МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра ВТ

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Операционные системы реального времени»

Тема: ПРЕРЫВАНИЯ В ОС FreeRTOS

Вариант 1

Студенты гр. 6492 _____ Мурашко А. С.
Огурецкий Д. В.

Преподаватель _____ Гречухин М. Н

Санкт-Петербург
2020

Цель: изучить организацию прерываний в ОС FreeRTOS

Задание:

В рамках данной работы предлагается создать две задачи. Одна из них будет управлять периферией визуализации, вторая – обрабатывать прерывание от внешнего источника (тактовая кнопка на плате) и передавать через очередь команду первой задаче о переключении варианта визуализации.

1. Открыть IDE Keil MDK-ARM, создать новый проект по алгоритму из работы 1. В файле конфигурации FreeRTOSConfig.h необходимо разрешить использовать нужные в работе функции (с помощью директив вида #define INCLUDE_vTaskDelay 1).
2. Создать бинарный семафор, необходимый для синхронизации задачи IST с ISR. Создать очередь для обмена данными между задачами.
3. Создать две функции-задачи. Одна из них будет отвечать за визуализацию на выбранной периферии по указанным согласно 2 режимам: мигание диодом с частотой 2 и 10 Гц. Вторая будет являться обработчиком прерываний (IST).
4. Настроить систему прерываний используемого МК.
5. Скомпилировать проект. Загрузить его на плату, наблюдать работу с подключенной периферией. Так как отсутствует периферия, нажатие кнопки производится вручную путем изменения состояния порта.

Порядок работы:

Для организации прерывания используется гибридная многозадачность во FreeRTOS. Гибридная многозадачность сочетает в себе автоматический вызов планировщика каждый квант времени, а также возможность принудительного, явного вызова планировщика. Полезной гибридная многозадачность может оказаться, когда необходимо сократить время реакции системы на прерывание. В этом случае в конце тела обработчика прерывания производят вызов планировщика, что приводит к переключению на задачу, ожидающую наступления этого прерывания.

Какого-либо специального действия для включения режима гибридной многозадачности не существует. Достаточно разрешить вызов планировщика каждый квант времени (макроопределение configUSE_PREEMPTION в файле FreeRTOSConfig.h должно быть равным 1) и в явном виде вызывать планировщик в обработчиках прерываний с помощью API-функции portYIELD_ FROM_ISR().

Изучив теорию 3 лабораторной работы было определено что такое очереди сообщений и как их организовывать. Создание очереди производится с помощью xQueueCreate(). Так как имеется всего 2 варианта визуализации, то достаточно очереди длинной в 1 сообщение длиной в 1 байт (то есть char). Запись в очередь можно реализовать с помощью одной из функций

xQueueSendToFront() и xQueueSendToBack(), т.к. длина очереди 1 и неважно в конец или в начало помещать сообщение. Для чтения из очереди используется функция xQueueReceive(), т.к. у нас длина очереди 1 и необходимо удалять считанное значение из очереди для её освобождения.

Т.к. clearINTFlag(EXTI0_IRQn) не поддерживается компилятором C99 , то используется прямая адресация.
Программа Keil не позволяет проводить симуляцию входных портов, их изменение попросту заблокировано, поэтому отладку выполнить нельзя.

Код

```
1  #include "stm32f4xx.h"                  // Device header
2  #include "FreeRTOSConfig.h"             // ARM.FreeRTOS::RTOS:Config
3  #include "FreeRTOS.h"                   // ARM.FreeRTOS::RTOS:Core
4  #include "task.h"                       // ARM.FreeRTOS::RTOS:Core
5  #include "semphr.h"                     // ARM.FreeRTOS::RTOS:Core
6
7  xQueueHandle xQueue; //global queue
8  SemaphoreHandle_t xSemaphore; // initializing semaphore's handle
9
10 void  EXTI10_IRQHandler()
11 {
12     BaseType_t needCS = pdFALSE;
13                         /* Clear interrupt flag */
14                 EXTI->PR &= ~(EXTI_PR_PR0);//This bit is set when the se-
   lected edge event arrives on the external interrupt line.
15     xSemaphoreGiveFromISR(xSemaphore, &needCS);
16     portYIELD_FROM_ISR(needCS);
17 }
18
19
20 void Blinking(void* xSemaphore)
21 {
22         int n ;
23         #define f2HZ 0
24         #define f10HZ 1
25         char flag_mode = f2HZ; //
26
27         while(1)
28         {
29                 xQueueReceive(xQueue,&flag_mode,pdMS_TO_TICKS(0));
30                 if(flag_mode==f2HZ)
   {n=pdMS_TO_TICKS(250);}
31                 else if(flag_mode==f10HZ){n=pdMS_TO_TICKS(50);}
32                 GPIOA -> ODR ^= GPIO_ODR_ODR_5; // switching LED
33                 vTaskDelay(n);
34         }
35 }
36
37 void change_mode(void* xSemaphore)
38 {
39           #define fHz 0
40                 char flag_mode = 0;
41                 while(1)
42                 {
```

```
43                          if(xSemaphoreTake(xSemaphore, 0))
44                          {
45                                  flag_mode ^= (1 << fHz); //change mode
46                                  xQueueSendTo-
47 Front(xQueue,&flag_mode,pdMS_TO_TICKS(0));
48                          }
49                          vTaskDelay(pdMS_TO_TICKS(1000));
50                  }
51 }
52
53 int main(void)
54 {
55         RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN; // turning on GPIOA
56         GPIOA->MODER |= GPIO_MODER_MODER5_0; // setting A5 to output
57         GPIOA->MODER |= GPIO_MODER_MODER5_0; // setting A5 to output
58         //GPIOA->MODER |= GPIO_MODER_MODER0_1; //  A0 default input
59         RCC->APB2ENR |=RCC_APB2ENR_SYSCFGEN ; //System configuration con-
   troller clock enable
60         SYSCFG->EXTICR[0] |= SYSCFG_EXTICR1_EXTI0_PA ;//These bit 0 are
   written by software to select the source input for the EXTIx external in-
   terrupt.
61         EXTI->IMR |= EXTI_IMR_MR0; // Interrupt request from line 0 is not
   masked
62         EXTI->RTSR |= EXTI_RTSR_TR0 ;//  Rising trigger enabled (for Event
   and Interrupt) for input line
63         NVIC_EnableIRQ(EXTI0_IRQn); //Enables a Device specific interrupt
   number in the NVIC interrupt controller
64         __enable_irq();
65
66         //create xQueue
67         xQueue = xQueueCreate(1, sizeof(char));
68         if(xQueue == NULL) return 1;   //memory empty
69
70         xSemaphore = xSemaphoreCreateBinary();
71         if(xSemaphore == NULL) return 1; //memory empty
72
73         xTaskCreate(Blinking, "Task1", configMINIMAL_STACK_SIZE, xSema-
   phore, 2, NULL);
74         xTaskCreate(change_mode, "Task2", configMINIMAL_STACK_SIZE,xSema-
   phore, 3, NULL);
75         vTaskStartScheduler();
76
77         while(1)
78         {
79         }
80 }
```

Вывод: в ходе работы было изучено использование прерываний для обработки внешнего события: нажатия кнопки. Также освоена передача значений между задачами через очередь.