ELECTRONICS ENGINEERING

ELEC335 - MICROPROCESSORS LABORATORY

LAB #4

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.

CODE:

```
//problem 1 bsp.h file
#ifndef BSP_H
#define BSP_H

void BSP_IO_pin_init(volatile uint32_t port, volatile uint32_t pin,
volatile uint32_t mode, volatile uint32_t pullup_pulldown);
void BSP_IO_pin_write(volatile uint32_t port, volatile uint32_t pin,
volatile uint32_t write);
uint32_t BSP_IO_pin_read(volatile uint32_t port, volatile uint32_t pin);
void BSP_EXTI_init(volatile uint32_t port, volatile uint32_t pin,
volatile uint32_t redge_fedge_mode, volatile uint32_t mask);
void BSP_SysTick_init(volatile uint32_t number);
void BSP_TIMx_init(volatile uint32_t tim_value, volatile uint32_t
counter_value, volatile uint32_t psc_value, volatile uint32_t
up_down_count);
void BSP_IWDG_init();
void BSP_IWDG_feed();

#endif

void BSP_IO_pin_init(volatile uint32_t port, volatile uint32_t pin,
volatile uint32_t mode, volatile uint32_t pullup_pulldown){
	RCC->IOPENR |= (1U << port);
	switch(port){
		case 0:
			GPIOA->MODER &= ~(3U << 2*pin);
			GPIOA->MODER |= (mode << 2*pin);
			GPIOA->PUPDR |= (mode == 0) ? (pullup_pulldown <<
2*pin) : GPIOA->PUPDR;
			break;
		case 1:
			GPIOB->MODER &= ~(3U << 2*pin);
			GPIOB->MODER |= (mode << 2*pin);
			GPIOB->PUPDR |= (mode == 0) ? (pullup_pulldown <<
2*pin) : GPIOB->PUPDR;
			break;
		case 2:
			GPIOC->MODER &= ~(3U << 2*pin);
			GPIOC->MODER |= (mode << 2*pin);
```

```
				GPIOC->PUPDR |= (mode == 0) ? (pullup_pulldown <<
2*pin) : GPIOC->PUPDR;
				break;
			case 3:
				GPIOD->MODER &= ~(3U << 2*pin);
				GPIOD->MODER |= (mode << 2*pin);
				GPIOD->PUPDR |= (mode == 0) ? (pullup_pulldown <<
2*pin) : GPIOD->PUPDR;
				break;
			/*case 4:
				GPIOE->MODER &= ~(3U << 2*pin);
				GPIOE->MODER |= (mode << 2*pin);
				GPIOE->PUPDR |= (mode && 0) ? (pullup_pulldown <<
2*pin) : GPIOE->PUPDR;
				break;*/
			case 5:
				GPIOF->MODER &= ~(3U << 2*pin);
				GPIOF->MODER |= (mode << 2*pin);
				GPIOF->PUPDR |= (mode == 0) ? (pullup_pulldown <<
2*pin) : GPIOF->PUPDR;
				break;
		}
}

void BSP_IO_pin_write(volatile uint32_t port, volatile uint32_t pin,
volatile uint32_t write){
		switch(port){
			case 0:
				switch(write){
					case 0:
						GPIOA->ODR &= ~(1U << pin);
						break;
					case 1:
						GPIOA->ODR |= (1U << pin);
						break;
					case 2:
						GPIOA->ODR ^= (1U << pin);
						break;
				}
				break;
			case 1:
				switch(write){
					case 0:
						GPIOB->ODR &= ~(1U << pin);
						break;
					case 1:
						GPIOB->ODR |= (1U << pin);
						break;
					case 2:
						GPIOB->ODR ^= (1U << pin);
						break;
				}
				break;
		case 2:
			switch(write){
				case 0:
					GPIOC->ODR &= ~(1U << pin);
					break;
				case 1:
					GPIOC->ODR |= (1U << pin);
					break;
				case 2:
					GPIOC->ODR ^= (1U << pin);
					break;
			}
			break;
		case 3:
			switch(write){
				case 0:
					GPIOD->ODR &= ~(1U << pin);
					break;
				case 1:
					GPIOD->ODR |= (1U << pin);
					break;
				case 2:
					GPIOD->ODR ^= (1U << pin);
					break;
			}
			break;
		/*case 4:
			switch(write){
				case 0:
					GPIOE->ODR &= ~(1U << pin);
					break;
				case 1:
					GPIOE->ODR |= (1U << pin);
					break;
				case 2:
					GPIOE->ODR ^= (1U << pin);
					break;
			}
			break;*/
		case 5:
			switch(write){
				case 0:
					GPIOF->ODR &= ~(1U << pin);
					break;
				case 1:
					GPIOF->ODR |= (1U << pin);
					break;
				case 2:
					GPIOF->ODR ^= (1U << pin);
					break;
			}
			break;
	}
```

