GEBZE TECHNICAL UNIVERSITY

ELECTRONICS ENGINEERING

ELEC 335

Microprocessors Lab

LAB #4

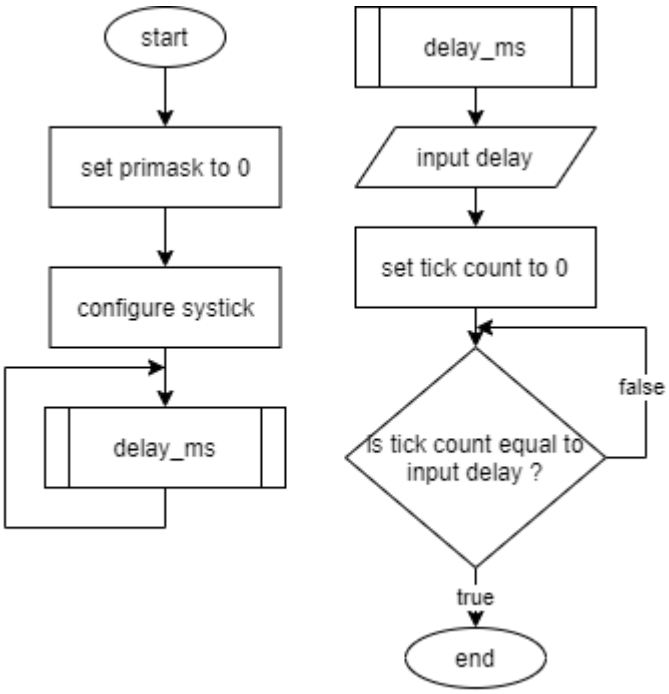
Prepared by
1) 1801022037 Ömer Emre POLAT

1. Introduction

In this lab we will look into timers including SysTick timer, internal timers and watchdog timers.

2. Problem 1

In this problem we will implement a delay function with the time input using SysTick interrupts as a timer. First we can create the flowchart of this code.

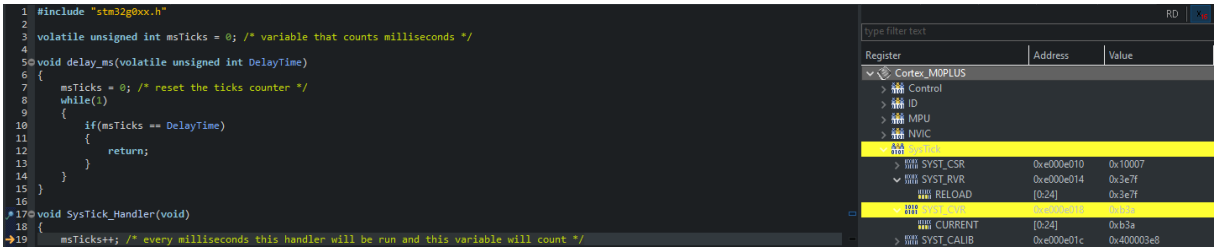


With the flowchart done we can write the actual code using the flowchart as a template.

```
#include "stm32g0xx.h"

volatile unsigned int msTicks = 0; /* variable that counts milliseconds */

void delay_ms(volatile unsigned int DelayTime)
{
	msTicks = 0; /* reset the ticks counter */
	while(1)
	{
		if(msTicks == DelayTime)
		{
			return;
		}
	}
}

void SysTick_Handler(void)
{
```

```
	msTicks++; /* every milliseconds this handler will be run and this
variable will count */
}

int main(void) {

	__set_PRIMASK(0U); /* set PRIMASK to 0 */

	SysTick_Config(16000); /* set SysTick reload value to 16000 and enable
SysTick handler */

	delay_ms(5000); /* wait for 1000ms */

    while(1)
    {

    }

    return 0;
}
```

With the code done we can test the code using debug mode in STM32 IDE.



As we can see the current count register count for the given value of 16000 which is 1/1000 of the clock speed which results in a millisecond delay. For every 16000 clock ticks a tick counter variable is incremented. This variable is necessary for the delay function with a millisecond input.

