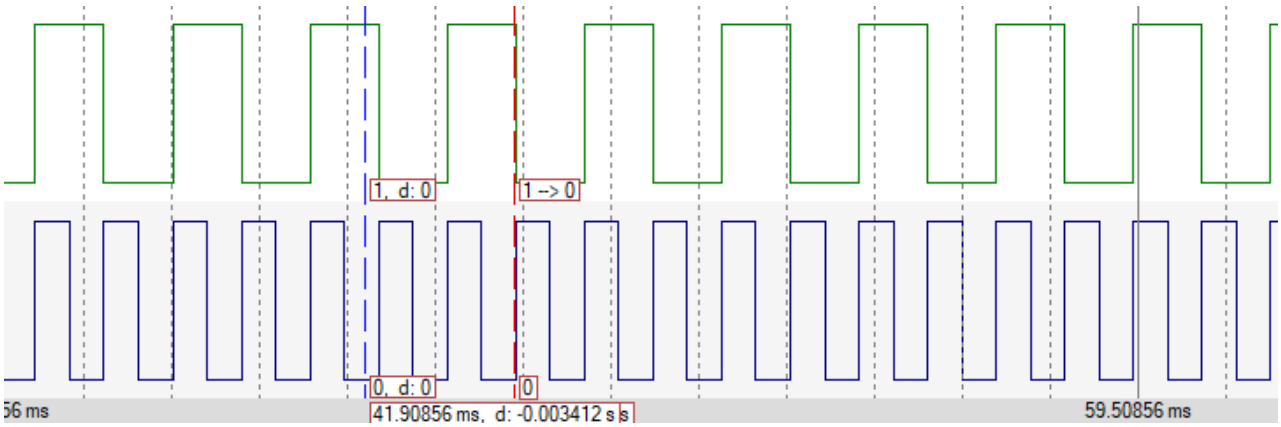
		бит регистра GPIOA_ODR) Бит 10 BS10: устанавливает нуль на линии №10 ПВВ A (10й бит регистра GPIOA_ODR)

4. Эпюры сигналов на линиях в/в

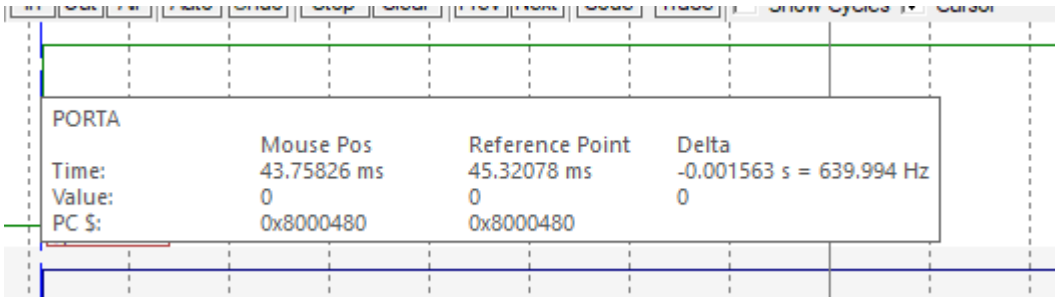
Частота OSC:



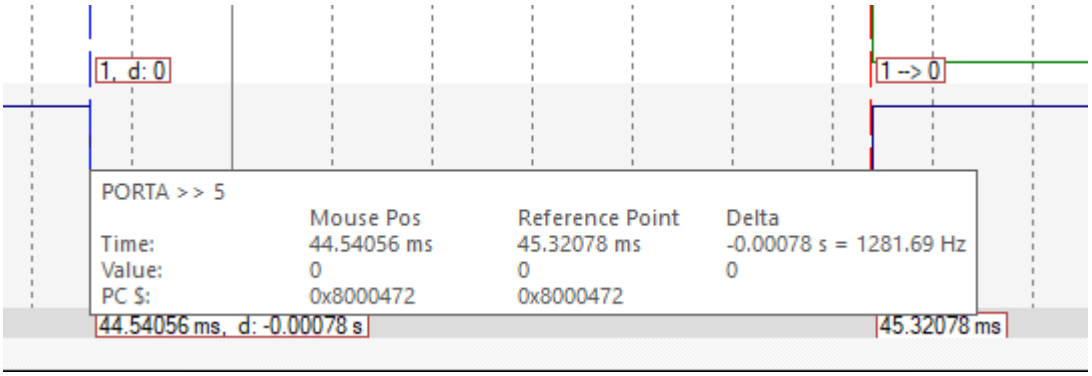
Эпюры сигналов:



Измерение сигнала PA5:



Измерение сигнала PA10:



Характеристика	Линия PA5	Линия PA10
Период, мс	1,563	0,78
Частота, Гц	639.8	1282

Раздел 4.

Задание.

В соответствии с вариантом (табл. 4.2) нужно написать на языке «си» программу и

отладить её работу, в которую входит:

а) настройка частоты тактирования МК согласно варианту:

б) переключение в цикле заданного вывода;

в) настройка МСО на вывод сигнала согласно варианту. Вывод сигнала МСО на линию

PA8 в симуляторе отсутствует, поэтому правильность настройки не проверить.

Используя инструмент Logic Analyzer измерить эпюру сигнала заданного вывода.

Включить оптимизацию компиляции выставив опцию Optimization: в положение -03.

Пересобрать проект, снова измерить эпюру сигнала, сравнить с полученной ранее.

Вариант 4.

Варианты индивидуальных заданий к разделу 4

Номер варианта	Частоты для настройки SYSCLK (источник (I)-HSI или (E)-HSE)			выводов (ножек-pin)
	МГц	AHB Pre	MCO	
1	48 (I)	8	HSE	PA9
2	56 (E)	8	HSI	PA4
3	64 (I)	8	HSE	PA6
4	12 (I)	2	HSE	PB5

