Experiment 6: Exception and Interrupt

In this experiment, student will learn how use Exception and Interrupt. Exception and Interrupt are mechanisms that occur within the microprocessor when an event or an abnormal Interrupt / Event (Exception) occurs while running called Run-time. Interrupts usually are the event program developer is interested in which not error happens by microcontroller run into the events that are predetermined. Because the method of handling an event is the same and can be handled by software, then, Exceptions and Interrupts can be disabled or enabled on demand by developer. Sometime Interrupts and Exceptions are considered as the same mechanism for STM32F103C8T6. The microcontroller works on Interrupt or Exception by switching from Thread / Unprivileged mode to Handler / Privileged mode. It is necessary to operate in Privileged mode because these events require the microprocessor to be executed with all privileges since software has to manage all memory and peripherals to handle exceptions. Causes of Exceptions and Interrupts can occur for a variety of reasons, inside peripherals, external inputs, or software, which are intentional or set to occur. When an Exception or Interrupt occurs, the microprocessor system will go to work in special predetermined software called Handler that are contained in specific address inside memory space. The Exception Handler for Interrupt are named IRQ0 to IRQ67 and called the Interrupt Service Routine (ISR). All Cortex-M Exception Handlers are listed in Table 1. From this table, it can be seen that each exception has its own unique memory address and has only a word or 4 bytes, putting program directly into this memory impossible. Thus, it can only store the vector to point to address of larger programs for that Exception. Therefore, when developing programs for executing Exceptions, this memory will be used to store group of vectors and is called a vector table that is handled by a nested vectored interrupt controller (NVIC) to handle each type of exception. Exceptions can be caused by both hardware and software. Hardware can be classified as nonmarkable and markable. Nonmarkable is a situation in which hardware has a problem and cannot program the system to ignore the alarm from this situation. For Markable is a situation in which the hardware is alerting the situation that comes from the hardware mostly from Peripheral and external signal through the GPIO pin of the IC. This kind of signal can be Ignore means no need to respond. What happens, an exception on the software side is a signal from a situation where the program has a problem (Fault), calls for management from the system (System Service Call) and performs Real-time Debug as shown in the Figure 1.

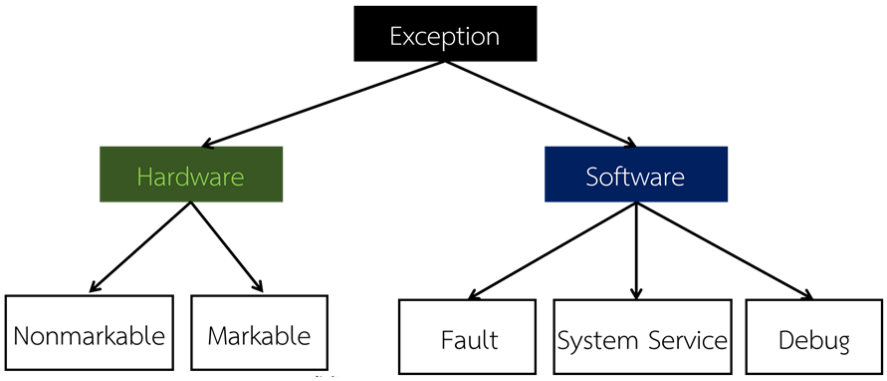


Figure 1. Exception Type

Table 1. Exception in Cortex-M3 microprocessor.

IRQ number	Offset Address	Vector
67	0x014C	IRQ67
. . .	. . .	. . .
2	0x0048	IRQ2
1	0x0044	IRQ1
0	0x0040	IRQ0
-1	0x003C	Systick
-2	0x0038	PendSV
. . .	. . .	. . .
-5	0x002C	SVCall
. . .	. . .	. . .
-10	0x0018	Usage fault
-11	0x0014	Bus fault
-12	0x0010	Memory management fault
-13	0x000C	Hard fault
-14	0x0008	NMI
	0x0004	Reset

1. External Interrupt (EXTI)

The EXTI peripheral is used to get an interrupt when a GPIO is toggling. EXTI is used to manage external interrupts and events. External interrupt is the occurrence of an event. At the excitation of the external IC via the STM32F103's GPIO pin, the excitation can be achieved by rising edge and/or falling edge from the external signal. There are 19 edge detector modules each type (edge detector), each input from the output signal can be set independently and can use up to 16 signals (line).

