Санкт-Петербургский Политехнический Университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Аппаратные платформы встраиваемых систем

Отчет по лабораторной работе №3

На тему **«Система прерываний МК. Использование АЦП»**

Работу выполнили:

Студенты гр. 13541/1 Онищенко Д.И.

Шаменов А.А.

Преподаватель: Васильев А.Е.

Санкт – Петербург

2019

Содержание

Цель работы:

Получение навыков работы с прерываниями МК STM32, а так же изучение работы аналогово-цифрового преобразователя.

Подготовка к работе:

1. Подготовить проект в IARWE, согласно документу IAR Project for IAR SK Board
2. Ознакомиться со схемой платы IAR SK (STM32F407ZG-board-schematic.pdf)
3. Ознакомиться с документацией МК STM32F407 (STM32F4xx_RM.pdf)

Теоретическая информация:

Система прерываний МК STM32

Прерывание - это событие, как правило, связанное с каким-либо блоком периферии микроконтроллера STM32. Событий, которые могут породить прерывание может быть множество. Например, если речь о таком блоке периферии как таймер, то там могут быть такие события: переполнение счетчика, превышение счетчиком порогового значения, и т.д.

Использование прерываний позволит нашей программе мгновенно реагировать на подобные события. Сам термин прерывание говорит о том, что что-то должно прерваться и в нашем случае прервется выполнение основного кода вашей программы, и управление будет передано некоторой другой функции, которая называется обработчиком прерывания.

Стоит отметить, что от одного и того же периферийного блока (а в некоторых случаях и от разных), возникает одно и тоже прерывание. Следовательно, вызывается один и тот же обработчик прерывания, чтобы

правильно среагировать на него, необходимо проверять флаги состояния такого прерывания.

Что же следует учитывать при работе с прерываниями МК STM32:

1. Разрешение работы тех или иных прерываний независимо включается или выключаются.
2. Прерывания имеют приоритет.
3. Прерывания могут быть вызваны программным путем.
4. Если для прерывания нет обработчика, а оно возникло, то будет вызван обработчик по умолчанию.

Все возможные прерывания контроллера stm32f4xx описаны в файле stm32fxx.h библиотеки CMSIS в структуре IRQn.

Итак, для разрешения работы прерывания от конкретного перефирийного блока, необходимо для начала подключить файлы библиотеки SPL для работы с NVIC (Nested Vectored Interrupt Controller – контроллер прерываний) (файл misc.h). Как и всегда, для конфигурации контроллера прерываний понадобится проинициализировать специальную структуру и заполнить ее поля:

```
1. //Initialize the structure to configure the interrupt
2. //controller
3. NVIC_InitTypeDef NVIC_INITStruct;
4. // Select the interrupt controller channel to configure
5. /* In this case, set the interrupt from the 1st and 10th
6. timer*/
7. NVIC_INITStruct.NVIC_IRQChannel = TIM1_UP_TIM10_IRQn;
8. // Set priorities in the range from 0 to 15
9. NVIC_INITStruct.NVIC_IRQChannelPreemptionPriority = 0;
10.NVIC_INITStruct.NVIC_IRQChannelSubPriority = 0;
11.//Enable interrupt
12.NVIC_INITStruct.NVIC_IRQChannelCmd = ENABLE;
13.NVIC_Init(&NVIC_INITStruct);
```

Для корректной обработки прерываний, необходимо проверить флаг состояния того или иного прерывания, к примеру следующая функция

проверяет что в данный момент обрабатывается прерывание по переполнению от 10ого таймера-счетчика:

```
1.    TIM_GetITStatus(TIM10, TIM_IT_Update) != RESET
```

Также необходимо очищать соответствующий флаг, в конце тела обработчика прерывания:

```
1.    TIM_ClearITPendingBit(TIM10, TIM_IT_Update)
```

Кроме обработки «стандартных» прерываний, таких, к примеру, как переполнение таймера-счетчика или наличие сообщения в CAN-контроллере, можно синтезировать и программные прерывания, к примеру, по нажатию кнопки, связанной с 0-вым пином порта A (необходимы заголовочные файлы stm32f4xx_syscfg.h и stm32f4xx_exti.h библиотеки SPL**):**

```
1.    // Select the port and pin for being used as EXTI
2.    SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA,
3.    EXTI_PinSource0);
4.    // Struct for settings
5.    EXTI_InitTypeDef exti;
6.    // Select the line for which exit interrupt generation is
7.    //configured
8.    EXTI_Line = EXTI_Line0;
9.    // A interrupt is generated (not a event)
10.   exti.EXTI_Mode = EXTI_Mode_Interrupt;
11.   //Interrupt generated by the leading edge of the pulse
12.   exti.EXTI_Trigger = EXTI_Trigger_Rising;
13.   exti.EXTI_LineCmd = ENABLE;
14.   EXTI_Init(&exti);
```

Соответственно, обрабатывать в таком случае необходимо прерывание EXTI0_IRQn

Аналогово-цифровой преобразователь

АЦП – аналого-цифровой преобразователь (Analog-to-Dogital Converter). АЦП преобразует аналоговый сигнал в цифровой код.