1. Листинг программы:

```
#include "RTE_Components.h"
#include CMSIS_device_header
#include <stdio.h>

int fputc(int c, FILE *f) {
            return (ITM_SendChar(c));
}


int main() {
      volatile uint32_t StartUpCounter = 0, HseStatus = 0;

      // check default clock frequency
      SystemCoreClockUpdate();
      ITM_SendChar('\n');
      printf("Start clk=%d Hz\n", SystemCoreClock);

      // enable HSE
      SET_BIT(RCC -> CR, RCC_CR_HSEON);

      do {
            HseStatus = RCC -> CR & RCC_CR_HSERDY;
            StartUpCounter++;
      } while ((HseStatus == 0) && (StartUpCounter != 0x5000));

      // check if hse ready
      if ((RCC->CR & RCC_CR_HSERDY) != RESET)
      {
            // set up FLASH - commands prefetch latency
```

```
            // 000: Zero wait state, if 0 < HCLK <= 24 MHz ->
FLASH_ACR_LATENCY_0
            // 001: One wait state, if 24 MHz < HCLK <= 48MHz ->
FLASH_ACR_LATENCY_1
            // 010: Two wait state, if 48 MHz < HCLK <= 72MHz ->
FLASH_ACR_LATENCY_2
            //   0: Prefetch is disabled
            //    1: Prefetch is enabled -> FLASH_ACR_PRFTBE
            FLASH -> ACR = FLASH_ACR_PRFTBE | FLASH_ACR_LATENCY_0;

            // HCLK = SYSCLK / ...
            // SYSCLK divided by 2 -> RCC_CFCR_HPRE_DIV2
            RCC->CFGR |= RCC_CFGR_HPRE_DIV2; // AHB Pre = 2 by var
            //RCC->CFGR |= RCC_CFGR_HPRE_DIV1;

            // set up PLL to 12MHz = 8 MHz (HSI) / 2 * 3

            // disable PLL for configuration
            CLEAR_BIT(RCC -> CR, RCC_CR_PLLON);

            // Bit 16 PLLSRC: -> RCC_CFGR_PLLSRC
            //   0: HSI/2 selected as PLL input clock
            //   1: HSE selected as PLL input clock
            // Bits 21:18 PLLMUL: -> RCC_CFGR_PLLMUL
            //   0111: PLI input clock x 3 -> RCC_CFGR_PLLMUL3
            // Bit 17 PLLXTPRE: -> RCC_CFGR_PLLXTPRE
            //   0: HSE clock not divided
            //   1: HSE clock divided by 2
            // PLL configuration: PLLCLK = HSI / 2 * 3 = 12 MHz
            RCC -> CFGR &= ~(RCC_CFGR_PLLSRC | RCC_CFGR_PLLXTPRE |
RCC_CFGR_PLLMULL);
            RCC -> CFGR |= (RCC_CFGR_PLLSRC_HSI_Div2 | RCC_CFGR_PLLMULL3);

            // enable PLL: PLLON
            // 0: PLL OFF;
            // 1: PLL ON
            SET_BIT(RCC -> CR, RCC_CR_PLLON) ;
            // wait for PLL READY state
            while ((RCC->CR & RCC_CR_PLLRDY) == 0) {}
            // select PLL as source for SYSCLK
            RCC->CFGR &= ~(RCC_CFGR_SW);
            RCC->CFGR |= RCC_CFGR_SW_PLL;
            // wait for setting PLL as source for SYSCLK
            while ((RCC->CFGR & RCC_CFGR_SWS) != RCC_CFGR_SWS_PLL) {}
      }
      else
      {
            // HSE is not ready
            while(1){}
      }

      SystemCoreClockUpdate() ; // check new frequency
      printf("After configuration elk=%d Hz\n", SystemCoreClock);
      // setup Main Clock Output (MCO) to HSE
      SET_BIT(RCC -> CFGR, RCC_CFGR_MCO_HSE);

      SET_BIT(RCC -> APB2ENR, RCC_APB2ENR_IOPAEN) ; //allow GPIOA work
      // Reset PA8 config
      GPIOA->CRH &= ~(GPIO_CRH_MODE8 | GPIO_CRH_CNF8) ;
      //MODE: output with max freq 50 MHz
      //CNF: Alternate function output Push-pull
      SET_BIT (GPIOA->CRH, GPIO_CRH_MODE8 | GPIO_CRH_CNF8_1);

      // enable GPIOC PB5
      SET_BIT (RCC->APB2ENR, RCC_APB2ENR_IOPBEN);
```

```
	// setup PB5
	GPIOB->CRL &= ~(GPIO_CRL_MODE5 | GPIO_CRL_CNF5);
	SET_BIT(GPIOB->CRL, GPIO_CRL_MODE5);// High speed
	while(1){
		// set 1 on PB5
		GPIOB->BSRR = GPIO_BSRR_BS5;
		// reset bit PB5
		GPIOB->BRR = GPIO_BSRR_BS5;
	}

	for(;;){}
	return 0;
	}
```

2. Отпечатки окон Watch и Debug (printf)

Watch 1

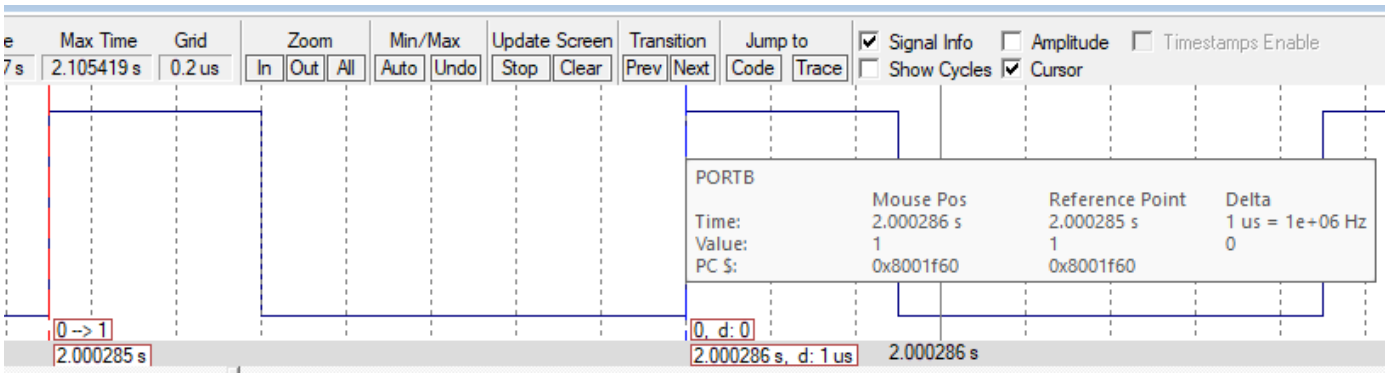
Name	Value	Type
HSI_RC	8000000	ulong
SYSCLK	12000000	ulong
OSC	8000000	ulong
SystemCoreClock	6000000	uint
HCLK	6000000	ulong
<Enter expression>		

Debug (printf) Viewer

```
Start clk=8000000 Hz
After configuration clk=6000000 Hz
```

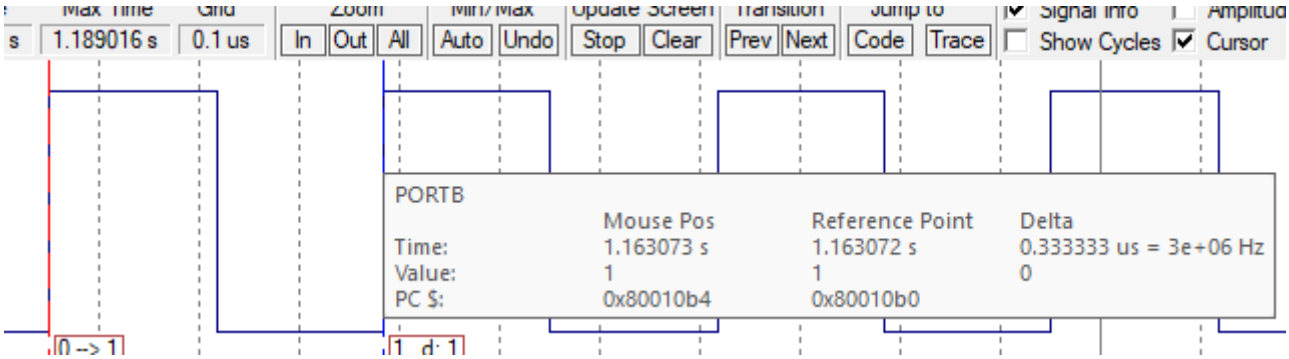
3. Эпюры сигналов

При оптимизации -O0:



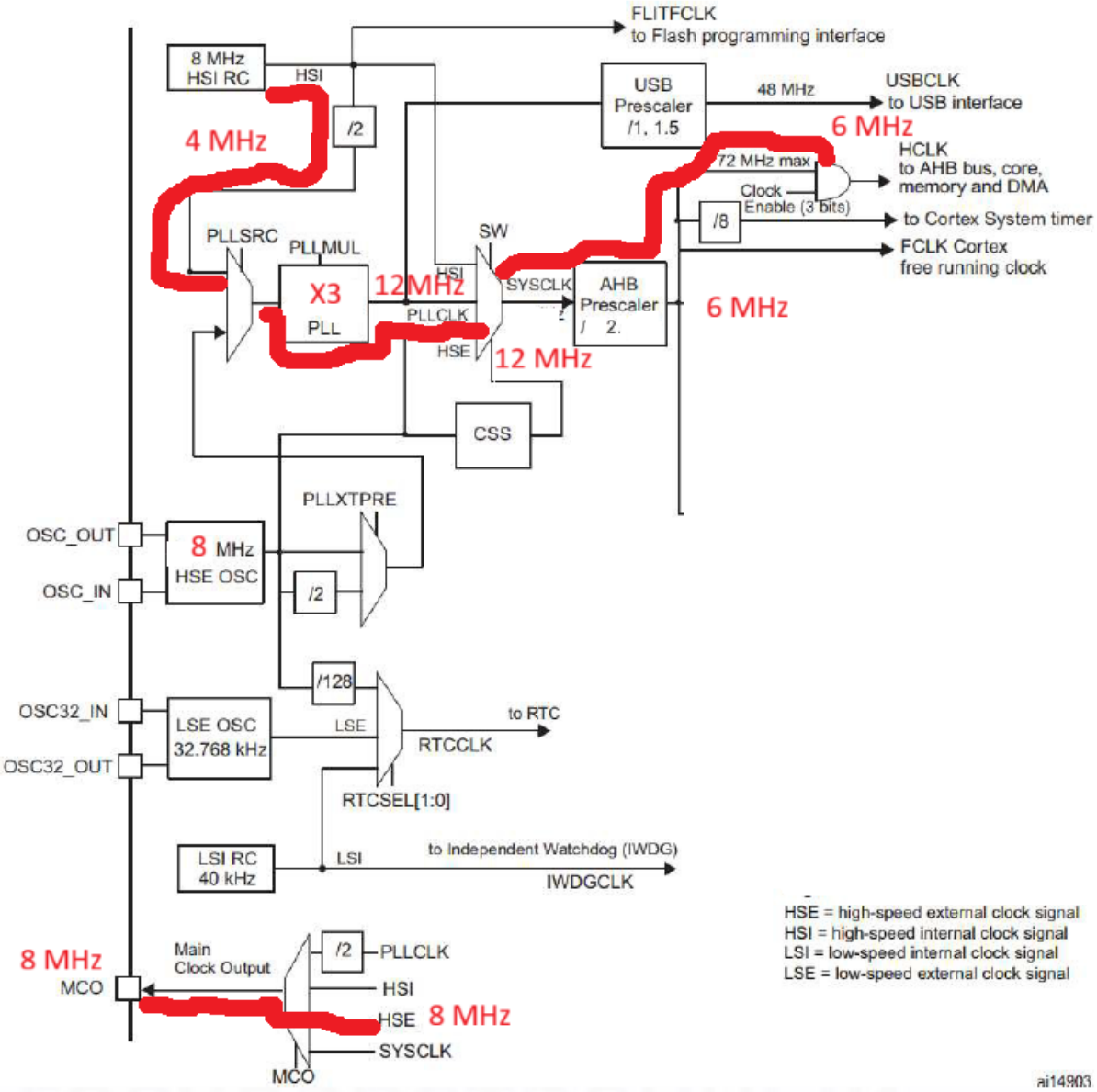
Частота – 1 МГц.

При оптимизации -O3:



Частота – 3 МГц.

4. Маршрут настройки частоты



Раздел 5.

Задание

Используя библиотеку CMSIS, написать и отладить работу программы на языке «си», в которую входит настройка внешних прерываний (EXTI) на четыре линии ПВВ A/B. согласно номеру варианта таблицы 5.1.

Этап №1. Не меняя приоритеты и группировку приоритетов (оставляем по умолчанию после сброса) проверить в режиме отладки порядок обработки:

a) одновременно установленных четырёх прерываний;

b) поступления прерываний последовательно.

Отразить в выводах выявленный порядок обработки прерываний, прерывают ли прерывания друг друга и если да, то в каком порядке.

Этап №2. Не меняя группировку приоритетов, установить приоритеты согласно варианту таблицы 5.1, исследовать изменение порядка обработки прерываний.

Этап .№3. Настроить приоритеты прерываний, используя группы и подгрупны, согласно варианту таблицы 5.1, исследовать изменение порядка обработки прерываний.

Выяснить влияние прерываний на сигнал линии PC13 управляемой системным таймером. Пояснить в выводах особенности обработки прерываний при различных настройках.

Используя библиотеку SPL, написать вторую аналогичную программу этапа 3 и отладить её работу.

Вариант 4.

Варианты индивидуальных заданий к разделу 5

Номер вари-анта	Номера линий	Количество групп и подгрупп приоритетов		Номера приоритетов до настройки группировки приоритетов и после, соответственно порядку линий	
		Групп	Подгрупп	до	после №группы(№подгруппы)
1	PA0,PB1,PA2,PB3	8	2	99,114,128,113	2(0), 1(0), 1(1), 0(1)
2	PB0,PA1,PB2,PA4	4	4	103,118,132,117	2(2), 1(1), 1(0), 0(0)
3	PA0,PB1,PA2,PB10	2	8	107,122,136,121	0(1), 1(1), 1(0), 0(0)
4	PB0,PA1,PA3,PB4	8	2	111,126,140,125	4(0), 3(1), 3(0), 2(1)

1. Листинги программы

Файл инициализации кнопок .ini:

```
KILL BUTTON 4
KILL BUTTON 3
KILL BUTTON 2
KILL BUTTON 1

OSC = 8000000;
PORTA |= 0x000A; // 1,3 to 1
PORTB |= 0x0011; //0,4 to 1
SIGNAL void toggle_B0() {
      PORTB &= ~0x01;
      swatch (0.02);
      PORTB |= 0x01;
}
SIGNAL void toggle_A1() {
      PORTA &= ~0x02;
      swatch (0.02);
      PORTA |= 0x02;
}
SIGNAL void toggle_A3() {
      PORTA &= ~0x08;
      swatch (0.02);
      PORTA |= 0x08;
}
SIGNAL void toggle_B4() {
      PORTB &= ~0x10;
      swatch (0.02);
      PORTB |= 0x10;
}

DEFINE BUTTON "Key PB0", "toggle_B0()"
DEFINE BUTTON "Key PA1", "toggle_A1()"
DEFINE BUTTON "Key PA3", "toggle_A3()"
DEFINE BUTTON "Key PB4", "toggle_B4()"
```