Simultaneously, each channel can be formed by setting in Register to select a group of GPIOs connected to that channel. Due to limitation of the IC chip, STM32F103C8T6 has only 10 external Interrupt Handlers for all external signal as shown in Figure 2. Therefore, to handle all GPIO, GPIO pins from different port must be grouped into on one interrupt signal for example PA0, PB0, PC0, Px0 is grouped into EXTI0. To use the Interrupts in C language, nested vectored interrupt controller (NVIC) must be set and the function of interrupt must be named as preassigned as shown in Figure 3.

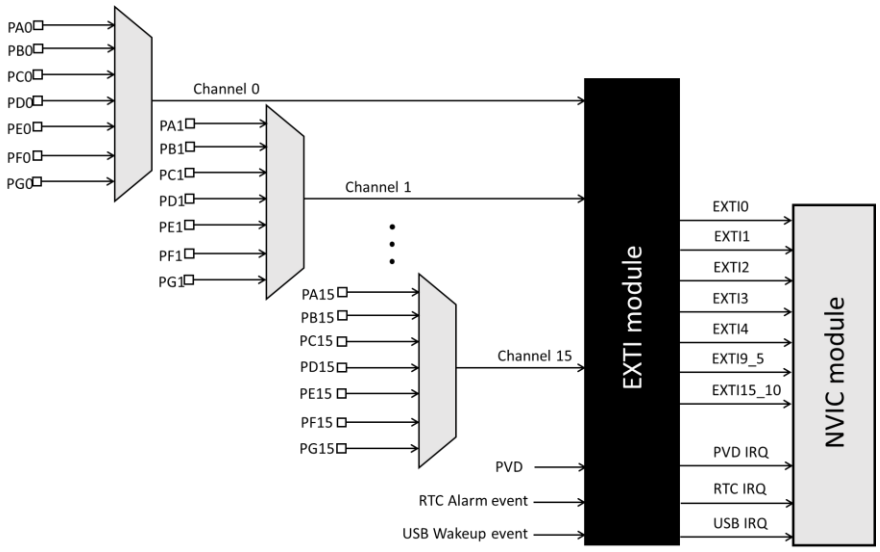


Figure 2. EXIT of STM32F103C8T6.

```
        .syntax     unified
        .thumb
// Exception vector table--Common to all Cortex-M3
_vectors:
        .word       __stack_end__
        .word       _start
        .word       NMI_Handler
        .word       HardFault_Handler
        .word       MemManage_Handler
        .word       BusFault_Handler
        .word       UsageFault_Handler
        .word       0
        .word       0
        .word       0
        .word       0
        .word       SVC_Handler
        .word       DebugMon_Handler
        .word       0
        .word       PendSV_Handler
        .word       SysTick_Handler
// Hardware interrupts specific to the STM32F103
        .word       WWDG_IRQHandler
        .word       PVD_IRQHandler
        .word       TAMPER_IRQHandler
        .word       RTC_IRQHandler
        .word       FLASH_IRQHandler
        .word       RCC_IRQHandler
        .word       EXTI0_IRQHandler
        .word       EXTI1_IRQHandler
        .word       EXTI2_IRQHandler
        .word       EXTI3_IRQHandler
        .word       EXTI4_IRQHandler
        .word       DMA1_Channel1_IRQHandler
        .word       DMA1_Channel2_IRQHandler
        .word       DMA1_Channel3_IRQHandler
        .word       DMA1_Channel4_IRQHandler
        .word       DMA1_Channel5_IRQHandler
        .word       DMA1_Channel6_IRQHandler
        .word       DMA1_Channel7_IRQHandler
        .word       ADC1_2_IRQHandler
        .word       USB_HP_CAN1_TX_IRQHandler
        .word       USB_LP_CAN1_RX0_IRQHandler
        .word       CAN1_RX1_IRQHandler
        .word       CAN1_SCE_IRQHandler
        .word       EXTI9_5_IRQHandler
        .word       TIM1_BRK_IRQHandler
        .word       TIM1_UP_IRQHandler
        .word       TIM1_TRG_COM_IRQHandler
        .word       TIM1_CC_IRQHandler
        .word       TIM2_IRQHandler
        .word       TIM3_IRQHandler
        .word       TIM4_IRQHandler
        .word       I2C1_EV_IRQHandler
        .word       I2C1_ER_IRQHandler
        .word       I2C2_EV_IRQHandler
        .word       I2C2_ER_IRQHandler
        .word       SPI1_IRQHandler
        .word       SPI2_IRQHandler
        .word       USART1_IRQHandler
        .word       USART2_IRQHandler
        .word       USART3_IRQHandler
        .word       EXTI15_10_IRQHandler
        .word       RTCAlarm_IRQHandler
        .word       USBWakeUp_IRQHandler
        .word       0
        .word       0
        .word       0
```