Микроконтроллер – цифровое устройство, которое обменивается с внешним миром цифровыми сигналами: нулями и единицами. Однако, иногда перед микроконтроллером встает задача произвести измерение какой-либо плавно изменяющейся величины. Это может быть всё то, что принимает несколько промежуточных состояний (а не только два) например это может быть температура, напряжение, сила тока, освещенность и т.д. Однако пины микроконтроллера, а так же цифровые сигналы имеют только два состояния: высокий и низкий уровень. Для решения проблем измерения аналоговых величин и был придуман аналого-цифровой преобразователь.

Кратко принцип работы АЦП можно описать следующим образом: на вход АЦП поступает аналоговый сигнал и через некоторое время на выходе АЦП можно прочитать результат преобразования, то есть цифровое представление аналогово сигнала.

Приведем характеристики АЦП, представленные в контроллерах семейства STM32F4:

1. 12, 8 или 6-битные преобразования
2. Каждый из представленных в конкретной плате АЦП может обслуживать 19 каналов: от 16 внешних и 2 внутренних источников.
3. АЦП поддерживает различные режимы преобразования:
    - однократное
    - непрерывное
    - по триггеру
    - по таймеру
4. Результат выравнивается вправо или влево (настраивается)
5. АЦП имеет возможность генерации прерываний и сигналов для DMA
6. Скорость оцифровки сигнала – до 0.9 MSPS

Структура АЦП приведена на рисунке 1:

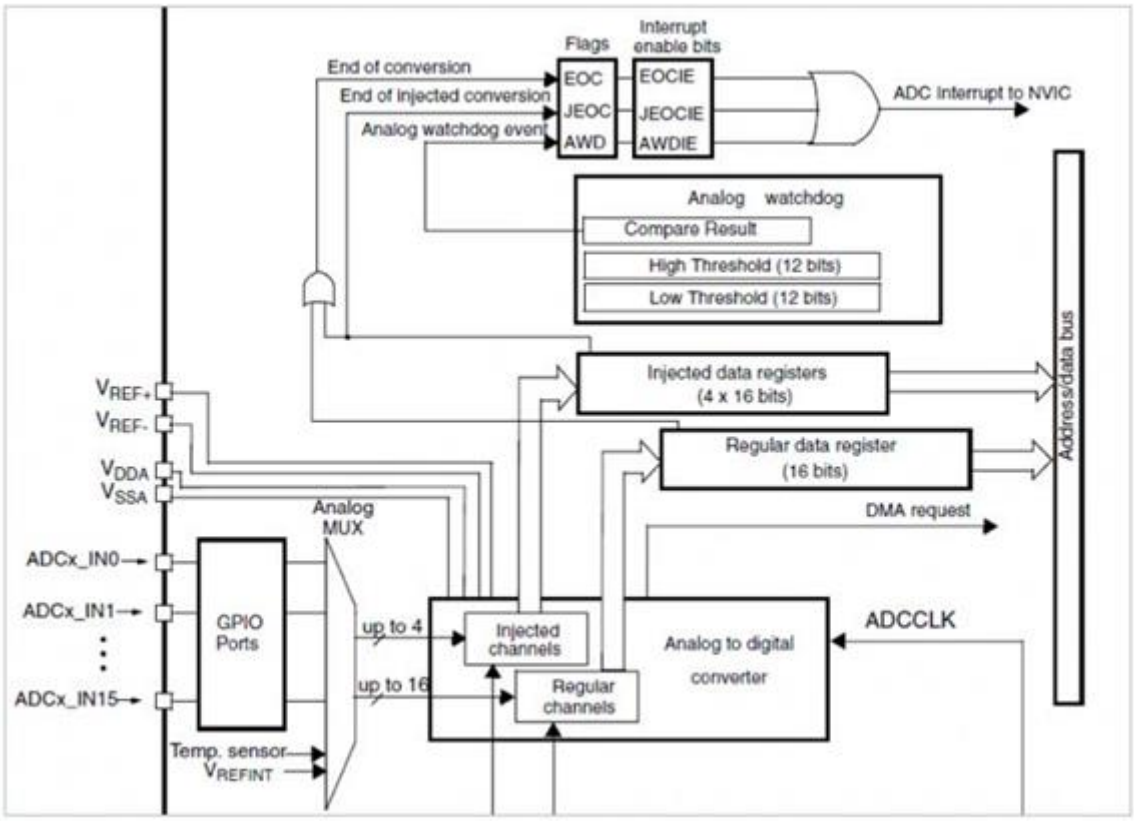

Рисунок 1 – Структура аналогово-цифрового преобразователя

Напряжение, которое поступает на АЦП, рассчитывают по формуле:

$$V_{adc} = \frac{V_{ref}}{2^{Nbits}} ADCValue$$

V_{ref} – опорное напряжение.

Еще одной особенностью АЦП, представленной в STM32F4 является наличие встроенного в АЦП – модуль датчика температуры окружающей среды, который так же можно включить программно.

Температуру, измеренную датчиком, высчитывают по формуле:

$$T^0 = \frac{V_{adc} - V25}{AVG\_SLOPE} + 25$$

V25 – напряжение при температуре 25 градусов. AVG_SLOPE - параметр, зависящий от опорного напряжение, в представленном МК он равен 0.0025.