```c
}

uint32_t BSP_IO_pin_read(volatile uint32_t port, volatile uint32_t pin){
	switch(port){
		case 0:
			return (GPIOA->IDR >> pin) & 1;
			break;
		case 1:
			return (GPIOB->IDR >> pin) & 1;
			break;
		case 2:
			return (GPIOC->IDR >> pin) & 1;
			break;
		case 3:
			return (GPIOD->IDR >> pin) & 1;
			break;
		/*case 4:
			return (GPIOE->IDR >> pin) & 1;
			break;*/
		case 5:
			return (GPIOF->IDR >> pin) & 1;
			break;
	}
	return 0xFFFFFFFF;
}

void BSP_EXTI_init(volatile uint32_t port, volatile uint32_t pin,
volatile uint32_t redge_fedge_mode, volatile uint32_t mask){
	volatile uint32_t exticr_num;
	if((pin <= 3)){
		exticr_num = 0;
	}
	else if((pin >= 4) && (pin <= 7)){
		exticr_num = 1;
	}
	else if((pin >= 8) && (pin <= 11)){
		exticr_num = 2;
	}
	else if((pin >= 12) && (pin <= 15)){
		exticr_num = 3;
	}

	EXTI->EXTICR[exticr_num] |= (port << 8*(pin % 4));
	if(redge_fedge_mode == 1){
		EXTI->RTSR1 |= (1U << pin);
	}
	else if(redge_fedge_mode == 0){
		EXTI->FTSR1 |= (1U << pin);
	}
	EXTI->IMR1 = (mask == 1) ? (EXTI->IMR1 | (1U << pin)) : ((mask ==
0) ? (EXTI->IMR1 & ~(1U << pin)) : EXTI->IMR1);
```

```
      if((pin <= 1)){
            NVIC_EnableIRQ(EXTI0_1_IRQn);
      }
      else if((pin >= 2) && (pin <= 3)){
            NVIC_EnableIRQ(EXTI2_3_IRQn);
      }
      else if((pin >= 4) && (pin <= 15)){
            NVIC_EnableIRQ(EXTI4_15_IRQn);
      }
}

void BSP_SysTick_init(volatile uint32_t number){
      SysTick_Config(number*SystemCoreClock/1000);
      NVIC_EnableIRQ(SysTick_IRQn);

}
void BSP_TIMx_init(volatile uint32_t tim_value, volatile uint32_t
counter_value, volatile uint32_t psc_value, volatile uint32_t
up_down_count){
      switch(tim_value){
            case 1:
                  RCC->APBENR2 |= (1U << 11);

                  TIM1->CR1 = 0;
                  TIM1->CR1 |= (1U << 7);
                  TIM1->CR1 = up_down_count == 0 ? (TIM1->CR1 & ~(1U <<
4)) : (up_down_count == 1 ? (TIM1->CR1 | (1U << 4)) : TIM1->CR1);
                  TIM1->CNT = 0;

                  TIM1->PSC = psc_value;
                  TIM1->ARR = counter_value;

                  TIM1->DIER |= (1U << 0);
                  TIM1->CR1 |= (1U << 0);

                  NVIC_EnableIRQ(TIM1_BRK_UP_TRG_COM_IRQn);

                  break;
            case 2:
                  RCC->APBENR1 |= (1U << 0);

                  TIM2->CR1 = 0;
                  TIM2->CR1 |= (1U << 7);
                  TIM2->CR1 = up_down_count == 0 ? (TIM2->CR1 & ~(1U <<
4)) : (up_down_count == 1 ? (TIM2->CR1 | (1U << 4)) : TIM2->CR1);
                  TIM2->CNT = 0;

                  TIM2->PSC = psc_value;
                  TIM2->ARR = counter_value;

                  TIM2->DIER |= (1U << 0);
                  TIM2->CR1 |= (1U << 0);
                  NVIC_EnableIRQ(TIM2_IRQn);

                  break;
            case 3:
                  RCC->APBENR1 |= (1U << 1);

                  TIM3->CR1 = 0;
                  TIM3->CR1 |= (1U << 7);
                  TIM3->CR1 = up_down_count == 0 ? (TIM3->CR1 & ~(1U <<
4)) : (up_down_count == 1 ? (TIM3->CR1 | (1U << 4)) : TIM3->CR1);
                  TIM3->CNT = 0;

                  TIM3->PSC = psc_value;
                  TIM3->ARR = counter_value;

                  TIM3->DIER |= (1U << 0);
                  TIM3->CR1 |= (1U << 0);

                  NVIC_EnableIRQ(TIM3_IRQn);

                  break;
      }
}

void BSP_IWDG_init(){
      IWDG->KR = 0xCCCC;
}
void BSP_IWDG_feed(){
      IWDG->KR = 0xAAAA;
}
```

