GEBZE TECHNICAL UNIVERSITY

ELECTRONIC ENGINEERING

ELEC335 – MICROPROCESSORS LABAORATORY

LAB 4

HAZIRLAYANLAR
1801022035 – Ruveyda Dilara Günal
1801022071 – Alperen Arslan
1901022255 – Emirhan Köse

Problem 1

Create a Board Support Package (BSP) that you will be using for the rest of the semester. This BSP should only consist of board related functions. Some of these functions include:

- Configure / turn on / turn off / toggle on-board LED.
- Configure / read on-board button.
- Initialize and configure the processor clock.
- Initialize and configure the interrupts / exceptions.
- Initialize and configure the SysTick timer. (Problem 2)
- Initialize and configure the watchdog timer. (Problem 4)
- Initialize and configure the timers if any.
- Initialize and configure the external interrupts.

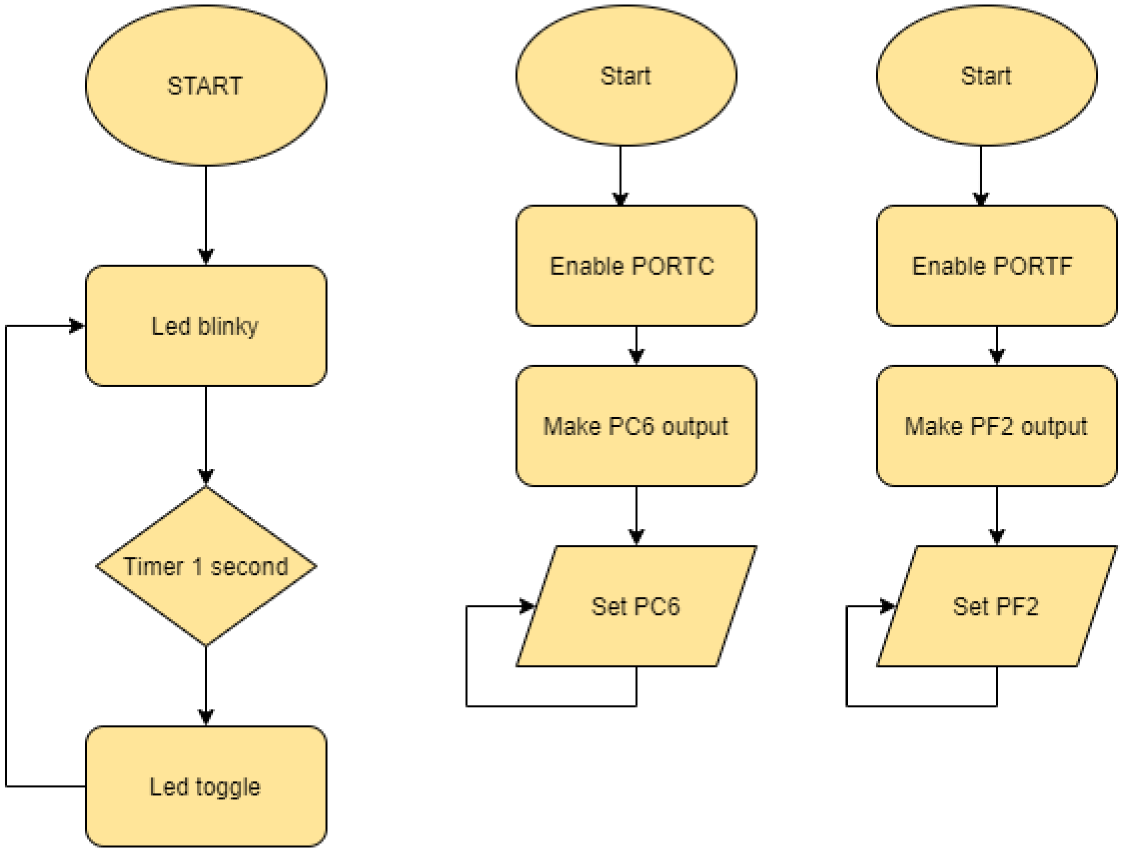

Figure 1: Problem 1 Flowchart

```
#include "bsp.h"
#include "stm32g0xx.h"

void BSP_onboardLed_init(){
```

```
	RCC->IOPENR |=(1U<<2); // PORT C is ENABLED
	GPIOC->MODER &=~(3U<<2*6);
	GPIOC->MODER |=(1U<<2*6); //PC6 IS ADJUSTED AS INPUT
}

void BSP_onboardLed_Toggle(){
	GPIOC->ODR ^=(1U<<6);
}
void BSP_onboardLed_on(){

	GPIOC->ODR |=(1U<<6);
}

void BSP_onboardLed_off(){

	GPIOC->ODR &=~(1U<<6);

}
void BSP_Delay(volatile unsigned int s){
	for (s; s>0 ;s--);
}

void BSP_onboardButton_init(){
RCC->IOPENR |=(1<<5);
GPIOF->MODER &=~(3U<<2*2);
}

int BSP_onbaordButton_read(){
	int ret_value=((GPIOF->IDR >>2) & 1);

	if(ret_value) return 0;
	else return 1;
}
void BSP_system_init(){
	//SystemCoreClockUpdate();
	BSP_onboardLed_init();
	BSP_onboardButton_init();
	//Systick_Config(SystemCoreClock/1000); //16M/1000 1 ms

}
void init_timer1(){

	RCC->APBENR2 |=(1U<<11); // enable tim1 module
	TIM1->CR1=0; //zero out the cotnrol register just in case
	TIM1->CR1 |=(1<<7);
	TIM1->CNT = 0; //zero out counter
	 //1 seconds interrupt
	TIM1->PSC = 999;
	TIM1->ARR = 16000;
```

```
	TIM1->DIER |=(1<<0);// update interrupt enable
	TIM1->CR1 |=(1<<0); //TIM 1 is ENABLED

	NVIC_EnableIRQ(TIM1_BRK_UP_TRG_COM_IRQn);
	NVIC_SetPriority(TIM1_BRK_UP_TRG_COM_IRQn, 1);

}
void TIM1_BRK_UP_TRG_COM_IRQHandler(void){
		BSP_onboardLed_Toggle();
		TIM1->SR &=~(1U<<0); //clear update status register.
}
void updateProcessorsClock(){

	SystemCoreClockUpdate();
}

__STATIC_INLINE uint32_t SysTick_Config(uint32_t ticks)
{
  if ((ticks - 1UL) > SysTick_LOAD_RELOAD_Msk)
  {
    return (1UL);
/* Reload value impossible */
  }

  SysTick->LOAD  = (uint32_t)(ticks - 1UL);
/* set reload register */
  NVIC_SetPriority (SysTick_IRQn, (1UL << __NVIC_PRIO_BITS) - 1UL);
/* set Priority for Systick Interrupt */