Листинг программы этапа 3:

```
#include "RTE_Components.h"
#include CMSIS_device_header
#include <stdio.h>

int fputc(int c, FILE *f) {
      return (ITM_SendChar(c));
}


void delay (void) {
      volatile uint32_t i=600000;// delay
      while (i > 0){
            i--;
      }
}

int main(void) {
      uint32_t priGroup = 0, PreemptPriority=0, SubPriority=0;
      SystemCoreClockUpdate();
      printf("clk=%d Hz\n", SystemCoreClock);

      // var 4: PB0, PA1, PA3, PB4 lines

      // allow GPIO A,C,B work
      RCC->APB2ENR|=RCC_APB2ENR_IOPAEN|RCC_APB2ENR_IOPCEN|RCC_APB2ENR_IOPBEN;

      //PC13 on output
```

```
	GPIOC->CRH &= ~(GPIO_CRH_MODE13 | GPIO_CRH_CNF13);
	SET_BIT(GPIOC->CRH, GPIO_CRH_MODE13);

	//PB0, PA1, PA3, PB4 Input, Pull up
	GPIOA->CRL = 0;
	SET_BIT(
		GPIOA->CRL,
		GPIO_CRL_CNF1_1|GPIO_CRL_CNF3_1);
	SET_BIT(
		GPIOA->ODR,
		GPIO_ODR_ODR1|GPIO_ODR_ODR3); //pu11 up on start

	GPIOB->CRL = 0;
	SET_BIT(
		GPIOB->CRL,
		GPIO_CRL_CNF0_1|GPIO_CRL_CNF4_1);
	SET_BIT(
		GPIOB->ODR,
		GPIO_ODR_ODR0|GPIO_ODR_ODR4); //pu11 up on start

	// alternate mode for GPIO
	SET_BIT(
		RCC->APB2ENR,
		RCC_APB2ENR_AFIOEN);

	// EXTI as external inputs in alternate functions IO
	// PB0>>EXTI0, PA1>>EXTI1, PA3>>EXTI3
	AFIO->EXTICR[0] = AFIO_EXTICR1_EXTI0_PB
		| AFIO_EXTICR1_EXTI1_PA
		| AFIO_EXTICR1_EXTI3_PA;
	// PB4>>EXTI4
	AFIO->EXTICR[1] = AFIO_EXTICR2_EXTI4_PB;


	// prioritetes by var: 111, 126, 140, 125
	// 4(0) 3(1) 3(0) 2(1)

	//NVIC_SetPriorityGrouping(NVIC_PriorityGroup_3);
	// by var 4: 1 bit for sub-priority(1-0), 3 for priority (0-7)
	NVIC_SetPriorityGrouping(4);
	priGroup = NVIC_GetPriorityGrouping();
	printf("Priority Grouping=%d\r\n", priGroup);

	//NVIC_SetPriority(EXTI0_IRQn, 111);
	NVIC_SetPriority(
		EXTI0_IRQn,
		NVIC_EncodePriority(priGroup, 4, 0));

	NVIC_DecodePriority(
		NVIC_GetPriority(EXTI0_IRQn),
		priGroup,
		&PreemptPriority,
		&SubPriority);
	printf(
		"EXTI0 Preempt Priority=%d \tSubPriority=%d\r\n",
		PreemptPriority,
		SubPriority) ;

	//NVIC_SetPriority(EXTI1_IRQn, 126) ;
	NVIC_SetPriority(
		EXTI1_IRQn,
		NVIC_EncodePriority(priGroup, 3, 1));

	NVIC_DecodePriority(
		NVIC_GetPriority(EXTI1_IRQn),
```

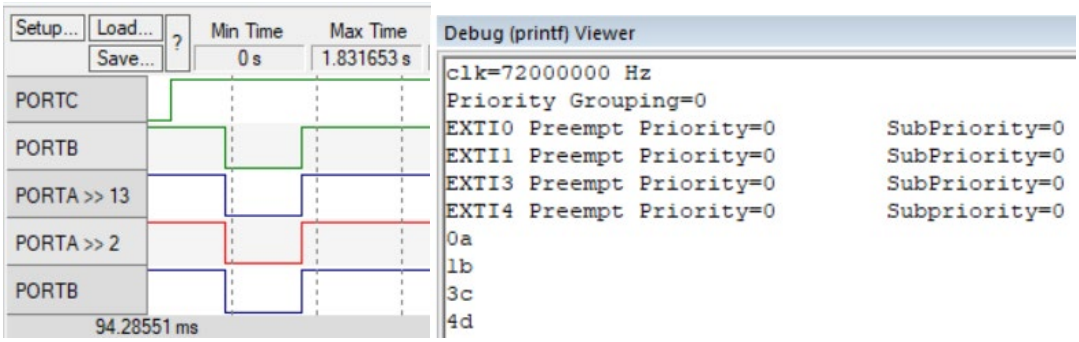
```
            priGroup,
            &PreemptPriority,
            &SubPriority) ;
      printf(
            "EXTI1 Preempt Priority=%d \tSubPriority=%d\r\n",
            PreemptPriority,
            SubPriority);

      //NVIC_SetPriority(EXTI3_IRQn, 140);
      NVIC_SetPriority(
            EXTI3_IRQn,
            NVIC_EncodePriority(priGroup, 3, 0));

      NVIC_DecodePriority(
            NVIC_GetPriority(EXTI3_IRQn),
            priGroup,
            &PreemptPriority,
            &SubPriority);
      printf(
            "EXTI3 Preempt Priority=%d \tSubPriority=%d\r\n",
            PreemptPriority,
            SubPriority);

      //NVIC_SetPriority(EXTI4_IRQn, 125);
      NVIC_SetPriority(
            EXTI4_IRQn,
            NVIC_EncodePriority(priGroup, 2, 1));

      NVIC_DecodePriority(
            NVIC_GetPriority(EXTI4_IRQn),
            priGroup,
            &PreemptPriority,
            &SubPriority);
      printf(
            "EXTI4 Preempt Priority=%d \tSubpriority=%d\r\n",
            PreemptPriority,
            SubPriority) ;

      // Falling trigger event configuration
      SET_BIT(
            EXTI->FTSR,
            EXTI_FTSR_TR0|EXTI_FTSR_TR1|
            EXTI_FTSR_TR3|EXTI_FTSR_TR4);

      // allow interruptions for external lines
      SET_BIT(
            EXTI->IMR,
            EXTI_IMR_MR0|EXTI_IMR_MR1|
            EXTI_IMR_MR3|EXTI_IMR_MR4);

      NVIC_EnableIRQ(EXTI0_IRQn);
      NVIC_EnableIRQ(EXTI1_IRQn);
      NVIC_EnableIRQ(EXTI3_IRQn);
      NVIC_EnableIRQ(EXTI4_IRQn);
      SysTick_Config(0x6DDD00);
      while (1){}
}

// switch PC13 every 100ms
void SysTick_Handler(void) {
      GPIOC->ODR^=1<<13;
}

void EXTI0_IRQHandler (void){
      EXTI->PR = EXTI_PR_PR0;
```

```
	ITM_SendChar('0');
	delay();
	ITM_SendChar('a');
	ITM_SendChar('\n');
}

void EXTI1_IRQHandler (void) {
	EXTI->PR = EXTI_PR_PR1;
	ITM_SendChar('1');
	delay();
	ITM_SendChar('b');
	ITM_SendChar('\n');
}

void EXTI3_IRQHandler (void) {
	EXTI->PR = EXTI_PR_PR3;
	ITM_SendChar('3');
	delay();
	ITM_SendChar('c');
	ITM_SendChar('\n');
}

void EXTI4_IRQHandler(void) {
	EXTI->PR = EXTI_PR_PR4;
	ITM_SendChar('4');
	delay();
	ITM_SendChar('d');
	ITM_SendChar('\n');
}
```

2. Анализ поведения системы прерываний.