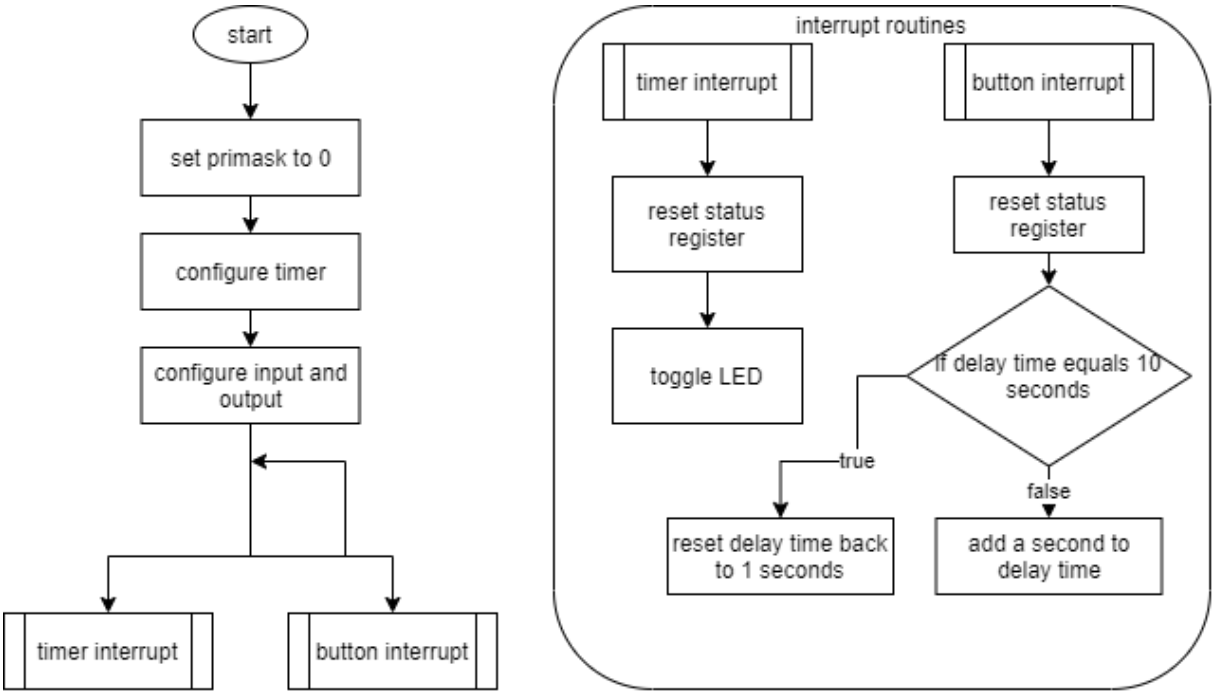
We have two types of methods for testing the accuracy of this delay function. Software and hardware methods.

For Software methods we can set another internal timer inside the board and use that timer as the comparison counter. When the delay function begins we can start the internal counter and when the delay function ends we can stop the counter and check the counter and calculate the time it took for the function to run.