  SysTick->VAL   = 0UL;
/* Load the SysTick Counter Value */
  SysTick->CTRL  = SysTick_CTRL_CLKSOURCE_Msk |
                   SysTick_CTRL_TICKINT_Msk   |
                   SysTick_CTRL_ENABLE_Msk;
/* Enable SysTick IRQ and SysTick Timer */
  return (0UL);
/* Function successful */
}

void externalInterrupt_init(int port_number){

	EXTI->RTSR1 |=(1U<<port_number);
	if (0<=port_number<4){
		if(port_number==0)
			EXTI->EXTICR[0]=1U;
		else if(port_number==1)
			EXTI->EXTICR[0]=(1U<<8);
		else if(port_number==2)
			EXTI->EXTICR[0]=(1U<<2*8);
```

```
		else
			EXTI->EXTICR[0]=(1U<<3*8);
	}

	else if(4<=port_number<7){
		if(port_number==4)
			EXTI->EXTICR[1]=1U;
		else if(port_number==5)
			EXTI->EXTICR[1]=(1U<<8);
		else if(port_number==6)
			EXTI->EXTICR[1]=(1U<<2*8);
		else
			EXTI->EXTICR[1]=(1U<<3*8);
	}
	else if (7<=port_number<10){
		if(port_number==7)
			EXTI->EXTICR[2]=1U;
		else if(port_number==8)
			EXTI->EXTICR[2]=(1U<<8);
		else if(port_number==9)
			EXTI->EXTICR[2]=(1U<<2*8);
		else
			EXTI->EXTICR[2]=(1U<<3*8);
	}
	else if (10<=port_number<13){
			if(port_number==10)
				EXTI->EXTICR[3]=1U;
			else if(port_number==11)
				EXTI->EXTICR[3]=(1U<<8);
			else if(port_number==12)
				EXTI->EXTICR[3]=(1U<<2*8);
			else
				EXTI->EXTICR[3]=(1U<<3*8);
		}
	EXTI->IMR1 |=(1U<<port_number);
	if(0<=port_number<=1){
	NVIC_SetPriority(EXTI0_1_IRQn,0);
	NVIC_EnableIRQ(EXTI0_1_IRQn);
	}
	else if(2<=port_number<=3){
		NVIC_SetPriority(EXTI2_3_IRQn,0);
		NVIC_EnableIRQ(EXTI2_3_IRQn);
	}
	else if(3<port_number<=15){
		NVIC_SetPriority(EXTI4_15_IRQn,0);
		NVIC_EnableIRQ(EXTI4_15_IRQn);
	}
}
```

```c
/*
 * bsp.h
 *
 *  Created on: 30 Kas 2021
 *      Author: Lenovo
 */
#include "stm32g0xx.h"
#ifndef BSP_H_
#define BSP_H_


void BSP_onboardLed_init();
void BSP_onboardLed_Toggle();
void BSP_onboardLed_on();
void BSP_onboardLed_off();
void BSP_Delay(volatile unsigned int);

void BSP_onboardButton_init();
int BSP_onbaordButton_read();

void BSP_system_init();
void init_timer1();
void TIM1_BRK_UP_TRG_COM_IRQHandler(void);
__STATIC_INLINE uint32_t SysTick_Config(uint32_t ticks);
void updateProcessorsClock();

void externalInterrupt_init(int port_number);

#endif /* BSP_H_ */
```

Problem 2

In this problem, you will work on creating an accurate delay function using the **SysTick** exception. Create a SysTick exception with 1 millisecond interrupt intervals. Then create a delay_ms(..) function that will accurately wait for (blocking) a given number of milliseconds.

- Demonstrate the operation using an oscilloscope.
- Compare this approach to the without timer approach, explain the differences.