Figure 3. NVIC ISR function names.

Experiment 6.1

1. First, new project according to experiment 0 and assign project name as Lab6.
2. Add the **DelayMC( )** function from **Experiment 3** into the **main.c**. Add **USART_setup( )**, **USART1_sendC( )** and **USART1_putS( )** functions from **Experiment 2** into the **main.c**
3. Write down the **GPIO_configuration( )** and **NVIC_Configuration( )** and **EXIT_init( )** functions to the **main.c** file in the project as below:

```
1.  void GPIO_Configuration()
2.  {
3.     GPIO_InitTypeDef GPIO_InitStruct;
4.     // Enable clock for GPIOB and GPIOC and GPIOA and AFIO
5.     RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB | RCC_APB2Periph_GPIOC,ENABLE);
6.     RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_AFIO,ENABLE);
7.     RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1,ENABLE);
       RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
       GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable, ENABLE);
8.     // Configure PA0 and PA1 as Analog input
9.     GPIO_InitStruct.GPIO_Pin= GPIO_Pin_0 | GPIO_Pin_1;
10.    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AIN;
11.    GPIO_Init(GPIOA, &GPIO_InitStruct);
12.    // Configure PC13 as output push pull 50MHz
13.    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_13;
14.    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
15.    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_Out_PP;
16.    GPIO_Init(GPIOC, &GPIO_InitStruct);
17.    // Configure PB1 and PB15 as output push pull 50MHz
18.    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_1 | GPIO_Pin_15;
19.    GPIO_Init(GPIOB, &GPIO_InitStruct);
20.    // Configure PA9 as alternative output push pull for Tx
21.    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_9;
22.    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
23.    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF_PP;
24.    GPIO_Init(GPIOA, &GPIO_InitStruct);
25.    // Configure PA10 as input floating for Rx
26.    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_10;
27.    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
28.    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN_FLOATING;
29.    GPIO_Init(GPIOA, &GPIO_InitStruct);
30.    GPIO_InitStruct.GPIO_Pin= GPIO_Pin_3 | GPIO_Pin_4 | GPIO_Pin_8 | GPIO_Pin_9;
31.    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
32.    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN_FLOATING;
33.    GPIO_Init(GPIOB, &GPIO_InitStruct);
34. }
```

```
void NVIC_Configuration(void)
{
   NVIC_InitTypeDef NVIC_InitStructure;
   NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4);
   // Enable the EXTI3 Interrupt (IRQ vector for Pins Px3)
   NVIC_InitStructure.NVIC_IRQChannel = EXTI3_IRQn;
   NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
   NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
   NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
   NVIC_Init(&NVIC_InitStructure);
   // Enable the EXTI4 Interrupt (IRQ vector for Pins Px4)
   NVIC_InitStructure.NVIC_IRQChannel = EXTI4_IRQn;
   NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
   NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
   NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
   NVIC_Init(&NVIC_InitStructure);
   // Enable the EXTI9_5 Interrupt (IRQ vector for Pins Px8 and Px9)
   NVIC_InitStructure.NVIC_IRQChannel = EXTI9_5_IRQn;
   NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
   NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
   NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
   NVIC_Init(&NVIC_InitStructure);
}
```

```
void EXIT_init(void)
{
   EXTI_InitTypeDef EXTI_InitStructure;
   // Connect EXTI Line to PB3 PB4 PB8 and PB9
   GPIO_EXTILineConfig(GPIO_PortSourceGPIOB, GPIO_PinSource3) ;
   EXTI_InitStructure.EXTI_Line = EXTI_Line3;   //Pin PB3
   EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
   EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising; //Rising to Interrupt
   EXTI_InitStructure.EXTI_LineCmd = ENABLE;              //Enable Interrupt
   EXTI_Init(&EXTI_InitStructure);

   GPIO_EXTILineConfig(GPIO_PortSourceGPIOB, GPIO_PinSource4);
   EXTI_InitStructure.EXTI_Line = EXTI_Line4;   // Pin PB4
   EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
   EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising; //Rising to Interrupt
   EXTI_InitStructure.EXTI_LineCmd = ENABLE;              // Enable Interrupt
   EXTI_Init(&EXTI_InitStructure);

   GPIO_EXTILineConfig(GPIO_PortSourceGPIOB, GPIO_PinSource8);
   EXTI_InitStructure.EXTI_Line = EXTI_Line8;   // Pin PB8
   EXTI_Init(&EXTI_InitStructure);
   GPIO_EXTILineConfig(GPIO_PortSourceGPIOB, GPIO_PinSource9);
   EXTI_InitStructure.EXTI_Line = EXTI_Line9;   // Pin PB9
   EXTI_Init(&EXTI_InitStructure);
}
```