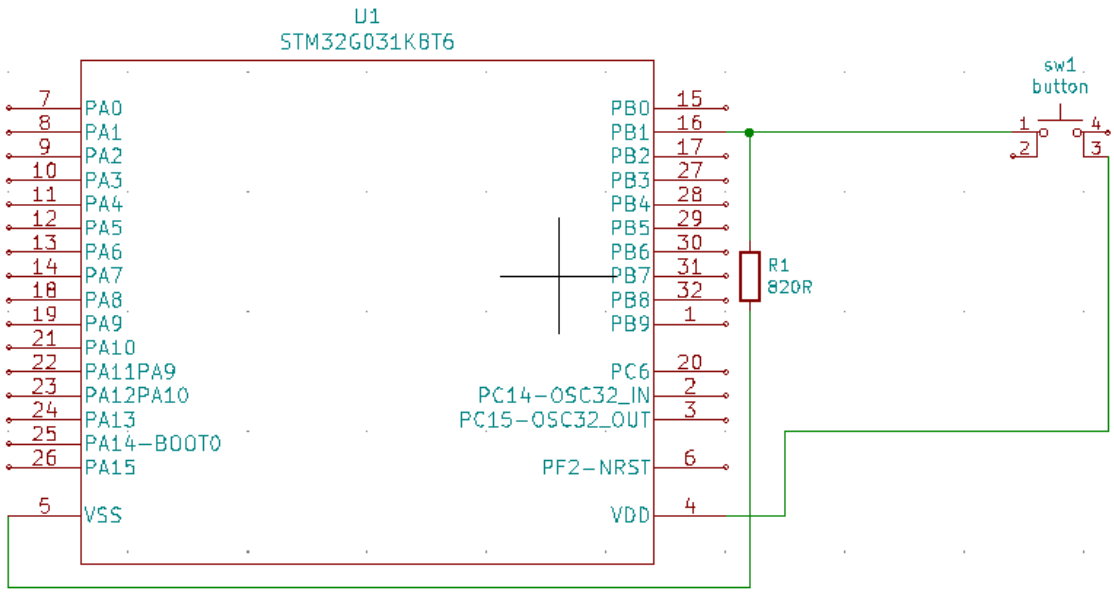
For Hardware methods we can set another board to work as an external counter that will count when the delay function is ran. This communication between boards can be set using IO pins.

3. Problem 2

For this problem we will implement a blinking LED that blinks with a certain interval. This interval will be determined using an external button. First we can create the flowchart for the code.



After the flowchart is created we can choose the pins for the LED and the button and create the block diagram of the connections. We can choose PB1 as the input pin.



Now that the block diagram is created the code can be created using the choosen pins.

```c
#include "stm32g0xx.h"

volatile uint32_t DelayTime = 1000;  /* 1000 ms delay time */

void EXTI0_1_IRQHandler(void)
{
      EXTI->RPR1 |= (1U << 1); /* clear pending on B1 */

    if(DelayTime != 10000)
    {
        DelayTime += 1000; /* increment the delay time by 1000 ms */
    }
    else
    {
        DelayTime = 1000; /* reset back the delay time */
    }

    TIM1->ARR = DelayTime; /* update autoreload time */
}

void TIM1_BRK_UP_TRG_COM_IRQHandler(void)
{
    TIM1->SR &= ~(1U << 0); /* reset status register */
    GPIOC->ODR ^= (1U << 6); /* toggle LED */
}

int main(void) {

    /* setting up the button input LED output and interrupt */
      __set_PRIMASK(0U); /* set PRIMASK to 0 */

      RCC->IOPENR |= (1U << 1); /* Enable GPIOB clock */
      RCC->IOPENR |= (1U << 2); /* Enable GPIOC clock */

      GPIOB->MODER &= ~(3U << 2*1); /* Setup PB1 as input */

      GPIOC->MODER &= ~(2U << 2*6); /* Setup PC6 as output */
      GPIOC->MODER |= (1U << 2*6);

      EXTI->RTSR1 |= (1U << 1); /* Rising edge selection B1 */
      EXTI->EXTICR[0] |= (1U << 8*1); /* 1U to select B1 from mux */
      EXTI->IMR1 |= (1U << 1); /* interrupt mask register B1 */

      EXTI->RPR1 |= (1U << 1); /* clear pending on B1 */

      NVIC_SetPriority(EXTI0_1_IRQn, 0x0U); /* Setting priority for EXTI0_1 */
      NVIC_EnableIRQ(EXTI0_1_IRQn); /* Enabling EXTI0_1 */

    /* setting up the timer and interrupt */

    RCC->APBENR2 |= (1U << 11); /* Enable TIM1 clock */

    TIM1->CR1 = 0; /* resetting control register */
    TIM1->CR1 |= (1U << 7); /* Autoreload preload enable */
    TIM1->CNT = 0; /* reset the timer counter */

    TIM1->PSC = 15999; /* prescaler 16000 so that we count ever milliseconds */
    TIM1->ARR = DelayTime; /* set the autoreload register */
```