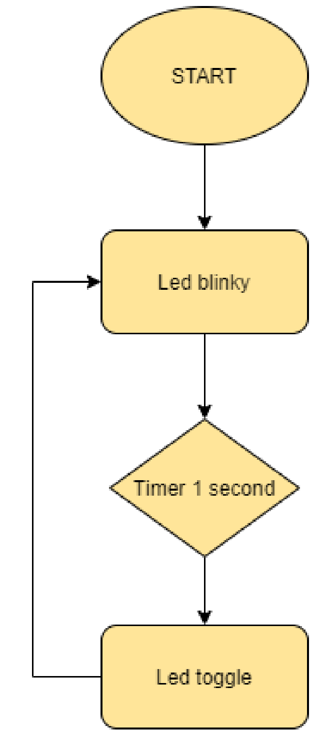

Figure 2: Problem 2 Flowchart

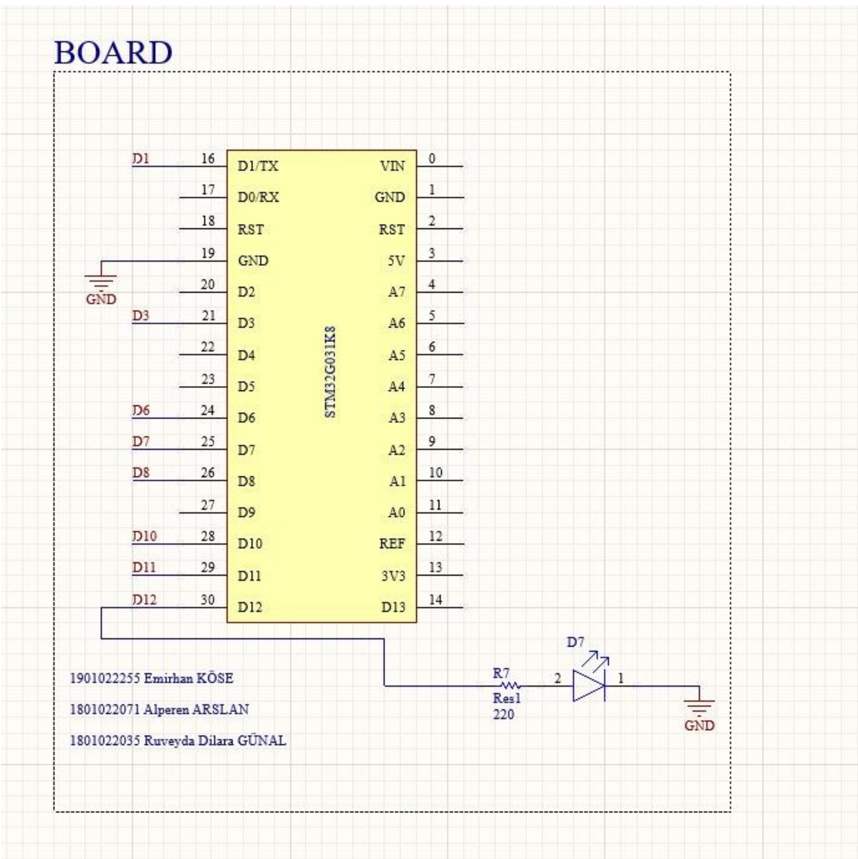

Figure 3: Problem 2 Block Diagram

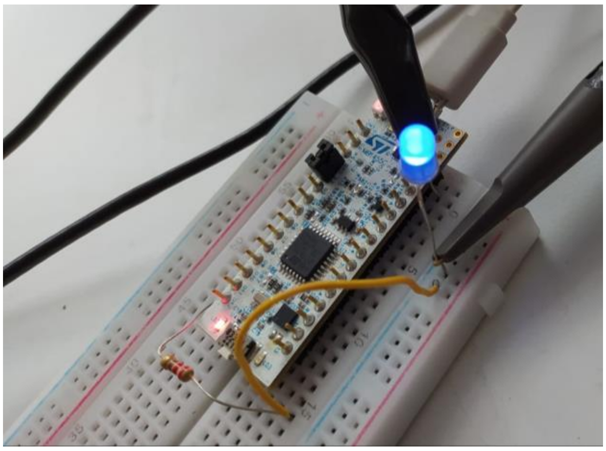

Figure 4: Problem 2 Circuit

```c
/*
 * main.c
 *
 * author: Alperen Arslan, Emirhan Köse, Ruveyda Dilara Günal

 */

#include "stm32g0xx.h"

void delay_ms(volatile unsigned int);


int main(void) {
SystemCoreClockUpdate();

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

    /* Setup PC6 as output */
    GPIOB->MODER &= ~(3U << 2*4);
    GPIOB->MODER |= (1U << 2*4);

    /* Turn on LED */
    GPIOB->ODR |= (1U << 4);

    int Start = SysTick->VAL;

    delay_ms(16000);

    int Stop = SysTick->VAL;

    unsigned int Delta = 0x00FFFFFF&(Start-Stop);
```

```
    while(1) {
        delay_ms(16000);
        /* Toggle LED */
        GPIOB->ODR ^= (1U << 4);
    }

    return 0;
}


void delay_ms(volatile unsigned int s){

     for(int i=s; i>0; i--){
      SysTick_Config(SystemCoreClock / 1000);
      }
}
```

```
#include "stm32g0xx.h"

void delay_ms(volatile unsigned int);


int main(void) {
SystemCoreClockUpdate();

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

    /* Setup PC6 as output */
    GPIOB->MODER &= ~(3U << 2*4);
    GPIOB->MODER |= (1U << 2*4);

    /* Turn on LED */
    GPIOB->ODR |= (1U << 4);



    while(1) {

     for(int s= 16000; s>0; s--){}
     GPIOB->ODR ^= (1U << 4);

    }

    return 0;
}
```