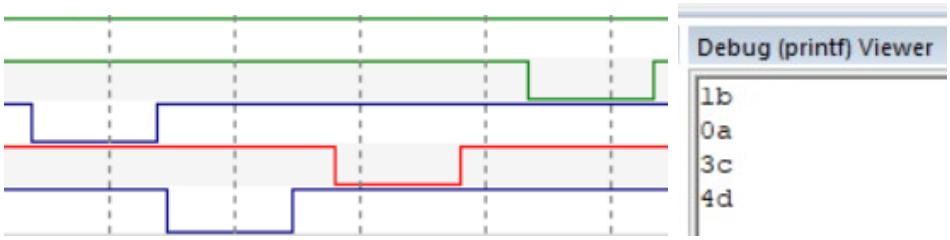
Рассмотрим поведение системы при установке приоритетов по умолчанию после сброса.

Одновременное возникновение прерываний:



Прерывания выполняются в соответствии со своими порядковыми номерами (Idx в NVIC). Как видно из вывода, у всех настроенных прерываний наивысший приоритет по умолчанию (0).

Последовательное возникновение прерываний:

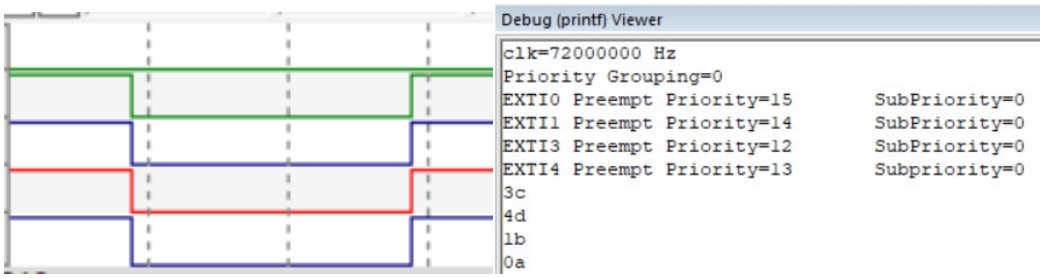


Аналогично, прерывания выполняются в соответствии со своими порядковыми номерами. Первым пришло прерывание EXTI1, оно сразу начало выполняться. Остальные прерывания возникли ещё до завершения прерывания EXTI1, поэтому они выполнились в соответствии со своими порядковыми номерами: EXTI0, EXTI3, EXTI4.

Перейдем к настройке приоритетов прерываний без изменения группировки приоритетов.

Согласно варианту приоритеты составляют 111, 126, 140, 125 для прерываний EXTI0, EXTI1, EXTI3, EXTI4 соответственно.

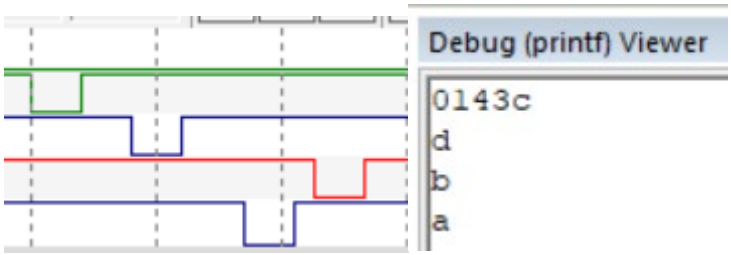
Одновременное возникновение прерываний:



Значения приоритетов стали 15,14,12,13. Для настройки приоритетов в МК STM32F103C8T6 выделено 4 бита (0-15), хотя архитектура ядра Cortex-M3 поддерживает до 256 уровней приоритетов. Поэтому в результате применения приоритетов, заданных по варианту, применились значения из младших 4 битов значений приоритетов.

При одновременном возникновении прерываний они выполняются в соответствии со своими приоритетами. Первым выполняется прерывание EXTI3, имеющее приоритет 12, затем прерывание EXTI4 с приоритетом 13 и т.д.

Последовательное возникновение прерываний:

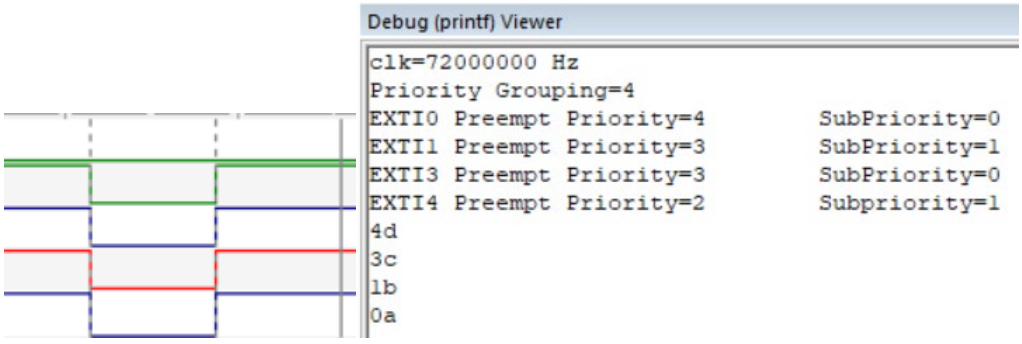


При последовательном выполнении прерываний одно прерывание может быть вытеснено другим, если у нового прерывания более высокий приоритет. В данном случае первым возникло прерывание по линии 0, но, прежде чем оно завершилось, возникло прерывание по линии 1, которое начало свое выполнение. Его в свою очередь вытеснило прерывание по линии 4. Последним возникло прерывание по линии 3, которое является самым приоритетным. После окончания прерывания по линии 3 все приостановленные прерывания по очереди завершили свою работу в соответствии со своими приоритетами: наименее приоритетное прерывание по линии 0 завершилось последним, хотя было вызвано первым.

Настроим группировку приоритетов. Согласно варианту должны быть настроены 8 групп и 2 подгруппы. Приоритеты для прерываний: 4(0) 3(1) 3(0) 2(1).

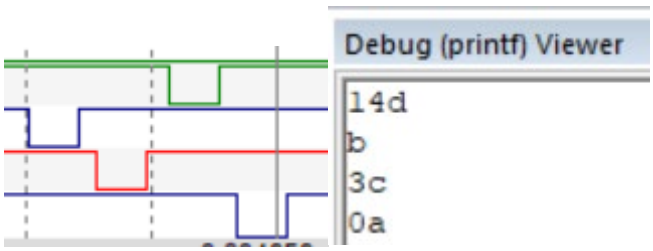
Для задания 8 групп необходимо три бита. Для задания 2 подгрупп нужен один бит. Для того, чтобы из имеющихся в нашем МК 4 битов 3 использовалось под группу, а 1 под подгруппу, необходимо задать группировку 4.

Одновременное возникновение прерываний:



При задании групп и подгрупп разрешение одновременно возникающих прерываний выполняется сначала по значению группы, а затем по значению подгруппы. Прерывание по линии 3 выполнилось раньше, чем по линии 1, несмотря на одинаковую группу: подгруппа прерывания по линии 3 выше, чем по линии 1.