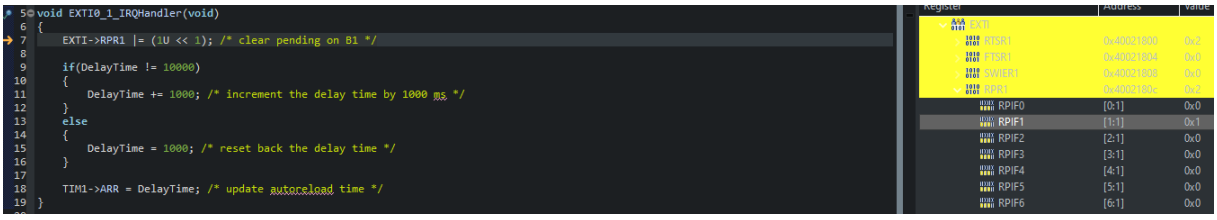
```
    TIM1->DIER |= (1U << 0); /* update interrupt enable */
    TIM1->CR1 |= (1U << 0); /* Enable TIM1 */

    NVIC_SetPriority(TIM1_BRK_UP_TRG_COM_IRQn, 0x3U); /* Setting priority for
TIM1 */
    NVIC_EnableIRQ(TIM1_BRK_UP_TRG_COM_IRQn); /* Enabling TIM1 */

    while(1)
    {

    }

    return 0;
}
```

The code works by having a global variable that stores the current delay amount in ms. The timer prescaling is set so that the (autoreload * (prescaler + 1)) will be equal to the clock speed. In this case we choose 15999 for the prescaler value. This value ensures that the autoreload value will be delay in milliseconds. After that we can write 1000 in autoreload register to delay for 1000 milliseconds. Using this functionality we can change the autoreload value to extend or shorten the delay time.

The timer interrupt toggles the LED and the button interrupt increments or resets the DelayTime variable.

Now that the code is written we can test the code in STM32 debug mode.

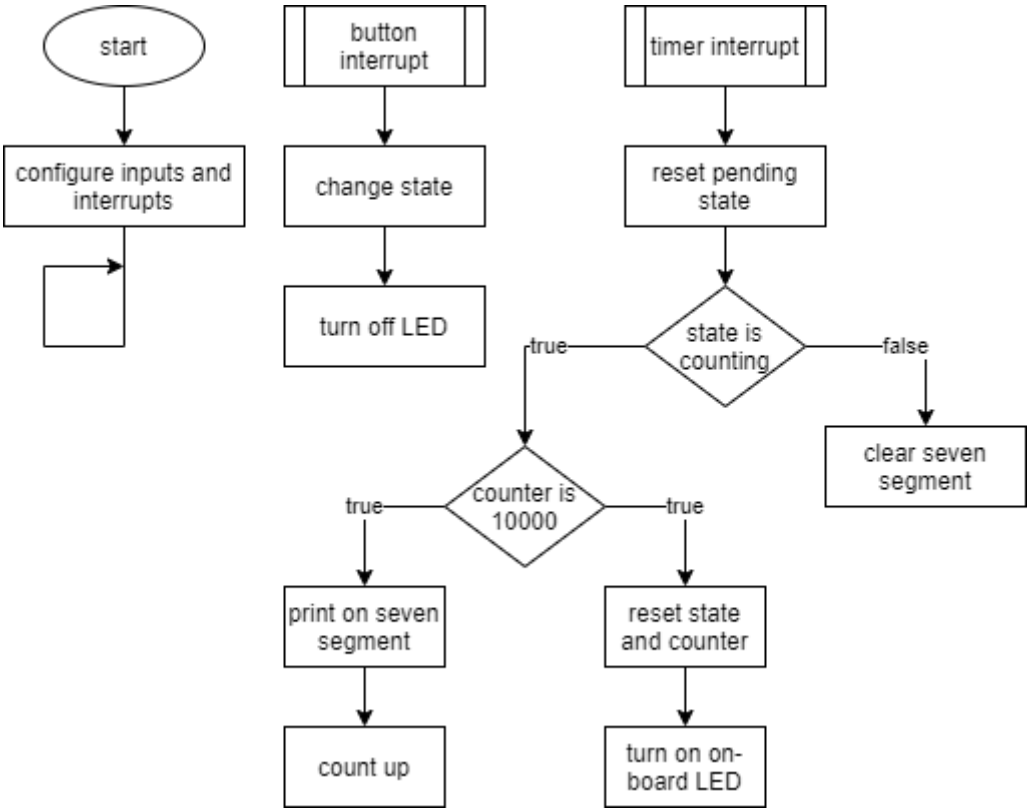
As we can see the timer interrupt executes every second in the beginning of the code.