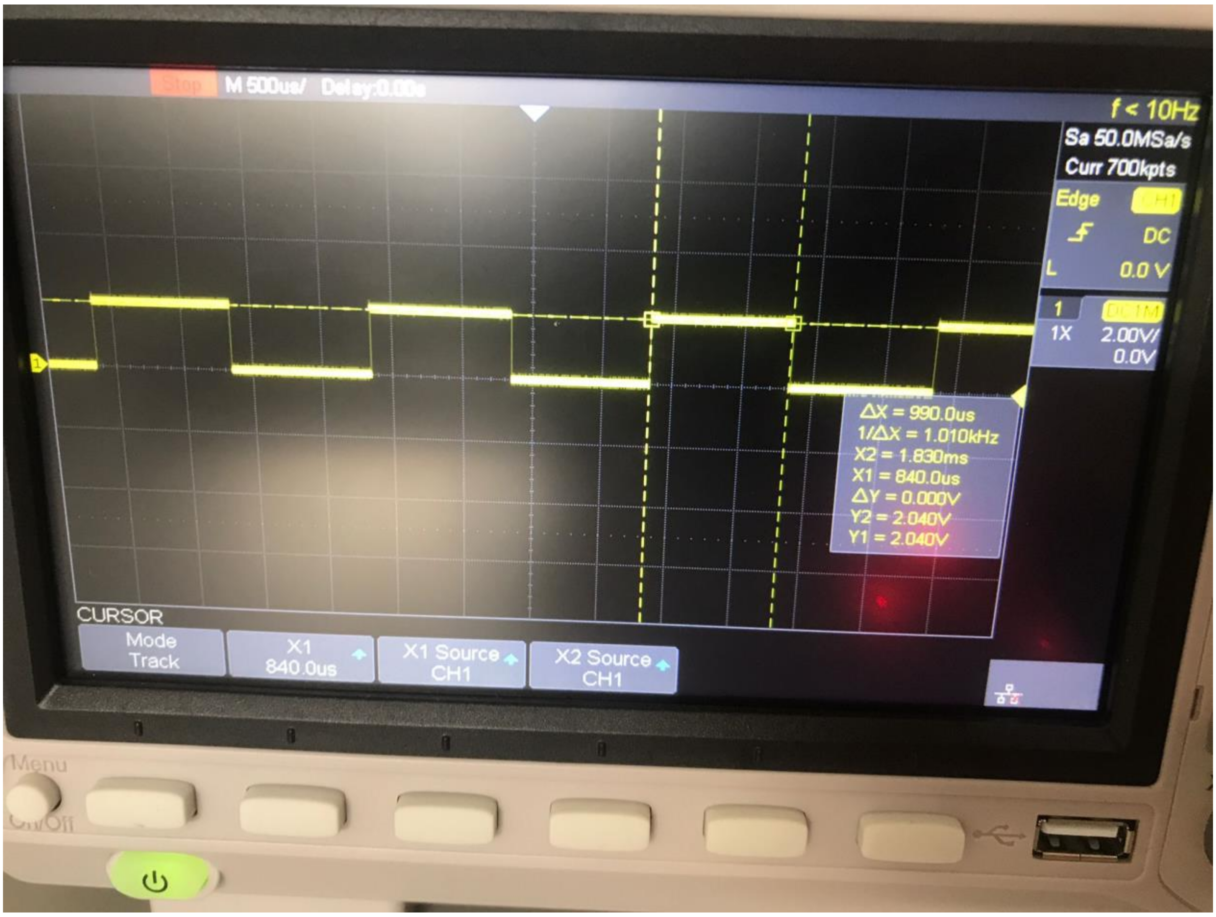

Figure 5: Oscilloscope Image

Problem 3

In this problem you are asked to implement a time counter using SSDs. Attach 4x SSDs and using **a state machine**, implement a time counter with different intervals. Assign each speed a mode and attach a button to cycle through the modes. (Each button press will cycle through these modes.)

- You should use a timer interrupt to display a given number on an SSD.
- Another timer should do the counting.
- This is the final implementation on how you should use the Seven Segment Display.

Modes:

- Mode 0 → No countdown (0 second interval)
- Mode 1 → Count down speed is ×1, with 1 second intervals.
- Mode 2 → Count down speed is ×2, with .5 second intervals
- Mode 3 → Count down speed is ×10, with .1 second intervals
- Mode 4 → Count down speed is ×100, with .01 second intervals
- Mode 5 → Count down speed is ×1000, with .001 second intervals

Warnings:

● Define an enum for your states, and cycle through them in each button press.

● Depending on the state, define the delay.

● Your flowchart should be detailed enough and should overlap with your implementation.

● Make sure your code works when the optimization is defined as -O2.

Requirements:

● Brightness on the Seven Segments should be equal.

● No bouncing on the buttons.

● Start from 00:00, 23:59, 23:58, ... and go down to 00:00 and keep counting again (free running time counter). The four SSDs must show the countdown as "**minute:second**" format without colon (**:**). For Mode 1, counting down must take 24 minutes to complete. For the other modes it should take less than 24 minutes, inversely proportional to the speed.

Questions:

● What is the difference in code size when the optimization is enabled / disabled? How about the actual counter speed? Is there any change? If so, what would be the difference?

● What is the code size percentage to the available ROM / RAM? How does optimization change this percentage?

● Is / Was there any brightness difference between the Seven Segments? If so, how did you fix it?

● Explain the difference between your implementation from the last lab? Which one is more convenient and scalable?