Последовательное возникновение прерываний:



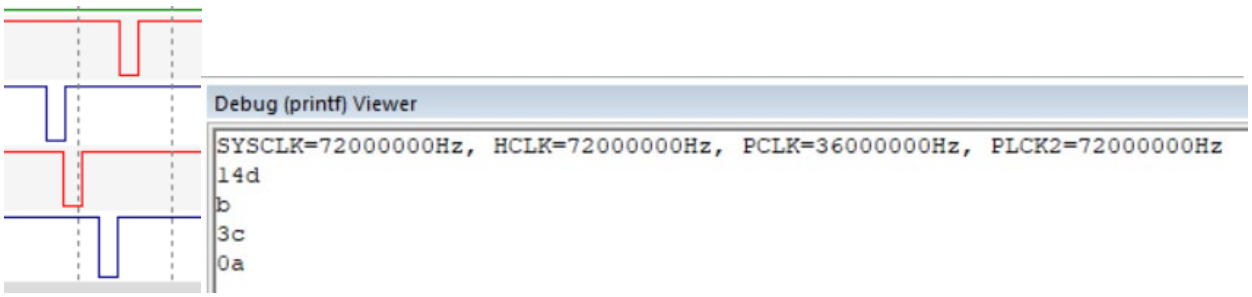
В данном случае сначала возникло прерывание по линии 1, затем по 3, 0 и 4. Несмотря на то, что прерывание по линии 3 имеет более высокий подприоритет, оно не вытеснило прерывание по линии 1, т.к. у них одна группа. Подприоритет используется для разрешения порядка одновременно возникающих прерываний, но он не позволяет прерываниям вытеснять другие прерывания той же группы.

Прерывание по линии 4 вытеснило прерывание по линии 1, т.к. у него больше приоритет по группе. После окончания его работы прерывание по линии 1 продолжило работу (а не произошёл запуск прерывания по линии 3). Только после его окончания отработало прерывание по линии 3, а затем отработало наименее приоритетное прерывание по линии 0.

Таким образом, группа прерываний влияет на одновременно возникающие прерывания и на вытеснение одних прерываний другими при последовательном возникновении. Подгруппа же используется только для разрешения одновременно возникающих прерываний.

Написана программа для выполнения этапа 3 с использованием стандартной библиотеки периферии (SPL).

Пример выполнения для этапа 3:



Листинг программы:

```
#include "RTE_Components.h"
#include CMSIS_device_header
#include <stdio.h>

int fputc(int c, FILE *f) {
      return (ITM_SendChar(c));
```

```
}

void delay (void) {
	volatile uint32_t i=600000;// delay
	while (i > 0){
		i--;
	}
}

void EXTI_Config(void);

EXTI_InitTypeDef   EXTI_InitStructure;
GPIO_InitTypeDef   GPIO_InitStructure;
NVIC_InitTypeDef   NVIC_InitStructure;

int main()
{
	uint32_t cnt = 0;
	GPIO_InitTypeDef GPIO_InitStructure;
	RCC_ClocksTypeDef RCC_ClockFreq;
	RCC_GetClocksFreq(&RCC_ClockFreq);
	printf(
		"SYSCLK=%dHz, HCLK=%dHz, PCLK=%dHz, PLCK2=%dHz\n",
		RCC_ClockFreq.SYSCLK_Frequency,
		RCC_ClockFreq.HCLK_Frequency,
		RCC_ClockFreq.PCLK1_Frequency,
		RCC_ClockFreq.PCLK2_Frequency);

	if (SysTick_Config(0x6DDD00))
  {
    /* Capture error */
    while (1);
  }


	RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);

	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13;
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
	GPIO_Init(GPIOC, &GPIO_InitStructure);


	RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);

	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
	GPIO_InitStructure.GPIO_Speed = 0;

	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1 | GPIO_Pin_3;
	GPIO_Init(GPIOA, &GPIO_InitStructure);

	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_4;
	GPIO_Init(GPIOB, &GPIO_InitStructure);

	/* Enable AFIO clock */
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);

	EXTI_Config();

	while(1){}
	return 0;
}
```

```
// configuring EXTI
void EXTI_Config(void)
{
  /* Connect EXTI Lines to pins */
  GPIO_EXTILineConfig(GPIO_PortSourceGPIOB, GPIO_PinSource0);
      GPIO_EXTILineConfig(GPIO_PortSourceGPIOA, GPIO_PinSource1);
      GPIO_EXTILineConfig(GPIO_PortSourceGPIOA, GPIO_PinSource3);
      GPIO_EXTILineConfig(GPIO_PortSourceGPIOB, GPIO_PinSource4);

  /* Configure EXTI lines */
  EXTI_InitStructure.EXTI_Line = EXTI_Line0 | EXTI_Line1 | EXTI_Line3 |
EXTI_Line4;
  EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
  EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
  EXTI_InitStructure.EXTI_LineCmd = ENABLE;
  EXTI_Init(&EXTI_InitStructure);

      NVIC_PriorityGroupConfig(NVIC_PriorityGroup_3);


  /* Enable and set EXTI0 Interrupt to the lowest priority */
      NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;

  NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 4;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
  NVIC_Init(&NVIC_InitStructure);

      NVIC_InitStructure.NVIC_IRQChannel = EXTI1_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 3;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
  NVIC_Init(&NVIC_InitStructure);

      NVIC_InitStructure.NVIC_IRQChannel = EXTI3_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 3;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
  NVIC_Init(&NVIC_InitStructure);

      NVIC_InitStructure.NVIC_IRQChannel = EXTI4_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 2;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
  NVIC_Init(&NVIC_InitStructure);
}

// switch PC13 every 100ms
void SysTick_Handler(void) {
      GPIOC->ODR^=1<<13;
}

void EXTI0_IRQHandler (void){
      EXTI->PR = EXTI_PR_PR0;
      ITM_SendChar('0');
      delay();
      ITM_SendChar('a');
      ITM_SendChar('\n');
}

void EXTI1_IRQHandler (void) {
      EXTI->PR = EXTI_PR_PR1;
      ITM_SendChar('1');
      delay();
      ITM_SendChar('b');
      ITM_SendChar('\n');
}
```

```
void EXTI3_IRQHandler (void) {
	EXTI->PR = EXTI_PR_PR3;
	ITM_SendChar('3');
	delay();
	ITM_SendChar('c');
	ITM_SendChar('\n');
}

void EXTI4_IRQHandler(void) {
	EXTI->PR = EXTI_PR_PR4;
	ITM_SendChar('4');
	delay();
	ITM_SendChar('d');
	ITM_SendChar('\n');
}
```

Раздел 6

Задание

В соответствии с вариантом нужно написать на языке «си» программу генерации прямоугольных импульсов на двух линиях в/в одновременно с заданной частотой.

Один из таймеров настроить на счёт вверх (LL_TIM_COUNTERMODE_UP). Второй таймер на счёт вниз (LL_TIM_COUNTERMODE_DOWN). Сравнить эпюры полученных TIMx->CNT сигналов.