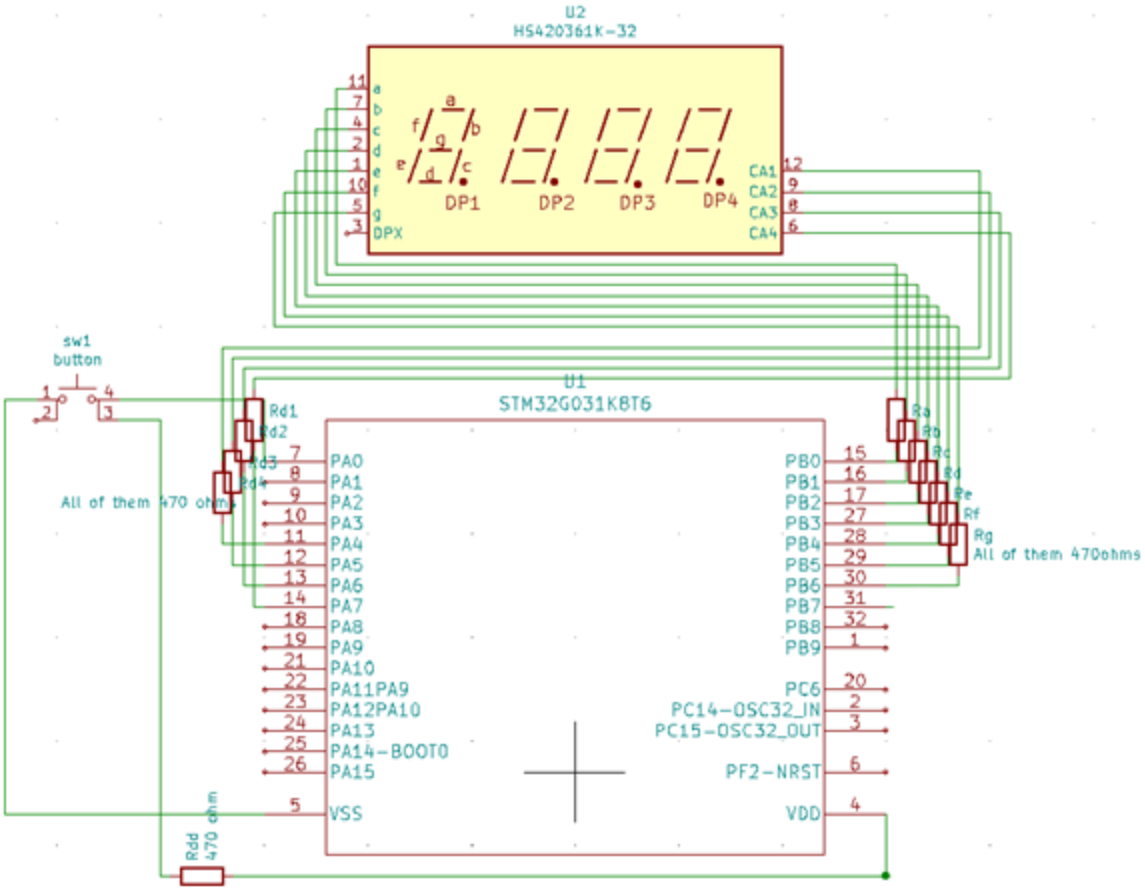
When the button is pressed the button interrupt routine is executed and the delay value is incremented or reset back.