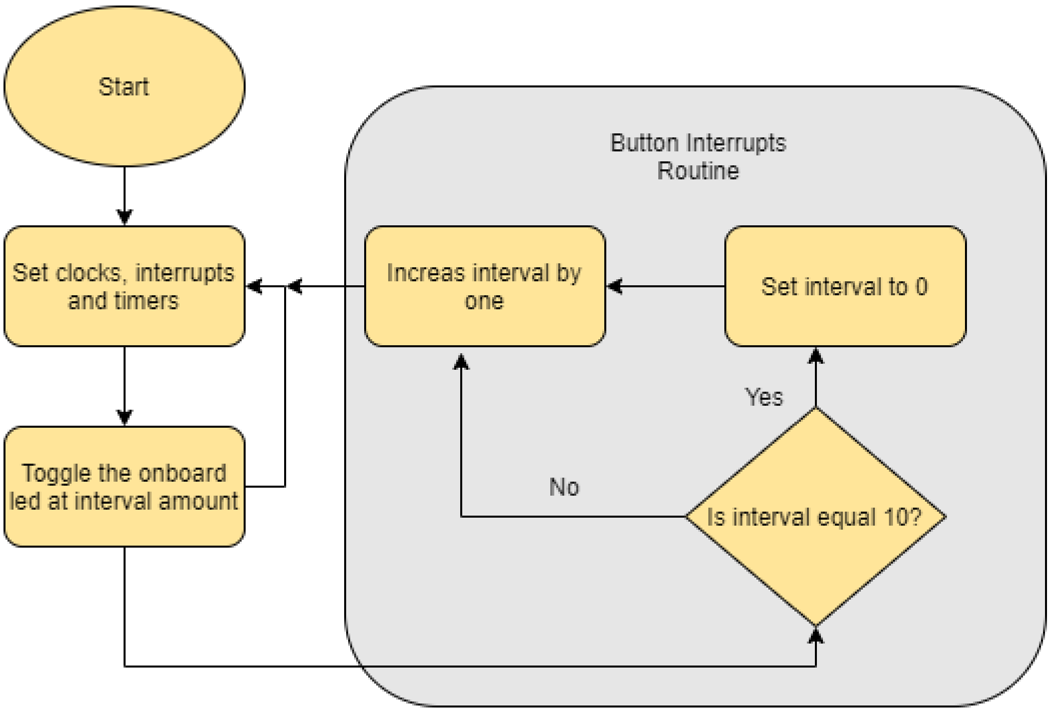

Figure 6: Problem 3 Flowchart

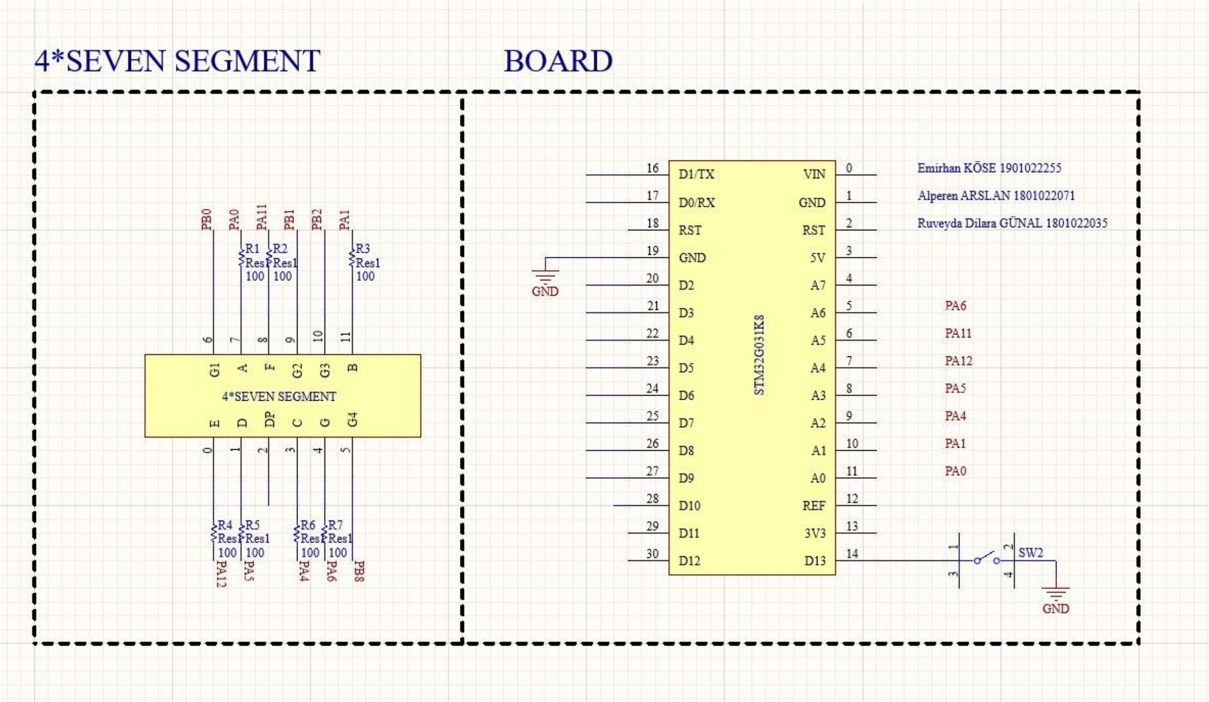

Figure 7: Problem 3 Block Diagram

```
/*
 * main.c
 *
 * author: Alperen Arslan, Emirhan Köse, Ruveyda Dilara Günal

 */
#include "stm32g0xx.h"
int i, j, k, l;
int count=0;

interval=1000000;

enum mode{m0,m1,m2,m3,m4,m5}state;


void EXTI2_3_IRQHandler(void) {

      if (state != 5) {
      state ++;
      }
      else {
      state = 0;
      }
      switch(state){
      case 0:
```

```
			delay(1000000);
			while(1){
			if((GPIOB->IDR >> 3) & 1){
				break;
			}
			run();
			}
		case 1:
			interval=1000000;
		break;
		case 2:
			interval=500000;
		break;
		case 3:
			interval=100000;
		break;
		case 4:
			interval=10000;
		break;
		case 5:
			interval=1000;
		break;
		}

EXTI->RPR1 |= (1U << 3);
}

void SysTick_Handler(void)  {

run();


   SysTick->VAL = 0;
}

int main(void) {
state=0;
	/*Enable GPIOA clock */
	RCC->IOPENR |= (1U << 0);
	/*Enable GPIOB clock */
	RCC->IOPENR |= (1U << 1);

	  /* setup PB(0,1,2,8) for seven segment D4,D3,D2,D1 for in
MODER */
		GPIOB->MODER &= ~(0x3003F);
		GPIOB->MODER |= (0x10015);

	/* Setup PA0, PA1, PA4, PA5, PA12, PA11, PA6 as output for
SSD */
```

```
      GPIOA->MODER &= ~(3U << 2*0);
      GPIOA->MODER |=  (1U << 2*0);  // PA0 is output

      GPIOA->MODER &= ~(3U << 2*1);
      GPIOA->MODER |=  (1U << 2*1);  // PA1 is output

    GPIOA->MODER &= ~(3U << 2*4);
      GPIOA->MODER |=  (1U << 2*4);  // PA4 is output

      GPIOA->MODER &= ~(3U << 2*5);
      GPIOA->MODER |=  (1U << 2*5);  // PA5 is output

      GPIOA->MODER &= ~(3U << 2*12);
      GPIOA->MODER |=  (1U << 2*12);  // PA12 is output

      GPIOA->MODER &= ~(3U << 2*11);
      GPIOA->MODER |=  (1U << 2*11);  // PA11 is output

      GPIOA->MODER &= ~(3U << 2*6);
      GPIOA->MODER |=  (1U << 2*6);  // PA6 is output
/* Setup PB3 as input */
      GPIOB->MODER &= ~(3U << 2*3);

      EXTI->RTSR1 |= (1U << 3); // B3
      EXTI->EXTICR[0] |= (1U << 8*3);
      EXTI->IMR1 |= (1U << 3);

      NVIC_SetPriority(EXTI2_3_IRQn, 0);
      NVIC_EnableIRQ(EXTI2_3_IRQn);

   SysTick_Config(SystemCoreClock/100);
int a;
      while(1) {
            clearSSD();
//initial values for 00:00
            i=0,j=0,k=0,l=0;
            delay(1000000);
            for(l=2;l>=0;l--){