Вариант 4:

Таблица 6.1

Варианты индивидуальных заданий к разделу 6

Номер варианта	Частота SYSCLK, МГц	Генератор №1			Генератор №2		
		Таймер	Линия	Частота, Гц	Таймер	Линия	Частота, Гц
1	32	TIM2	PA9	3952	TIM3	PB8	208
2	36	TIM2	PA4	4180	TIM4	PA15	220
3	40	TIM3	PA6	4427	TIM2	PA10	233
4	44	TIM1	PB5	4693	TIM4	PA11	247

Листинг программы:

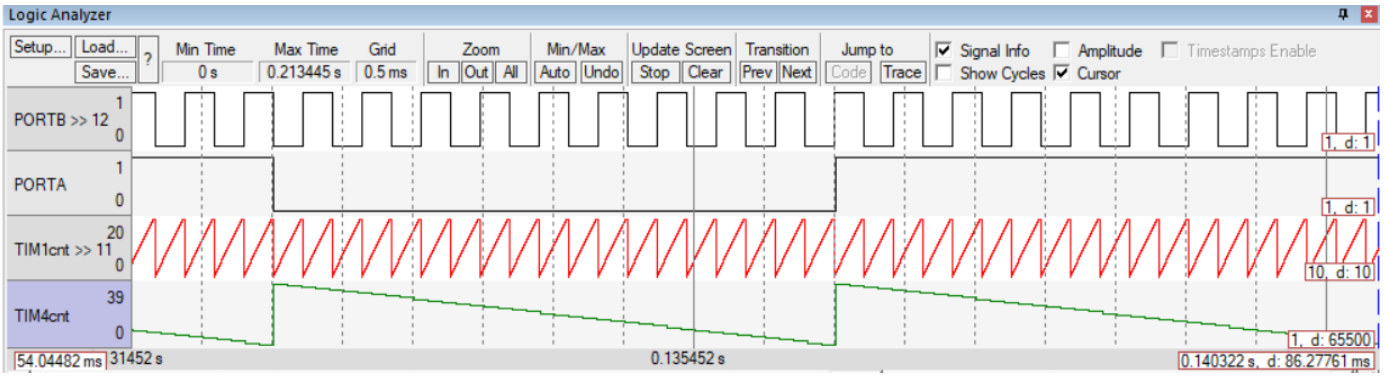
```
#include "stm32f1xx_ll_bus.h"
#include "stm32f1xx_ll_rcc.h"
#include "stm32f1xx_ll_system.h"
#include "stm32f1xx_ll_utils.h"
#include "stm32f1xx_ll_gpio.h"
#include "stm32f1xx_ll_exti.h"
#include "stm32f1xx_ll_tim.h"
#include "stm32f1xx_ll_cortex.h"
#include <stdio.h>
```

```
int fputc(int c, FILE *f) {
        return (ITM_SendChar(c));
}

uint16_t TIM4cnt=0; // current counter TIM4
uint16_t TIM1cnt=0; // current counter TIM1

// TIM1 PB5 4693
void setupTIM1() {
        // setup timer 1 (TIM1)
        LL_APB2_GRP1_EnableClock(LL_APB2_GRP1_PERIPH_TIM1);//enable timer
        LL_TIM_SetCounterMode(TIM1, LL_TIM_COUNTERMODE_UP);//count up

        //44,000,000 / (938 * 10) ~ 4691Hz
        uint16_t PSC = 9;
        uint16_t ARR = 937;

        // set prescaler
        //TIM1CLK = PCLK2x1=44MHz
        LL_TIM_SetPrescaler(
                TIM1,
                PSC);

        printf(
                "TIM1_PSC=%d\n",
                PSC);

        LL_TIM_SetAutoReload(TIM1, ARR);
        printf("TIM1_ARR=%d\n", ARR);

        // allow timer 4 interrupt on update
        LL_TIM_EnableIT_UPDATE(TIM1);
        NVIC_SetPriority(TIM1_UP_IRQn, 0);
        NVIC_EnableIRQ(TIM1_UP_IRQn); // allow TIM1 IRQ
        LL_TIM_EnableCounter(TIM1); // include counter timer
        LL_TIM_GenerateEvent_UPDATE(TIM1); // call update programmatically
}

// TIM4 PA11 246
void setupTIM4() {
        // setup timer 4 (TIM4)
        LL_APB1_GRP1_EnableClock(LL_APB1_GRP1_PERIPH_TIM4);//enable timer
        LL_TIM_SetCounterMode(TIM4, LL_TIM_COUNTERMODE_DOWN);//count down

        // 44,000,000/(423*422) ~ 246.72 Hz
        uint16_t PSC = 422;
        uint16_t ARR = 421;

        // set prescaler
        //TIM4CLK = PCLK1x2=44MHz
        LL_TIM_SetPrescaler(
                TIM4,
                PSC);

        printf(
                "TIM4_PSC=%d\n",
                PSC);

        LL_TIM_SetAutoReload(TIM4, ARR);
        printf("TIM4_ARR=%d\n", ARR);

        // allow timer 4 interrupt on update
```