Ход работы:

Задание состоит из следующих пунктов:

1. Изучить работу приоритетов прерываний, на примере одинаково сконфигурированных таймеров, с различными настройками приоритетов прерываний.

a. Приоритет прерывания одного таймера выше чем у другого

b. Одинаковые приоритеты прерываний. Тестирование работы контроллера прерываний вести с использованием отладчика и точек останова

2. Составить программу зажигающую диоды STAT1-4 «по кругу», переключение выбранного диода осуществлять через прерывание, генерируемое по нажатию USR_BTN.

3. Изучить работу АЦП подавая на любой удобный канал напряжение с потенциометра лабораторного стенда:

a. Найти граничные значения преобразования, используя однократные вычисления и отладчик (для данного пункта вычисления АЦП можно разрешать программно, используя функцию задержки)

b. Разбить диапазон на 4 равных промежутка

c. В зависимости от попадания очередной измеренной величины в конкретный промежуток зажигать 1, или 2, или 3, или 4 диода, где 4 горящих диода соответствуют максимальному значению величины. Запуск вычислений начинать по нажатию USR_BTN.

4. Составить программу измерения температуры окружающей среды, используя встроенный в АЦП датчик. Результаты наблюдать через отладчик.

Задание №1: Изучение работы приоритетов прерываний на примере одинаково сконфигурированных таймеров, но с различными настройками приоритетов прерываний.

Продемонстрируем последовательность действий для задания №1 в виде следующей схемы:

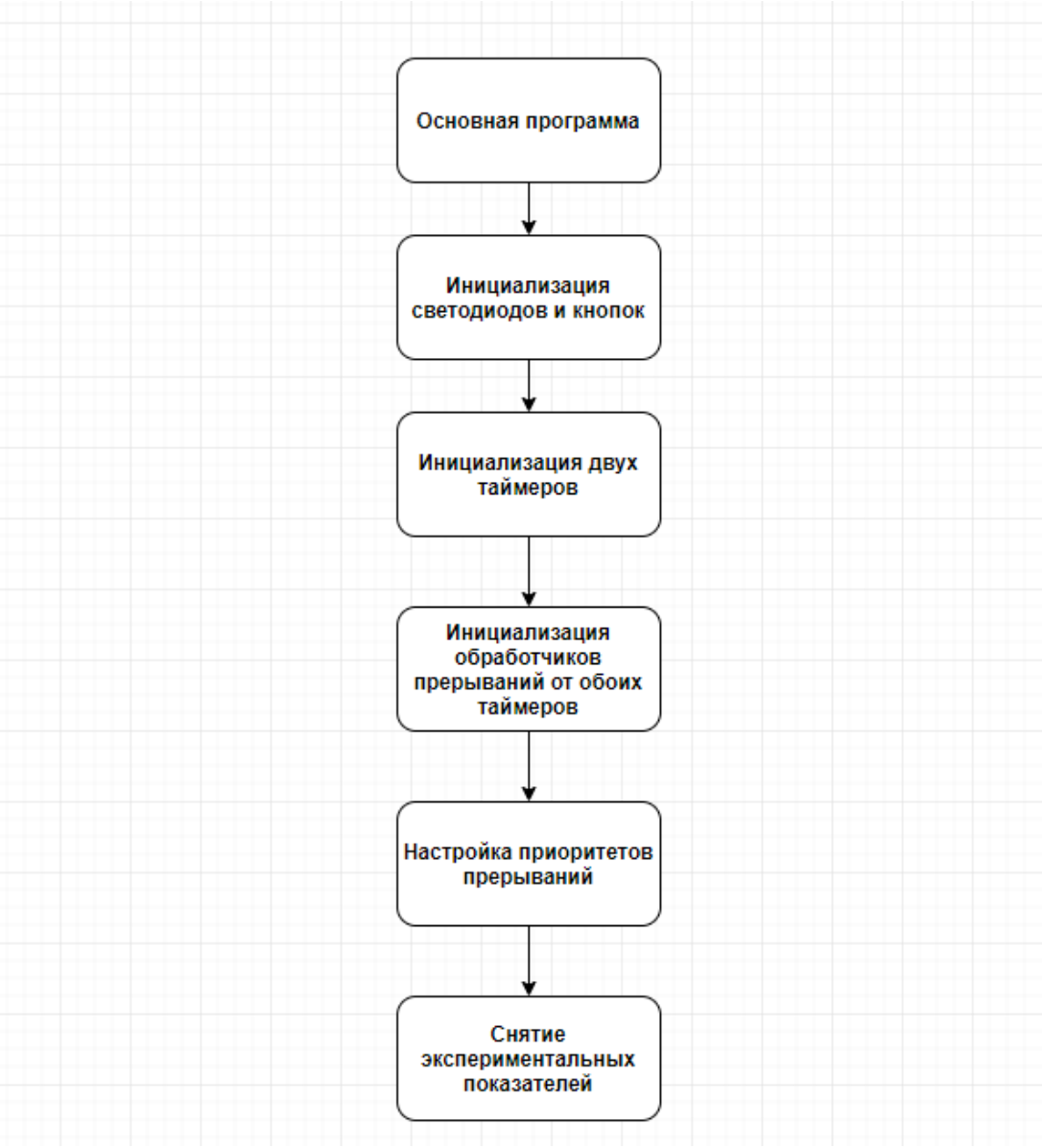

Рисунок 2 – Схема последовательности действии для задания №1

Проинициализируем кнопки и светодиоды. В дальнейшем, мы будем использовать данную инициализацию на других этапах выполнения данной лабораторной работы, так что эту часть можно опустить.

```
14. void BTN_and_LED_init (void) {
15.     /* For LED init */
16.     GPIO_InitTypeDef    GPIO_InitStruct;
17.     RCC_AHB1PeriphClockCmd(LED_RCC, ENABLE);
18.     GPIO_InitStruct.GPIO_Pin = LED_PIN_1 | LED_PIN_2
19.     | LED_PIN_3 | LED_PIN_4;
20.     GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;
21.     GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
22.     GPIO_InitStruct.GPIO_Speed = GPIO_Speed_100MHz;
23.     GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;
24.     GPIO_Init(LED_PORT, &GPIO_InitStruct);
25.     /* FOR User Button */
26.     RCC_AHB1PeriphClockCmd(BUTTON_USER_RCC, ENABLE);
27.     GPIO_InitStruct.GPIO_Pin = BUTTON_USER_PIN;
28.     GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN;
29.     GPIO_Init(BUTTON_USER_PORT, &GPIO_InitStruct);
30.     /* FOR WKUP */
31.     RCC_AHB1PeriphClockCmd(BUTTON_WKUP_RCC, ENABLE);
32.     GPIO_InitStruct.GPIO_Pin = BUTTON_WKUP_PIN;
33.     GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN;
34.     GPIO_Init(BUTTON_WKUP_PORT, &GPIO_InitStruct);
35. }
```

Проинициализируем два таймера. В нашем примере это будет 6 и 4:

Инициализация таймера 6:

```
1. void TIM6_init (void) {
2.     RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM6, ENABLE);
3.     TIM_TimeBaseInitTypeDef TIM_INITStruct;
4.     TIM_TimeBaseStructInit(&TIM_INITStruct);
5.     TIM_INITStruct.TIM_Prescaler = 25 - 1;
6.     TIM_INITStruct.TIM_Period = 2000;
7.     TIM_TimeBaseInit(TIM6, &TIM_INITStruct);
8.     TIM_ITConfig(TIM6, TIM_IT_Update, ENABLE);
9.     TIM_Cmd(TIM6, ENABLE);
10.    NVIC_EnableIRQ(TIM6_DAC_IRQn);
11. }
```

Инициализация таймера 4:

```
1.  void TIM4_init (void) {
2.     RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);
3.     TIM_TimeBaseInitTypeDef TIM_INITStruct;
4.     TIM_TimeBaseStructInit(&TIM_INITStruct);
5.     TIM_INITStruct.TIM_Prescaler = 25 - 1;
6.     TIM_INITStruct.TIM_Period = 500;
7.     TIM_TimeBaseInit(TIM4, &TIM_INITStruct);
8.     TIM_ITConfig(TIM6, TIM_IT_Update, ENABLE);
9.     TIM_Cmd(TIM4, ENABLE);
10.    NVIC_EnableIRQ(TIM6_DAC_IRQn);
11. }
```

Настроим два обработчика прерываний от обоих таймеров.

Для шестого таймера:

```
1. void TIM6_DAC_IRQHandler() {
2.      if (TIM_GetITStatus(TIM6, TIM_IT_Update) != RESET) {
3.        for(int i = 0; i < 120; i++){
4.          GPOI_ToggleBits(LED_PORT, LED_PIN_1);
5.          DELAY(50);
6.        }
7.        GPIO_ResetBits(LED_PORT, LED_PIN_1);
8.        TIM_ClearITPendingBit(TIM6, TIM_IT_Update);
9.      }
10.   }
```

Для четвертого таймера

```
1. void TIM4_DAC_IRQHandler() {
2.      if (TIM_GetITStatus(TIM4, TIM_IT_Update) != RESET) {
3.        for(int i = 0; i < 30; i++){
4.          GPOI_ToggleBits(LED_PORT, LED_PIN_4);
5.          DELAY(50);
6.        }
7.        GPIO_ResetBits(LED_PORT, LED_PIN_4);
8.        TIM_ClearITPendingBit(TIM3, TIM_IT_Update);
9.      }
10. }
```

Так как шестой таймер 120 раз переключает первый светодиод, а четвертый – 30 раз, то, следовательно, четвертый таймер работает с периодом

в четыре раза меньше, чем шестой, значит, его прерываний срабатывают в 4 раза чаще.