                  delay(1000);
                  if(l==2){
                        a=3;
                  }else{
                        a=9;
                  }
            for(k=a;k>=0;k--){
                  delay(1000);
for(j=5;j>=0;j--){
delay(1000);
            for(i=9;i>=0;i--){
```

```
delay(interval);
		}
}
	}
	}
	}
	return 0;
}

void clearSSD(void){
	/* Set all outputs connected to SSD (clear SSD) */
	GPIOA->ODR |= (1U << 0);    // PA0  A
	GPIOA->ODR |= (1U << 1);    // PA1  B
	GPIOA->ODR |= (1U << 4);    // PA4  C
	GPIOA->ODR |= (1U << 5);    // PA5  D
	GPIOA->ODR |= (1U << 12);   // PA12 E
	GPIOA->ODR |= (1U << 11);   // PA11 F
	GPIOA->ODR |= (1U << 6);    // PA6  G
}
void setSSD( int x ) {
	clearSSD();
	switch ( x )
	{
	case 0:
		GPIOA->ODR &= ~(1U << 0);    // PA0  A
		GPIOA->ODR &= ~(1U << 1);    // PA1  B
		GPIOA->ODR &= ~(1U << 4);    // PA4  C
		GPIOA->ODR &= ~(1U << 5);    // PA5  D
		GPIOA->ODR &= ~(1U << 12);   // PA12 E
		GPIOA->ODR &= ~(1U << 11);   // PA11 F
		break;

	case 1:
		GPIOA->ODR &= ~(1U << 1);    // PA1  B
		GPIOA->ODR &= ~(1U << 4);    // PA4  C
		break;

	case 2:
		GPIOA->ODR &= ~(1U << 0);    // PA0  A
		GPIOA->ODR &= ~(1U << 1);    // PA1  B
		GPIOA->ODR &= ~(1U << 5);    // PA5  D
		GPIOA->ODR &= ~(1U << 12);   // PA12 E
		GPIOA->ODR &= ~(1U << 6);    // PA6  G
		break;

	case 3:
		GPIOA->ODR &= ~(1U << 0);    // PA0  A
		GPIOA->ODR &= ~(1U << 1);    // PA1  B
		GPIOA->ODR &= ~(1U << 4);    // PA4  C
```

```c
        GPIOA->ODR &= ~(1U << 5);    // PA5  D
        GPIOA->ODR &= ~(1U << 6);    // PA6  G
        break;

    case 4:
        GPIOA->ODR &= ~(1U << 1);    // PA1  B
        GPIOA->ODR &= ~(1U << 4);    // PA4  C
        GPIOA->ODR &= ~(1U << 11);   // PA11 F
        GPIOA->ODR &= ~(1U << 6);    // PA6  G
        break;

    case 5:
        GPIOA->ODR &= ~(1U << 0);    // PA0  A
        GPIOA->ODR &= ~(1U << 4);    // PA4  C
        GPIOA->ODR &= ~(1U << 5);    // PA5  D
        GPIOA->ODR &= ~(1U << 11);   // PA11 F
        GPIOA->ODR &= ~(1U << 6);    // PA6  G
        break;

    case 6:
        GPIOA->ODR &= ~(1U << 0);    // PA0  A
        GPIOA->ODR &= ~(1U << 4);    // PA4  C
        GPIOA->ODR &= ~(1U << 5);    // PA5  D
        GPIOA->ODR &= ~(1U << 12);   // PA12 E
        GPIOA->ODR &= ~(1U << 11);   // PA11 F