PROBLEM 2) In this problem, you will work on creating an accurate delay function using the SysTick exception. Create a SysTick exception with 1 milisecond interrupt intervals. Then create a delay_ms(..) function that will accurately wait for (blocking) a given number of milliseconds.

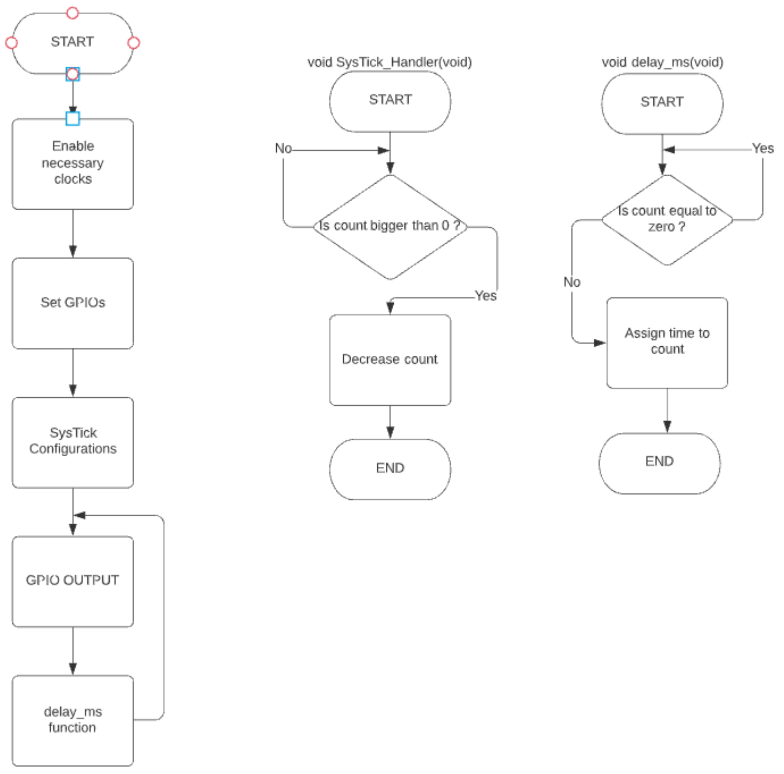


Figure 2.1) Flowchart of the problem 2

● Demonstrate the operation using an oscilloscope.