Теперь настроим приоритет двум прерываниям:

```
1. void set_ priorities(){
2.     NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
3.     //structure initialization
4.
5.
6.     NVIC_InitTypeDef NVIC_INITStruct;
7.
8.     //priority for TIM_6 form 0 to 15
9.     NVIC_INITStruct.NVIC_IRQChannel = TIM6_IRQn;
10.    NVIC_INITStruct.NVIC_IRQChannelPreemptionPriority = 0;
11.    NVIC_INITStruct.NVIC_IRQChannelSubPriority = 0;
12.    //Enable handler
13.    NVIC_INITStruct.NVIC_IRQChannelCmd = ENABLE;
14.    NVIC_Init(&NVIC_INITStruct);
15.
16.
17.    //priority for TIM_4 from 0 to 15
18.    NVIC_InitTypeDef NVIC_INITStruct1;
19.    NVIC_INITStruct1.NVIC_IRQChannel = TIM4_IRQn;
20.    NVIC_INITStruct1.NVIC_IRQChannelPreemptionPriority = 3;
21.    NVIC_INITStruct1.NVIC_IRQChannelSubPriority = 0;
22.    NVIC_INITStruct1.NVIC_IRQChannelCmd = ENABLE;
23.    NVIC_Init(&NVIC_INITStruct1);
24.}
```

Следующие два параметра используются для настройки приоритетов: PreemptionPriority и SubPriority. Действуют оба по принципу: чем меньше значение, тем больше приоритет.

Сконфигурируем все наши функции в основной программе:

```
1. int main(){
2.     BTN_and_LED_init();
3.     TIM6_init();
4.     TIM4_init();
5.     TIM6_DAC_IRQHandler();
6.     TIM4_DAC_IRQHandler();
7.     set_ priorities();
8.     while(1){

9.     }
10.}
```

Учтем следующий момент. Синий канал соответствует работе шестого таймера, а красный канал – четвертого.

Снимем показания с осциллографа:

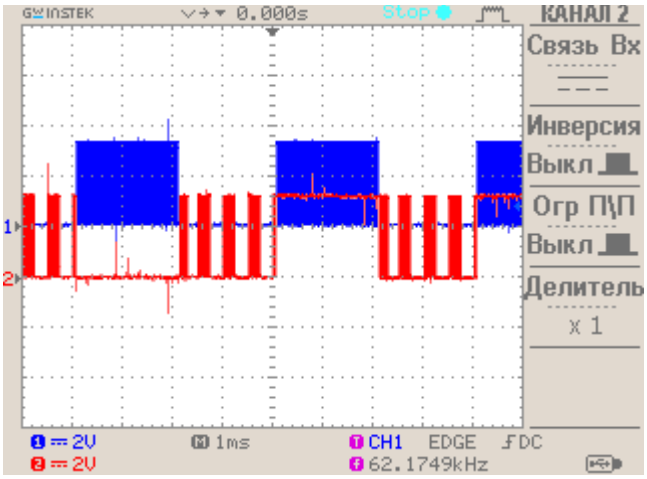

Рисунок 3 – Приоритет прерываний шестого таймера больше, чем четвертого

На плате можно заменить, что светодиод от четвертого таймера в тем моменты времени, когда мигает светодиод от шестого таймера не переключается, хотя должен если судить по частоте прерываний. Таким образом, прерываний четвертого таймера не может начать свою работу пока не закончится прерывание от шестого.

Если поменять приоритеты местами, видимо следующую картину:

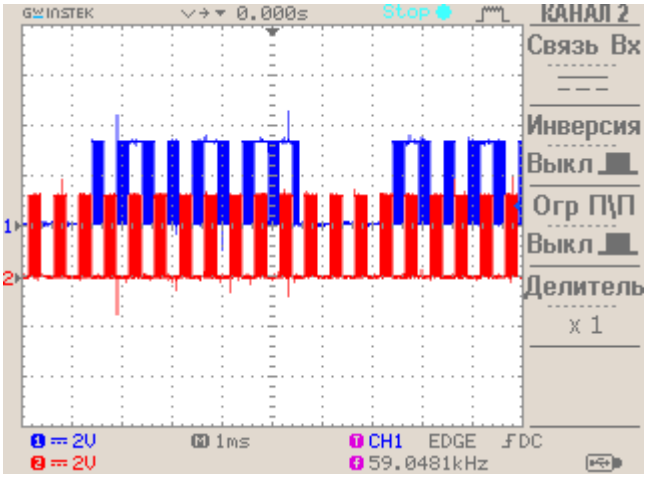

Рисунок 4 – Приоритет прерываний четвертого таймера больше, чем шестого

Видно, что прерываний от четвертого таймера прерывают обработку прерываний от шестого.