4. Problem 3

In this problem we have to create a counter using timers and implement it on STM32 board. We can create a flowchart for code.



With the flowchart created we can choose the pins for seven segment and button. Then we can create the block diagram for the connections.



Now that the block diagram is created we can write the code for the problem using the flowchart and the pins decided on the block diagram.

```
#include "stm32g0xx.h"
#include "stdlib.h"

volatile uint32_t CurrentState = 0; /* 0 is idle, 1 is counting */
volatile uint32_t Counter = 0; /* counter value */

/* set seven segment functions */

void SetClear(void)
{
	GPIOB->ODR |= (0x7FU);
}

void Set(volatile uint32_t digit)
{
	switch(digit)
	{
		case 0:
			GPIOB->ODR &= (0x40U);
			break;
		case 1:
			GPIOB->ODR &= (0x79U);
			break;
		case 2:
			GPIOB->ODR &= (0x24U);
			break;
		case 3:
			GPIOB->ODR &= (0x30U);
			break;
		case 4:
			GPIOB->ODR &= (0x19U);
			break;
		case 5:
			GPIOB->ODR &= (0x12U);
			break;
		case 6:
			GPIOB->ODR &= (0x02U);
			break;
		case 7:
			GPIOB->ODR &= (0x78U);
			break;
		case 8:
			GPIOB->ODR &= (0x00U);
			break;
		case 9:
			GPIOB->ODR &= (0x10U);
			break;
	}
}

/* set digit function */

void SetDClear(void)
{
	GPIOA->ODR &= ~(15U << 4); /* digit clear */
}
```

```
void SetD(volatile uint32_t digit) /* 0 = Digit1 */
{
	SetDClear();
	SetClear();
	GPIOA->ODR |= (1U << (4 + digit));
}

/* print and refresh functions */

void print(void)
{
	uint32_t* digits = (uint32_t*) calloc(sizeof(uint32_t), 4); /* array to
store digits of counter */
	uint32_t counter = Counter; /* copy of the counter */

		*(digits+0) = (counter/1000); /* thousands digit extraction */
		counter -= (*(digits+0) * 1000);
		*(digits+1) = (counter/100); /* hundreds digit extraction */
		counter -= (*(digits+1) * 100);
		*(digits+2) = (counter/10); /* tens digit extraction */
		counter -= (*(digits+2) * 10);
		*(digits+3) = counter; /* ones digit extraction */

	for(uint32_t i=0; i<4; i++)
	{
		SetD(i);
		Set(*(digits+i));
	}
	SetDClear();
	free(digits);
}

/* handler functions */

void EXTI0_1_IRQHandler(void)
{
	EXTI->RPR1 |= (1U << 1); /* clear pending on A1 */
	CurrentState = 1; /* counting state */
	GPIOC->ODR &= ~(1U << 6); /* turn off on-board LED */
}

void TIM1_BRK_UP_TRG_COM_IRQHandler(void)
{
    TIM1->SR &= ~(1U << 0); /* reset status register */

    if(CurrentState) /* counting */
    {
      if(Counter == 10000)
      {
		CurrentState = 0; /* switch to idle */
		Counter = 0;
		GPIOC->ODR |= (1U << 6); /* turn on on-board LED */
      }
      else
      {
		for(int i=0; i<14; i++)
		{
			print(); /* print the global counter */
```

```
            }
            Counter++;
      }
    }
    else
    {
      /* idle state */
      SetClear();
    }
}

/* main function */

int main(void) {

      /* setting up the seven segment pins and on board LED */

      GPIOB->MODER |= (0x5555U); /* Setup (B0, B7) as output */
      GPIOB->MODER &= ~(0xAAAAU); /* Outputs for seven segments */

      GPIOA->MODER |= (0x55U << 2*4); /* Setup (A4, A7) as output */
      GPIOA->MODER &= ~(0xAAU << 2*4); /* Outputs for digit selections */

      GPIOC->MODER &= ~(2U << 2*6); /* Setup PC6 as output */
      GPIOC->MODER |= (1U << 2*6);

    /* setting up the button input and interrupt */
      __set_PRIMASK(0U); /* set PRIMASK to 0 */

      RCC->IOPENR |= (7U << 0); /* Enable GPIOA,B,C clocks */

      GPIOA->MODER &= ~(3U << 2*1); /* Setup PA1 as input */

      EXTI->RTSR1 |= (1U << 1); /* Rising edge selection A1 */
      EXTI->EXTICR[0] &= ~(1U << 8*1); /* 1U to select A1 from mux */
      EXTI->IMR1 |= (1U << 1); /* interrupt mask register A1 */

      EXTI->RPR1 |= (1U << 1); /* clear pending on A1 */

      NVIC_SetPriority(EXTI0_1_IRQn, 0x0U); /* Setting priority for EXTI0_1 */
      NVIC_EnableIRQ(EXTI0_1_IRQn); /* Enabling EXTI0_1 */

    /* setting up the timer and interrupt */

    RCC->APBENR2 |= (1U << 11); /* Enable TIM1 clock */

    TIM1->CR1 = 0; /* resetting control register */
    TIM1->CR1 |= (1U << 7); /* ARPE buffering */
    TIM1->CNT = 0; /* reset the timer counter */

    TIM1->PSC = 7999; /* prescaler set to 16000-1 */
    TIM1->ARR = 1; /* set the autoreload register for 1 milliseconds */

    TIM1->DIER |= (1U << 0); /* update interrupt enable */
    TIM1->CR1 |= (1U << 0); /* Enable TIM1 */

    NVIC_SetPriority(TIM1_BRK_UP_TRG_COM_IRQn, 0x3U); /* Setting priority for
TIM1 */
    NVIC_EnableIRQ(TIM1_BRK_UP_TRG_COM_IRQn); /* Enabling TIM1 */
```

```
    while(1)
    {

    }

    return 0;
}
```