        GPIOA->ODR &= ~(1U << 6);    // PA6  G
        break;

    case 7:
        GPIOA->ODR &= ~(1U << 0);    // PA0  A
        GPIOA->ODR &= ~(1U << 1);    // PA1  B
        GPIOA->ODR &= ~(1U << 4);    // PA4  C
        break;

    case 8:
        GPIOA->ODR &= ~(1U << 0);    // PA0  A
        GPIOA->ODR &= ~(1U << 1);    // PA1  B
        GPIOA->ODR &= ~(1U << 4);    // PA4  C
        GPIOA->ODR &= ~(1U << 5);    // PA5  D
        GPIOA->ODR &= ~(1U << 12);   // PA12 E
        GPIOA->ODR &= ~(1U << 11);   // PA11 F
        GPIOA->ODR &= ~(1U << 6);    // PA6  G
        break;

    case 9:
        GPIOA->ODR &= ~(1U << 0);    // PA0  A
        GPIOA->ODR &= ~(1U << 1);    // PA1  B
        GPIOA->ODR &= ~(1U << 4);    // PA4  C
        GPIOA->ODR &= ~(1U << 5);    // PA5  D
```

```
            GPIOA->ODR &= ~(1U << 11);  // PA11 F
            GPIOA->ODR &= ~(1U << 6);   // PA6  G
            break;
      }
}

void on_SSD1() {/* turn on SSD 1(LEFT).*/
       /* turn on ODR*/
      GPIOB->ODR |= (0x100);
}
void off_SSD1() {/* turn off SSD 1(LEFT).*/
       /* turn on ODR*/
      GPIOB->BRR |= (0x100);
}
void on_SSD2() {/* turn on SSD 2.*/
       /* turn on ODR*/
      GPIOB->ODR |= (0x4);
}
void off_SSD2() {/* turn off SSD 2.*/
       /* turn on ODR*/
      GPIOB->BRR |= (0x4);
}
void on_SSD3() {/* turn on SSD 3.*/
       /* turn on ODR*/
      GPIOB->ODR |= (0x1);
}
void off_SSD3() {/* turn off SSD 3.*/
       /* turn on ODR*/
      GPIOB->BRR |= (0x1);
}
void on_SSD4() {/* turn on SSD 4.*/
       /* turn on ODR*/
      GPIOB->ODR |= (0x2);
}
void off_SSD4() {/* turn off SSD 4.*/
       /* turn on ODR*/
      GPIOB->BRR |= (0x2);
}

void delay_ms(volatile uint32_t s) {
    for(; s>0; s--);
}

void delay(volatile uint32_t s) {
    for(; s>0; s--);
}

void run(){
              on_SSD1();
```

```
              setSSD(l);
              delay(3250);
              off_SSD1();
              delay(100);

              on_SSD2();
              setSSD(k);
              delay(3250);
              off_SSD2();
              delay(100);

              on_SSD3();
              setSSD(j);
              delay(3250);
              off_SSD3();
              delay(100);

              on_SSD4();
              setSSD(i);
              delay(3250);
              off_SSD4();
              delay(100);
}
```