4. Write down the **EXTI3_IRQHandler( )**, **EXTI4_IRQHandler( )** and **EXTI9_5_IRQHandler( )** functions to the **main.c** file in the project before main( ) function as below:

```
1. void EXTI3_IRQHandler(void)
2. {
3.    USART1_putS("Interrupt occurred ! \n\r");
4.    if (EXTI_GetITStatus(EXTI_Line3) != RESET)
5.    {
6.          USART1_putS("Button 3 is pressed \n\r");
7.          //Clear the EXTI line 3 pending bit
8.    }
9.    EXTI_ClearITPendingBit(EXTI_Line3);
10.}
```

```
1. void EXTI4_IRQHandler(void)
2. {
3.    USART1_putS("Interrupt occurred ! \n\r");
4.    if (EXTI_GetITStatus(EXTI_Line4) != RESET)
5.    {
6.          USART1_putS("Button 4 is pressed \n\r");
7.          //Clear the EXTI line 3 pending bit
8.    }
9.    EXTI_ClearITPendingBit(EXTI_Line4);
10.}
```

```
11.void EXTI9_5_IRQHandler(void)
12.{
13.   USART1_putS("Interrupt occurred ! \n\r");
14.   USART1_putS("But which switch is pressed 1 or 2 ??\n\r");
15.   EXTI_ClearITPendingBit(EXTI_Line8);
16.   EXTI_ClearITPendingBit(EXTI_Line9);
17.}
```

4. Edit the **main( )** function in **main.c** as follow:

```
1. int main(void)
2. {
3.   unsigned char state = 0;
4.   GPIO_Configuration();
5.   GPIO_WriteBit(GPIOC,GPIO_Pin_13,0);
6.   USART_setup();
7.   NVIC_Configuration();
8.   EXIT_init();
```

```
9.    while(1)
10.   {
11.      if (state == 0) {
12.          GPIO_WriteBit(GPIOB,GPIO_Pin_15,1);
13.          state = 1;      }
14.      else {
15.          GPIO_WriteBit(GPIOB,GPIO_Pin_15,0);
16.          state = 0;      }
17.      DelayMC(100);
18.   }
19. }
```

5. Compile and program into the board. Connect the USB-to-Serial to desktop or notebook and find the virtual com port number via device manager. Open ProcessingGrapher program, set to com port number and Baud rate to 115200 and click connect. Reset the STM32F103C8T6 and explain the result from the experiment when press the switches.