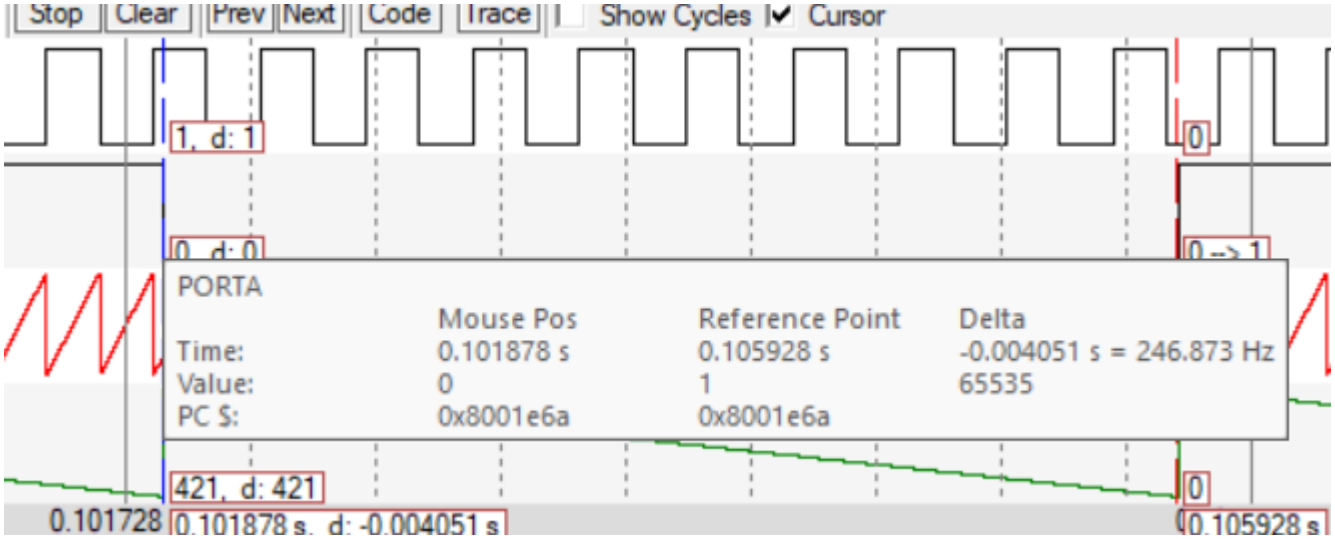
```
        LL_TIM_EnableIT_UPDATE(TIM4);
        NVIC_SetPriority(TIM4_IRQn, 0);
        NVIC_EnableIRQ(TIM4_IRQn); // allow TIM2 IRQ
        LL_TIM_EnableCounter(TIM4); // include counter timer
        LL_TIM_GenerateEvent_UPDATE(TIM4); // call update programmatically
}

int main (void){
        printf ("clk=%d Hz\n", SystemCoreClock);

        // config FLASH - 44 MHz SYSCLK -> one wait state
        LL_FLASH_SetLatency(LL_FLASH_LATENCY_1);

        // enable HSI
        LL_RCC_HSI_Enable();
        while (LL_RCC_HSI_IsReady() != 1){}

        // config PLL: 8 / 2 * 11 == 44 MHz
        LL_RCC_PLL_ConfigDomain_SYS(
                LL_RCC_PLLSOURCE_HSI_DIV_2,
                LL_RCC_PLL_MUL_11);
        LL_RCC_PLL_Enable();
        while (LL_RCC_PLL_IsReady() != 1){}

        // config SYSCLK source from PLL
        LL_RCC_SetAHBPrescaler(LL_RCC_SYSCLK_DIV_1);
        LL_RCC_SetSysClkSource(LL_RCC_SYS_CLKSOURCE_PLL);
        while(LL_RCC_GetSysClkSource() != LL_RCC_SYS_CLKSOURCE_STATUS_PLL){};

        LL_RCC_SetAPB1Prescaler(LL_RCC_APB1_DIV_2);
        LL_RCC_SetAPB2Prescaler(LL_RCC_APB2_DIV_1);
        SystemCoreClockUpdate();
        printf("clk=%d Hz\n", SystemCoreClock);

        // variant 4:
        // 1. TIM1 PB5 4693
        // 2. TIM4 PA11 246

        // setup PB5 and PA11 output
        LL_APB2_GRP1_EnableClock(
                LL_APB2_GRP1_PERIPH_GPIOB
                | LL_APB2_GRP1_PERIPH_GPIOA);

        LL_GPIO_SetPinMode(GPIOB, LL_GPIO_PIN_5, LL_GPIO_MODE_OUTPUT);
        LL_GPIO_SetPinMode(GPIOA, LL_GPIO_PIN_11, LL_GPIO_MODE_OUTPUT);

        // get and print clocks freq
        LL_RCC_ClocksTypeDef clocks;
        LL_RCC_GetSystemClocksFreq(&clocks);
        printf(
                "PCLK1=%d\nPCLK2=%d\n",
                clocks.PCLK1_Frequency,
                clocks.PCLK2_Frequency);

        // setup TIM1
        setupTIM1();

        // setup TIM4
        setupTIM4();

        while(1){
                TIM4cnt=TIM4->CNT;
                TIM1cnt=TIM1->CNT;
```

```
        }
}

void TIM4_IRQHandler(void) {
        if(LL_TIM_IsActiveFlag_UPDATE(TIM4) == 1) {
                        LL_TIM_ClearFlag_UPDATE(TIM4);
                        LL_GPIO_TogglePin(GPIOA, LL_GPIO_PIN_11);
        }
}

void TIM1_UP_IRQHandler(void) {
        if(LL_TIM_IsActiveFlag_UPDATE(TIM1) == 1) {
                        LL_TIM_ClearFlag_UPDATE(TIM1);
                        LL_GPIO_TogglePin(GPIOB, LL_GPIO_PIN_5);
        }
}
```

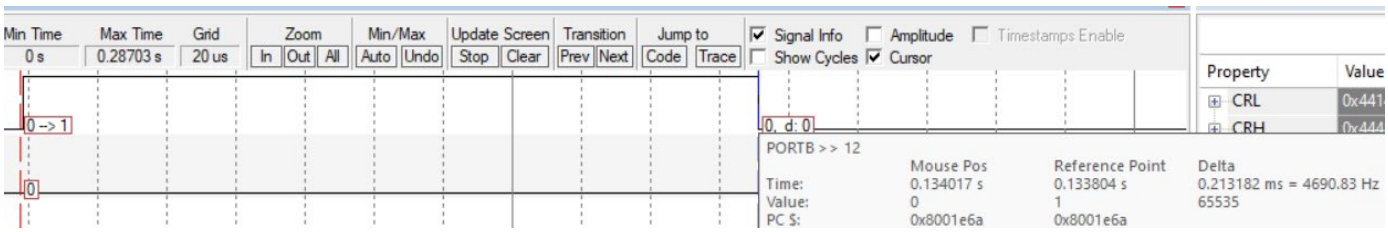
Снимки эпюр логического анализатора:



Измерения частоты для линии PA11:



Измерения частоты для линии PB5:



Погрешности полученных значений частот находятся в пределах менее 1%.

Задача:

Рассчитать коэффициенты настройки подсистемы тактирования МК STM32F103C8 и таймера TIMx для настройки прерывания по обновлению таймера один раз в 3+2x4=11 минут.

Рассчитать значения SYSCLK, PCLKx (PLL/HSE/HSI, значения коэфф. PLLMUL, AHB_Pre), частоту TIMxCLK, рассчитать значения регистров TIMx_PSC и TIMxARR.

Решение учебной задачи:

Воспользуемся упрощенной формулой частоты события обновления таймера (методические указания, формула 6.1):

$$f_{UE} = \frac{TIM\_CLK}{(PSC + 1)*(ARR + 1)}$$

Из RM0008, Fig. 100 получаем формулы:

$$CK\_CNT = \frac{TIM\_CLK}{PSC + 1}, \qquad f_{UE} = \frac{CK\_CNT}{ARR + 1}$$

Необходимо достичь частоты 1/660 Гц (один раз в 660 секунд, т.е. раз в 11 минут).

Рассчитаем значение TIMxCLK, которое нельзя превышать для возможности достижения необходимого интервала срабатывания таймера:

TIMx_CLK = 1/660 * (65535+1) * (65535+1) = 6507526,20(60)

Частота TIMxCLK не может превышать 6507526 Hz.

Необходимая частота может быть достигнута следующим образом:

1. SYSCLK = 72MHz = (PLLCLK=8MHz(HSE))x(PLLMUL=9)

2. HCLK = SYSCLK/16=4.5MHz (AHB_Pre=16)
3. PCLKx = 4.5/2 = 2.25MHz (APBx_Pre = 2)
4. TIMxCLK = 2.25 * 2 = 4.5 MHz < 6507526 Hz

Далее необходимо рассчитать значения TIMx_PSC и TIMx_ARR для полученной частоты TIMxCLK.

Рассчитаем максимально возможное значение CK_CNT.

CK_CNT = 1/660 *(65535+1) = 99,29(69).

Возьмем частоту 90, т.к. она без остатка делит 4,5MHz.

Посчитаем коэффициент ARR = (CK_CNT/Fue) – 1 = 90/(1/660) – 1= 59399, т.е. TIMx_ARR = 59399.

Посчитаем значение предделителя PSC:

PSC = TIMx_CLK / CK_CNT – 1 = 4500000/90 – 1 = 49999, т.е. TIMx_PSC=49999.

Проверка расчетов:

$$f_{UE} = \frac{TIMx\_CLK}{(PSC + 1)\cdot(ARR + 1)}$$ = 4500000/(49999+1)/(59399+1)=0,00(15)=1/660Hz, т.е. тактирование таймера раз в 660 секунд (11 минут).

Ответ: тактирование от PLL, источник HSE/1, PLLMUL = 9, SYSCLK=PLLCLK=72Mhz, AHB_Pre=16, HCLK = 4,5 MHz, APBx_Pre=2, TIMxCLK=4,5MHz, TIMx_PSC=49999, TIMx_ARR = 59399.