Теперь сделаем одинаковый приоритет, имеем следующую картину:

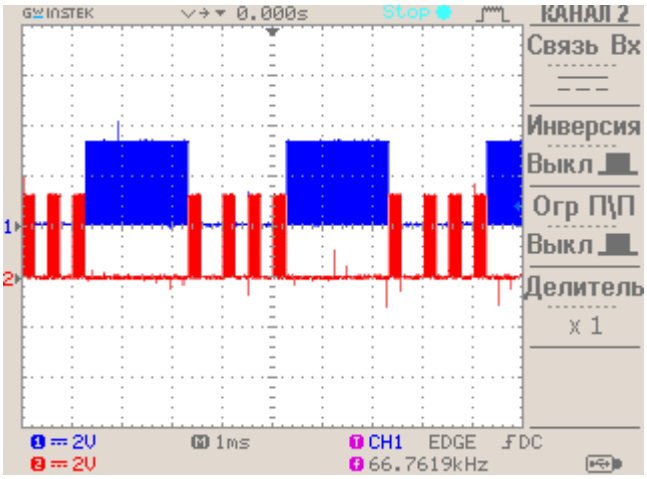

Рисунок 5 – Одинаковый приоритет прерываний

Задание №2: Составление пользовательской программы, зажигающей диоды STAT1-4 «по кругу»

Опишем последовательность действия для задания №2 в виде следующей схемы:

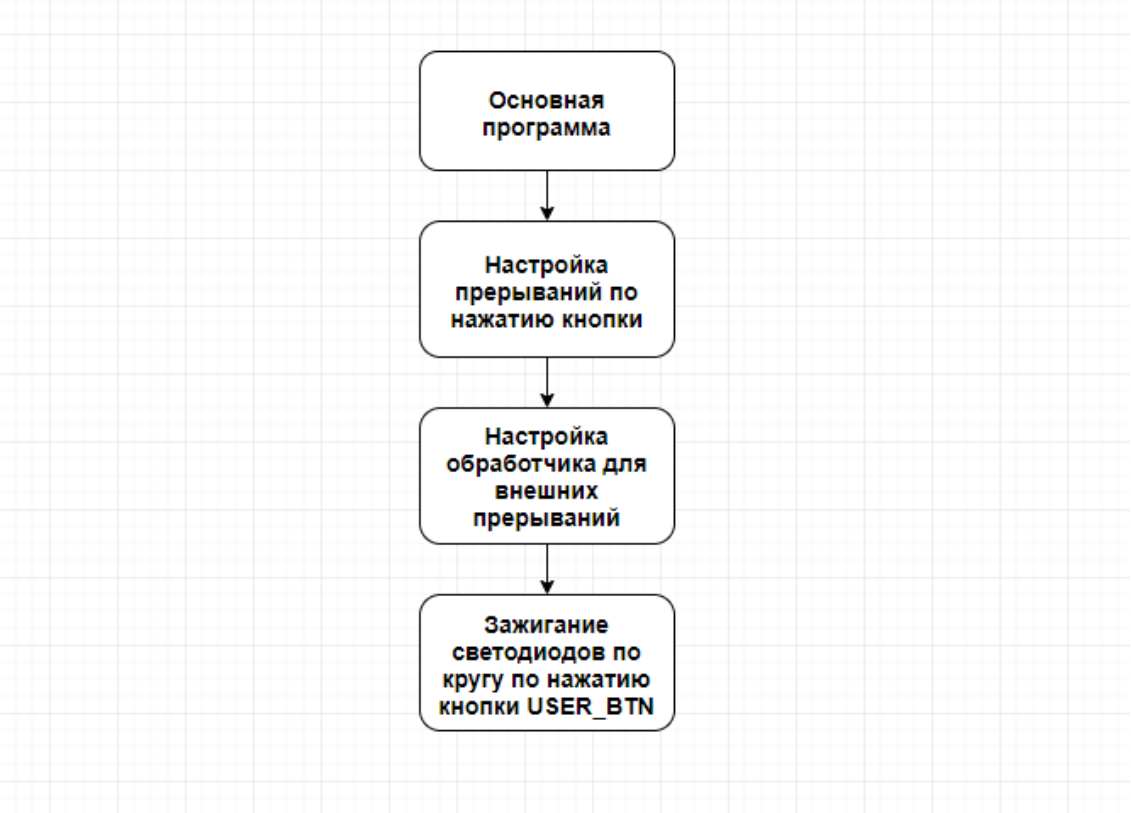

Рисунок 6 – Схема последовательности действии для задания №2

Настроим пользовательское прерывание от кнопки

```
void button_interrupt() {
    GPIO_ResetBits(LED_PORT, LED_PIN_2 | LED_PIN_3 |
    LED_PIN_4);
    GPIO_SetBits(LED_PORT, LED_PIN_1);
    /* For USER irq */
    EXTI_DeInit();
    // Select port and pin for used as EXTI
    SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOG,
                          EXTI_PinSource6;
    //Structure for configure
    EXTI_InitTypeDef exti;
    /*Select the line for which the generation of interrupt
    is configured*/
    exti.EXTI_Line = EXTI_Line6;
    // An interrupt is generated (not an event)
    exti.EXTI_Mode = EXTI_Mode_Interrupt;
    // Interrupt generated by the leading edge of the pulse
    exti.EXTI_Trigger = EXTI_Trigger_Rising;
    // Enable interrupt
    exti.EXTI_LineCmd = ENABLE;
    EXTI_Init(&exti);
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
    // Initialize the structure to configure the interrupt
    //controller
    NVIC_InitTypeDef NVIC_INITStruct;
    // Select the interrupt controller channel to configure
    //In this case, set the interrupt from the 6th timer
    NVIC_INITStruct.NVIC_IRQChannel = EXTI10_IRQn;
    // Set priorities in the range from 0 to 15
    NVIC_INITStruct.NVIC_IRQChannelPreemptionPrioritty = 0;
    NVIC_INITStruct.NVIC_IRQChannelSubPriority = 0;
    // Enable interrupt processing of the selected channel
    NVIC_INITStruct.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_INITStruct);
    NVIC_EnableIRQ (EXTI10_IRQn);
    EXTI_GenerateSWInterrupt(EXTI_Line6);
    EXTI_GenerateSWInterrupt(EXTI_Line6);
    EXTI_GenerateSWInterrupt(EXTI_Line6);
}
```