6. Modify the **`EXTI9_5_IRQHandler(void)`** to show which switch (switch 1 or 2) is pressed.

Lab instructor / TA signature

2. ADC interrupt

From the previous experiment, Figure 3 shows variety of Interrupt Service Routine can be used. In this experiment ADC1 interrupt will be explored.

Experiment 6.2

1. Use the same project as experiment 6.1. Add the **ADC_NVIC_Config( )**, and **ADC_Interrupt_Init( )** into the **main.c** as follow.

```
void ADC_Interrupt_Init(void)
{
	USART1_putS("ADC_Interrupt_Init \n\r");
	ADC_InitTypeDef  ADC_InitStructure;
	/* PCLK2 is the APB2 clock */ /* ADCCLK = PCLK2/8 = 72/8 = 9MHz*/
	RCC_ADCCLKConfig(RCC_PCLK2_Div8);
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
	ADC_DeInit(ADC1);
	ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
	/* Disabple the scan conversion do one at a time */
	ADC_InitStructure.ADC_ScanConvMode = DISABLE;
	ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
	/* Start conversion by software, not an external trigger */
	ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
	ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
	ADC_InitStructure.ADC_NbrOfChannel = 1;
	ADC_Init(ADC1, &ADC_InitStructure);
	/* Enable Interrupt Event EOC*/
	ADC_ITConfig(ADC1, ADC_IT_EOC, ENABLE);
	/* Enable ADC1 */
	ADC_Cmd(ADC1, ENABLE);
	/* Read from Channel 0 of ADC1 */
	ADC_RegularChannelConfig(ADC1, 0, 1, ADC_SampleTime_55Cycles5);
}
```

```
void ADC_NVIC_Config(void)
{
	NVIC_InitTypeDef NVIC_InitStructure;
	NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4);
	// Configure interrupt priority
	NVIC_InitStructure.NVIC_IRQChannel = ADC1_2_IRQn;
	NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
	NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
	NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
	NVIC_Init(&NVIC_InitStructure);
}
```

2. Add function **__reverse ( )** and **itoa( )** from **Experiment 4**. Add **ADC1_2_IRQHandler( )** function to the **main.c** as follow:

```
unsigned int ADC_interrupt_value=0;

void ADC1_2_IRQHandler()
{
     if (ADC_GetITStatus(ADC1,ADC_IT_EOC) !=RESET)
     {
     // Read ADC conversion value
          ADC_interrupt_value = ADC_GetConversionValue(ADC1);
     }
     ADC_ClearITPendingBit(ADC1,ADC_IT_EOC);
}
```

3. Edit the **main( )** function in **main.c** as follow:

```
int main(void)
{
  unsigned char state = 0, sbuf[20];
  GPIO_Configuration();
  GPIO_WriteBit(GPIOC,GPIO_Pin_13,0);
  USART_setup();
  NVIC_Configuration();
  EXIT_init();
  ADC_NVIC_Config();
  ADC_Interrupt_Init();
  /* Start the conversion by Software */
  ADC_SoftwareStartConvCmd(ADC1, ENABLE);
  while(1)
  {
     if (state == 0) {
         GPIO_WriteBit(GPIOB,GPIO_Pin_15,1);
         state = 1;     }
     else {
         GPIO_WriteBit(GPIOB,GPIO_Pin_15,0);
         state = 0;     }
     itoa(ADC_interrupt_value,sbuf, 10);
     USART1_putS(sbuf);
     USART1_putS("\n\r");
     DelayMC(100);
  }
}

```

4. Compile and program into the board. Connect the USB-to-Serial to desktop or notebook and find the virtual com port number via device manager. Open ProcessingGrapher program, set to com port number and Baud rate to 115200 and click connect. Reset the STM32F103C8T6 and rotate variable resistor VR0 on KU Lab board explain the result and also try to press switch again.

_____

_____

_____

_____

_____

_____

_____

Lab instructor / TA signature

3. Timer Interrupt for precise time delay

Timer is the module that related to (real) time thus we can use this module to generate time. In this topic, timer interrupt will be used to generate precise delay time. The timer3 will be used to generate 1 msec and it will increase global time variable to count time. The delay function will calculate different time between beginning and current time using global time variable and await until delay is done.