Problem 4

In this problem, you will work with watchdog timers. Take Problem 3 as base, and implement a window or independent watchdog timer with appropriate delay. The handler routine should restore the stack pointer and reset back the state of the program to the beginning.

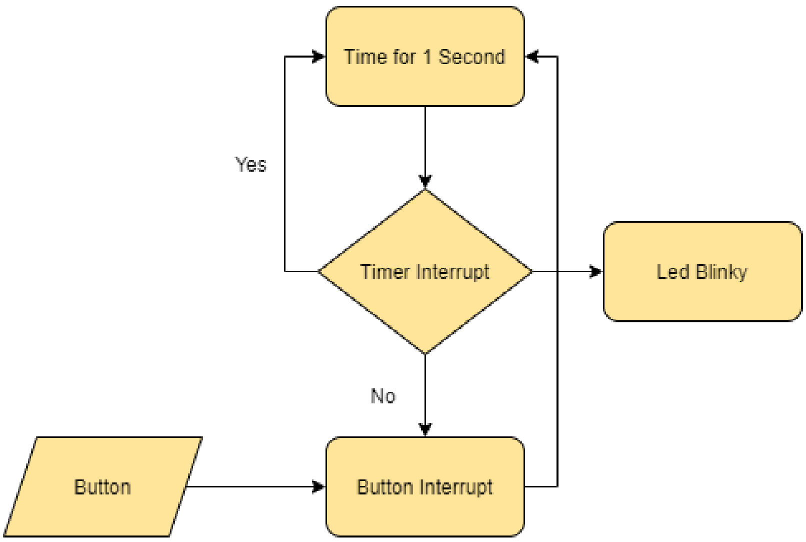

Figure 8: Problem 4 Flowchart

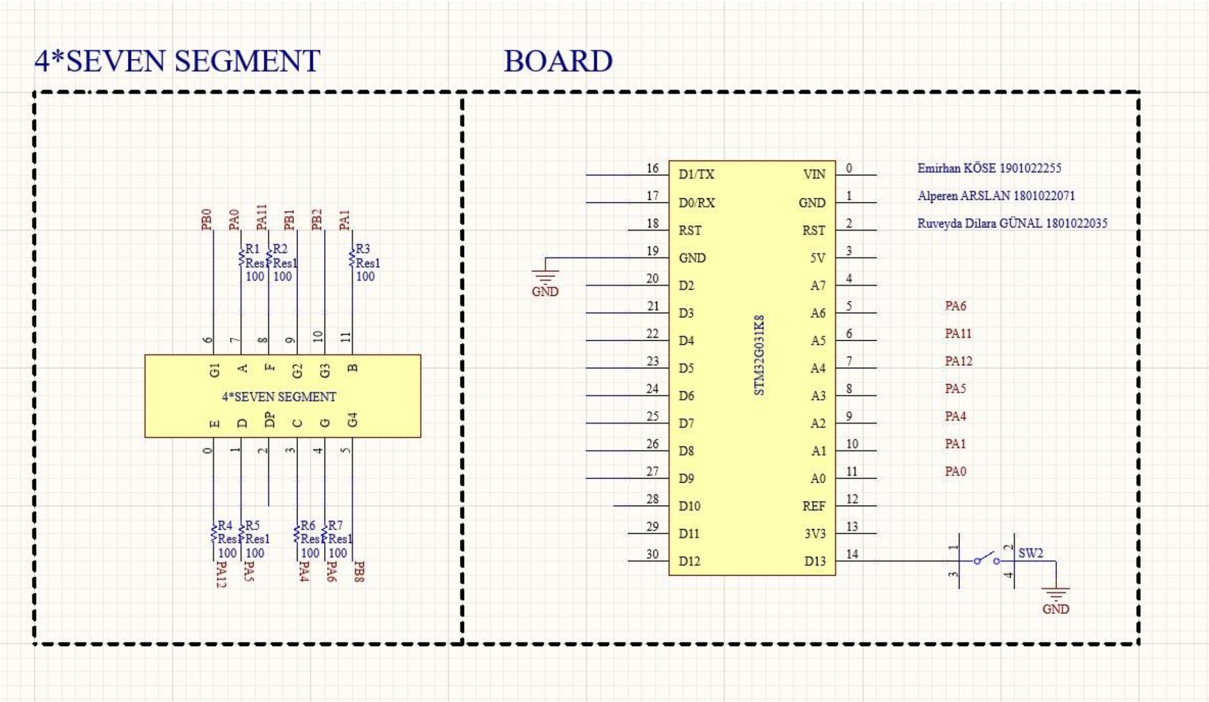

Figure 9: Problem 4 Block Diagram

```c
/*
* main.c
*
* author: Alperen Arslan, Emirhan Köse, Ruveyda Dilara Günal
*
*/
#include "stm32g0xx.h"
#define LEDDELAY 160000
#define delayled 16000000
int main(void);
void turn_off(void);
void init_wd(void);
void button(void);
void delay(volatile uint32_t);
void EXTI0_1_IRQHandler(void){
if (EXTI->RPR1 & (1U << 1)){
while(1) {
delay(LEDDELAY);
/* Toggle LED */
GPIOC->ODR ^= (1U << 6);
}
EXTI->RPR1 |= (1U << 1);
}
}
void NonMaskableInt_IRQHandler(void){
turn_off();
```

```
}
int main(void) {
/* Enable GPIOC clock */
RCC->IOPENR |= (1U << 2);
RCC->IOPENR |= (1U << 0);
/* Setup PC6 as output */
GPIOC->MODER &= ~(3U << 2*6);
GPIOC->MODER |= (1U << 2*6);
/* Setup PA1 as input */
GPIOA->MODER &= ~(3U << 2*1);
GPIOA->PUPDR |= (2U << 2*1); // Pull-down mode
/* Turn on LED */
GPIOC->ODR |= (1U << 6);
/*setup interrrupts for inputs*/
EXTI->EXTICR[0] |=(0U << 8*1);//PA1
/* MASK*/
EXTI->IMR1 |= (1U << 1);
/*rising edge*/
EXTI->RTSR1 |= (1U << 1);
/*NVIC*/
NVIC_SetPriority(EXTI0_1_IRQn, 0);
NVIC_EnableIRQ(EXTI0_1_IRQn);
init_wd();
while(1) {
}
return 0;
}
void init_wd(void){
RCC->CSR |= (3U << 0);
IWDG->KR = 0xAAAA;

while (IWDG->SR != 0) { }
IWDG->KR = 0x5555; // enable access to the e IWDG_PR, IWDG_RLR
IWDG->PR = 6;
IWDG->RLR = 0x0AA; //watchdog counter each time the value
IWDG->WINR= 0x0AA; // access protected
IWDG->KR = 0xAAAA;
IWDG->KR = 0xCCCC; //Starts if wasn't running yet
NVIC_SetPriority(NonMaskableInt_IRQn, 3);
NVIC_EnableIRQ(NonMaskableInt_IRQn);
}
void turn_off(void){
/* Turn off LED */
GPIOC->ODR |= (0U << 6);
delay(delayled);
main();
}
void delay(volatile uint32_t s) {
for(; s>0; s--);
```

```
}
void button(void){
}
```