Проведем инициализацию обработчика пользовательских прерываний:

```
1. void EXTI10_IRQn_Handler() {
2.     if( EXTI_GetITStatus(EXTI_Line6) != RESET) {
3.        GPIO_ToggleBits(LED_PORT, LED_PIN_1);
4.     }
5.     EXTI_ClearITPendingBit(EXTI_Line6);
6. }
```

На плате можно заметить, что светодиоды переключаются по кругу каждый раз, когда происходит нажатие кнопки USER_BTN, источник переключения – генерируемое прерывание. При вызове события EXTI_Trigger_Rising происходит прерывание по переднему фронту импульса. Таким образом, нет необходимости в ожидании опускания кнопки.

Задание №3: Изучение работы АЦП

Опишем последовательность действий для работы с АЦП в виде следующей схемы:

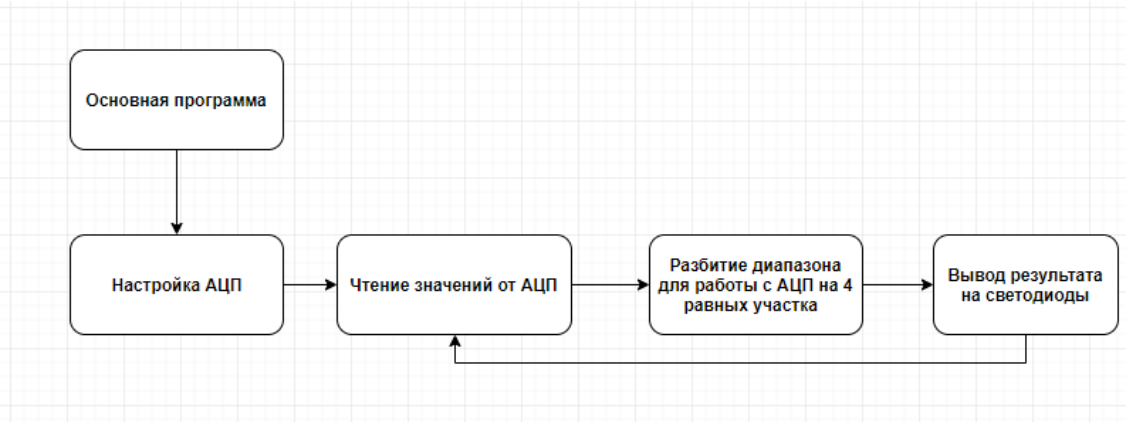

Рисунок 7 – Схема последовательности действий для задания №3

Для начала настроим АЦП для работы в Single-chanell режиме с использованием библиотеки SPL(файл stm32f4xx_adc.h):

```
1. void ADC_INIT() {
2.     /*Initialization of the structure for the configuration
3.     of the ADC*/
4.     ADC_InitTypeDef ADC_InitStructure;
5.     // Write default settings
6.     ADC_StructInit(&ADC_InitStructure);
7.     // Write default settings for the entire peripheral ADC
8.     ADC_CommonInitTypeDef adc_init;
9.     ADC_CommonStructInit(&adc_init);
10.    /* Enable ADC clocking */
11.    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
12.    /* reset ADC settings */
13.    ADC_DeInit();
14.    /* ADC work independently */
15.    adc_init.ADC_Mode = ADC_Mode_Independent;
16.    /* The frequency of the ADC - half the bus frequency */
17.    adc_init.ADC_Prescaler = ADC_Prescaler_Div2;
18.    /* Turn off multichannel conversion */
19.    ADC_InitStructure.ADC_ScanConvMode = DISABLE;
20.    /* Turn off continuous conversions */
21.    ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
22.    /* Start the conversion programmatically,
23.    and not by triggering */
24.    ADC_InitStructure.ADC_ExternalTrigConv=
25.    ADC_ExternalTrigConvEdge_None;
26.    /* 12 bit conversion. Result in 12 lower order bits */
27.    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
28.    /* Initialization of the peripheral ADC and ADC 1 with
29.    the selected settings */
30.    ADC_CommonInit(&adc_init);
31.    ADC_Init(ADC1, &ADC_InitStructure);
32.    /* Turn on the ADC1 */
33.    ADC_Cmd(ADC1, ENABLE);
34.}
```

Затем мы напишем функцию, которая запускает вычисления и корректно сохраняет результаты после преобразований АЦП:

```
1. u16_readADC1(u8 channel){
2.     /*Data is converted to ADC1, from the transmitted chan
3.     nel,
4.     first conversion priority, the conversion should not
5.     exceed 28 ADC operation cycles */
6.     // Allowing ADC1 operation
7.     ADC_RegularChannelConfig(ADC1, channel, 1,
8.     ADC_Sample_28Cycles)
9.     ADC_SoftwareStartConv (ADC1);
10.    // Waiting for end of conversion
11.    while (ADC_GetFlagStatus (ADC1, ADC_FLAG_EOC) == RESET);
12.    //Storethe result in the u16 variable
13.    return ADC_GetConversionValue (ADC1);
14.}
```

В функции main мы должны передать 10 канал для чтения данных с АЦП (так как необходимо считывать значения с потенциометра). Диапазон работы АЦП составляет 0 – 4096 (был найден при помощи отладчика). Его необходимо разбить на равные отрезки.