Experiment 6.3

1. Use the same project as experiment 6.2. Add **TIM3_Interrupt_Setup( )** , **TIM3_NVIC_Config( )** , **TIM3_IRQHandler( )** and **DelayTimer( )** functions into **main.c** as follow:

```
1.  void TIM3_Interrupt_Setup()
2.  {
3.      TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStructure;
4.      TIM_OCInitTypeDef TIM_OCInitStructure;
5.      /* TIM3 is connected to APB1 bus,
6.      timer_tick_frequency = 72,000,000 / (71 + 1) = 72,000,000/71 = 1,000,000 Hz
7.      1,000,000 Hz -> 1 MHz -> T = 1 usec
8.      1000 times 1 usec = 1000 use = 1 msec */
9.      RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3,ENABLE);
10.     /*Timer Base Initialization for timer3  */
11.     TIM_TimeBaseInitStructure.TIM_Prescaler=0x47; // 0x47 = 72 -1 = 71  //
12.     TIM_TimeBaseInitStructure.TIM_CounterMode = TIM_CounterMode_Up ;
13.     TIM_TimeBaseInitStructure.TIM_Period = 1000-1;  // 1 msec
```

```
    TIM_TimeBaseInitStructure.TIM_ClockDivision     = TIM_CKD_DIV1;
    TIM_TimeBaseInit(TIM3,&TIM_TimeBaseInitStructure);
    TIM_ITConfig(TIM3,TIM_IT_Update,ENABLE);
}
```

```
void TIM3_NVIC_Config(void)
{
    NVIC_InitTypeDef NVIC_InitStructure;
    // Priority grouping
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_3);
    // Configure interrupt priority
    NVIC_InitStructure.NVIC_IRQChannel = TIM3_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
```

```
unsigned int time_msec =0;
unsigned char nstate = 0;

void TIM3_IRQHandler(void)
{
    if(TIM_GetITStatus(TIM3, TIM_IT_Update) != RESET)
    {
        time_msec++;
        if (nstate == 0) {
            nstate = 1;
            GPIO_WriteBit(GPIOB,GPIO_Pin_15,0);
        }
        else {
            nstate = 0;
            GPIO_WriteBit(GPIOB,GPIO_Pin_15,1);
        }
    }
    TIM_ClearITPendingBit (TIM3,TIM_IT_Update);
}
```

```

void DelayTimer(unsigned int msec)
{
    unsigned int current_time = 0;
    current_time = time_msec;
    while( (current_time+msec) >= time_msec);
}
```

2. Edit the **main( )** function in **main.c** as follow:

```
int main(void)
{
  unsigned char state = 0;
  GPIO_Configuration();
  GPIO_WriteBit(GPIOC,GPIO_Pin_13,0);
  USART_setup();
  TIM3_NVIC_Config();
  TIM3_Interrupt_Setup();
  /* Start the timer 3 */
  TIM_Cmd(TIM3, ENABLE);
  while(1)
  {
     if (state == 0) {
         GPIO_WriteBit(GPIOB,GPIO_Pin_1,1);
         state = 1;     }
     else {
         GPIO_WriteBit(GPIOB,GPIO_Pin_1,0);
         state = 0;     }
     DelayTimer(1000);
  }
}
```

3. Compile and program into the board. Use oscilloscope to measure the frequency of PB15 and PB1. Explain the result below, and what will happen if we run this code more than 4,294,968 seconds or 71,583 minutes?

Frequency of PB1 = _____ Hz

Frequency of PB15 = _____ Hz

_____

_____

_____

_____

_____

_____

_____

Lab instructor / TA signature

4. USART Interrupt

USART module is the module that requires time to operate especially for receiving data. Regularly from the previous chapter, pulling technique is used to receive data, waiting in a loop until data is arrived USART RXN. In order to solve this problem, interrupt is used for receiving data from RXN, therefore, no waiting loop is required. In this experiment, STM32F103C8T6 will not wait for data to arrive but use interrupt technique that microcontroller will serve USART data only when data is arrived.