The code works as follows. There is two global variables that are counter and current state. These variables will be used in the interrupts. The timer interrupt is executed every 1ms to countdown the counter variable and print it on the seven segment display. The button interrupt is to change the current status variable to counting and turns off the on-board LED. The for loop exists to create more refresh rate for it to be bright enough to read.

Print function works by extracting the digits off the counter and printing the digits individually on the seven segment display. This is done by dynamically allocating a digit array with 4 elements. Then this array is filled using integer division and multiplication. After the digits array is filled the print function uses the arrays in order to choose which function to use and uses them in a switch case condition.

Now we can test the code on STM32 IDE and check if it works.

```
 89  /* handler functions */
 90
 91  void EXTI0_1_IRQHandler(void)
 92  {
 93      EXTI->RPR1 |= (1U << 1); /* clear pending on A1 */
 94      CurrentState = 1; /* counting state */
 95      GPIOC->ODR &= ~(1U << 6); /* turn off on-board LED */
 96  }
```

As we can see the button interrupt works as intended when the button is pressed.

```
 98  void TIM1_BRK_UP_TRG_COM_IRQHandler(void)
 99  {
100      TIM1->SR &= ~(1U << 0); /* reset status register */
101
102      if(CurrentState) /* counting */
103      {
104          if(Counter == 10000)
105          {
106              CurrentState = 0; /* switch to idle */
107              Counter = 0;
108              GPIOC->ODR |= (1U << 6); /* turn on on-board LED */
109          }
110          else
111          {
112              for(int i=0; i<14; i++)
113              {
114                  print(); /* print the global counter */
```

Same as before the timer interrupt interrupts every 1 millisecond to create a counter.

5. Results and General Comments

In this lab we have worked with timers and timer interrupt routines. The timers we have used are: SysTick timer, external timer and watchdog timers. We used timers to create varying delays and counters.

We used systick timer to create a variable delay function that delays for a given value in milliseconds. After that we have created a timer that increases its delay with each button press and resets when the button is pressed and the delay is 10 seconds.

In the third problem we used an external timer to create a seven segment display up counter that counts up to the value 9999 and lights up an LED when its done. This up counting should be done each millisecond and should total 10 seconds. The main

6. References

[1]. ELEC 335 Lecture Slides (5-6-7).