Зададим следующий алгоритм в функции main: в бесконечном цикле будем считывать значения с АЦП, и, в зависимости от его показаний, будем зажигать различное количество светодиодов.

Таким образом:

При значении от 3 до 1024 горит один светодиод. При значении 1023 до 2048 будут гореть два светодиода. При значении от 2047 до 3072 будут гореть три светодиода. При значении от 3071 до 4096 будет гореть все четыре светодиода.

```
1. int main(){
2.    int channel = 10;
3.    uint16_t count = readADC1(channel);
4.    while(1){
5.       if (abs(count - readADC1(channel)) >= 20)   {
6.         count = readADC1(channel);
7.         GPIO_ResetBits(LED_PORT, LED_PIN_1 | LED_PIN_2 |
8.         LED_PIN_3| LED_PIN_4);
9.       }
10.      if ((count > 3) && (count < 1024)) {
11.        GPIO_SetBits(LED_PORT, LED_PIN_1);
12.      }
13.      if ((count > 1023) && (count < 2048)) {
14.        GPIO_SetBits(LED_PORT, LED_PIN_1 | LED_PIN_2);
15.      }
16.      if ((count > 2047) && (count < 3072)) {
17.        GPIO_SetBits(LED_PORT, LED_PIN_1 | LED_PIN_2 |
18.        LED_PIN_3);
19.      }
20.      if ((count > 3071) && (count < 4096)) {
21.        GPIO_SetBits(LED_PORT, LED_PIN_1 | LED_PIN_2 |
22.        LED_PIN_3| LED_PIN_4);
23.      }
24.   }
25.}
```

Задание 4: Составить программу измерения температуры окружающей среды, используя встроенный в АЦП датчик.

Встроенный датчик температуры работает на 16 канале, следовательно, нужно изменить программу так, чтобы он читал значения с этого канала и зажигал соответствующие диоды в зависимости от температуры окружающей среды. Следует учесть, что интерпретация проводится следующим образом:

Если диапазон составляет до 960, то значит температура в помещении больше 20 градусов по Цельсию. В случае если диапазон до 970 – то температура в помещении больше 30 градусов по Цельсию и.т.д. В конце программы добавлена функцию Delay для того, чтобы замер температуры проводился реже.

В ходе выполнения программы на плате зажегся первый светодиод, а значение переменной temperature составило 963 (показание АЦП). При помощи формул было установлено, что температура составляет ~29^0C.

```
int main() {
  /* Leds and Buttons settings */
  BTN_and_LED_init();
  /* for ADC */
  ADC_INIT();
  /* enable temperature sensor */
  ADC_TempSensorVrefintCmd(ENABLE);
  int channel = 16;
  while(1){
    uint16_t temperature = readADC1(channel);
    GPIO_ResetBits(LED_PORT, LED_PIN_1 | LED_PIN_2 |
    LED_PIN_3 | LED_PIN_4);
    if (temperature > 960) {
      GPIO_SetBits(LED_PORT, LED_PIN_1);
    }
    if (temperature > 970) {
      GPIO_SetBits(LED_PORT, LED_PIN_2);
    }
    if (temperature > 980) {
      GPIO_SetBits(LED_PORT, LED_PIN_3);
    }
    if (temperature > 990) {
      GPIO_SetBits(LED_PORT, LED_PIN_4);
    }
    Delay(3000000);
  }
}
```

Вывод

В данной лабораторной работе были углублены знания о системах прерываний в МК, а также произошло ознакомление на практике с их приоритетами.

Необходимо отметить следующие особенности библиотеки SPL для работы с NVIC (Nested Vectored Interrupt Controller – контроллер прерываний) (файл misc.h): в ходе выполнения настройки приоритетов прерываний следует помнить, что два параметра - PreemptionPriority и SubPriority действуют по принципу: чем меньше значение, тем больший приоритет задается для этого прерывания. Таким образом, та часть программы, которая имеет больший приоритет прерываний, соответственно имеет полное право прервать другую программу. Если же приоритет обеих программ одинаковый, то и прерывать друг друга они не будут.

Также, ознакомились с АЦП микроконтроллера, и на практике было установлено минимальные и максимальные значения (диапазон 0 – 4096), интерпретировали эти значения для вывода результатов на диоды для частей 3 и 4 настоящей лабораторной работы. Необходимо отметить, что для четвертой части лабораторной работы следует, для выявления температуры среды, посчитать ее по специальным формулам, которые приведены в теоритической части лабораторной работы.