Experiment 6.4

1. Use the same project as experiment 6.3. Add **USART1_NVIC_Config( )** , **USART1_IRQHandler( )**, **TIM3_IRQHandler( )** and **DelayTimer( )** functions into **main.c** as follow:

```
void USART1_NVIC_Config(void)
{
	NVIC_InitTypeDef NVIC_InitStructure;
	// Priority grouping
	NVIC_PriorityGroupConfig(NVIC_PriorityGroup_3);
	// Configure interrupt priority
	NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
	NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
	NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
	NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
	NVIC_Init(&NVIC_InitStructure);
}
```

```
unsigned char nstate2 = 0;
void USART1_IRQHandler(void)
{
	unsigned char sData = 0;
	/* RXNE handler */
	if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET)
	{
		sData = USART_ReceiveData(USART1);
		if(sData == 't')
		{
			if (nstate == 0) {
				nstate = 1;
				GPIO_WriteBit(GPIOB,GPIO_Pin_15,1);
			}
			else {
				nstate = 0;
```

```
                    GPIO_WriteBit(GPIOB,GPIO_Pin_15,0);
               }
               USART1_putS(" { Done! } \n\r");
          }
          if(sData == 'r')
          {
               if (nstate2 == 0) {
                    nstate2 = 1;
                    GPIO_WriteBit(GPIOB,GPIO_Pin_1,1);
               }
               else {
                    nstate2 = 0;
                    GPIO_WriteBit(GPIOB,GPIO_Pin_1,0);
               }
               USART1_putS(" { Done! } \n\r");
          }
     }
}
```

2. Edit the **main( )** function in **main.c** as follow:

```
int main(void)
{
  unsigned char state = 0;
  GPIO_Configuration();
  GPIO_WriteBit(GPIOC,GPIO_Pin_13,0);
  USART_setup();
  USART1_NVIC_Config();
  USART_ITConfig(USART1,USART_IT_RXNE,ENABLE);
  USART1_putS("<<< Waiting for t and r command >>>\n\r");
  while(1)
  {
     DelayMC(1000);
  }
}
```

3. Compile and program into the board. Open ProcessingGrapher program, set to com port number and Baud rate to 115200 and click connect. Reset the STM32F103C8T6 and explain the result when send ‘r’ or ‘t’ to the board

_____

_____

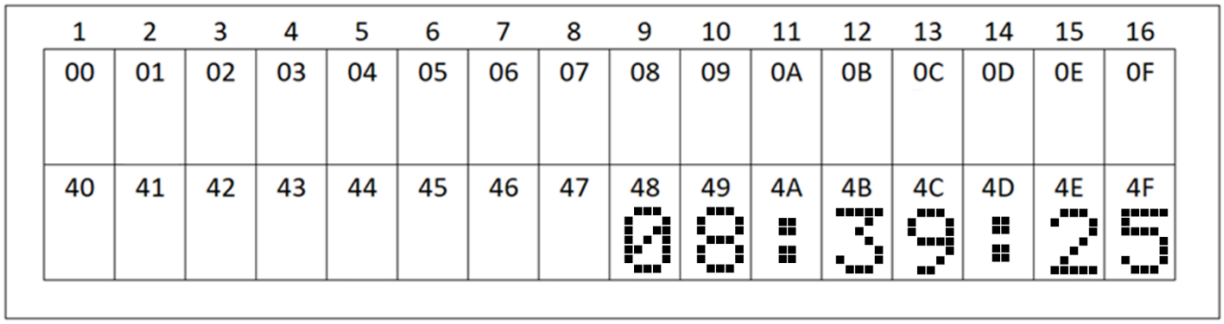
_____

_____

Lab instructor / TA signature

Homework

1. Write a clock using Timer 3 as time reference and display on LCD module as **hh:mm:ss** format below. This clock will count up every 1 sec.



Lab instructor / TA signature

2. Write a program to change the Duty Cycle of PWM signal with frequency of 1000 Hz by switch 1 and switch 2 using interrupt technique. When switch 1 is pressed the Duty Cycle increase by 2% and when switch 2 is pressed the Duty Cycle decrease by 2%.

Lab instructor / TA signature