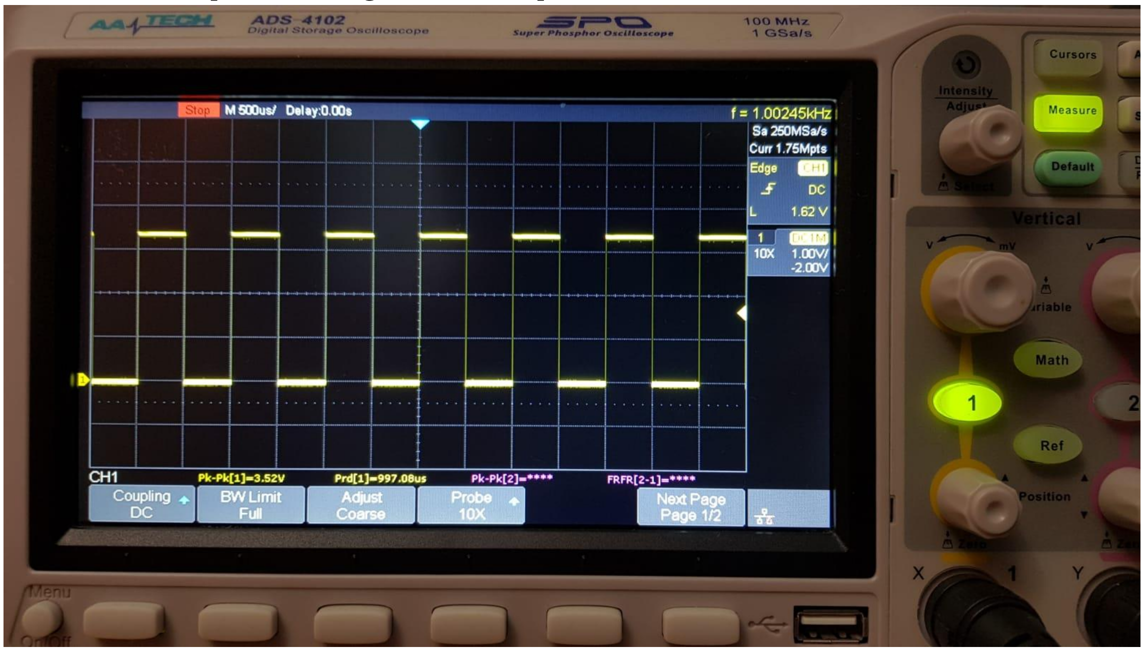


Figure 2.2) Timer Interrupt delay oscilloscope display (optimization -O0 and -O2) (period 997.08 us)

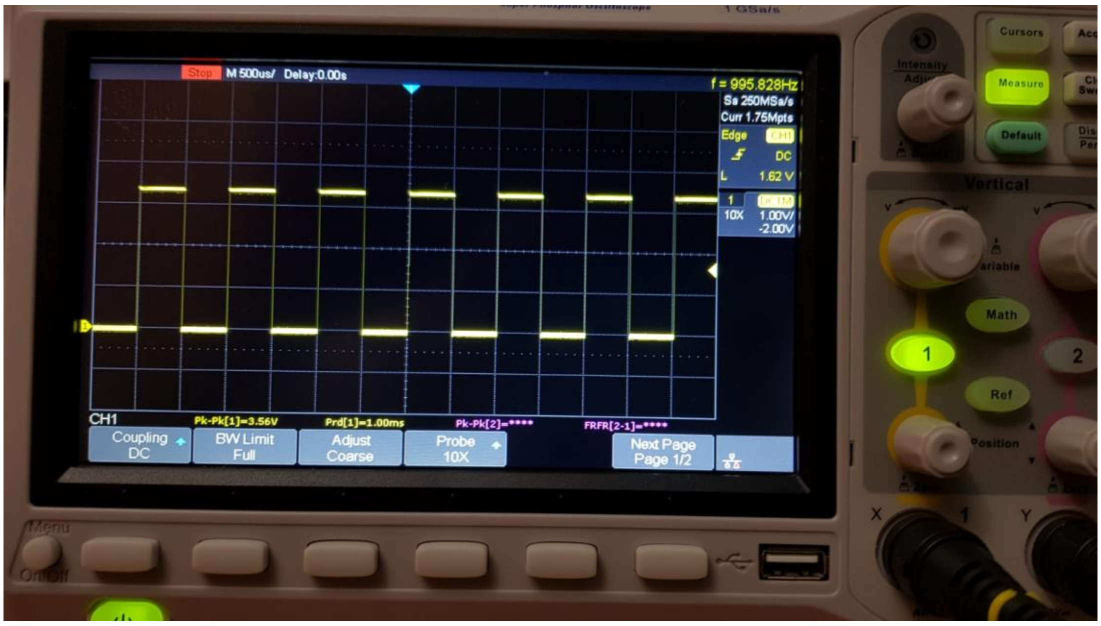


Figure 2.3 In main function delay oscilloscope display (optimization -O0) (period 1ms)

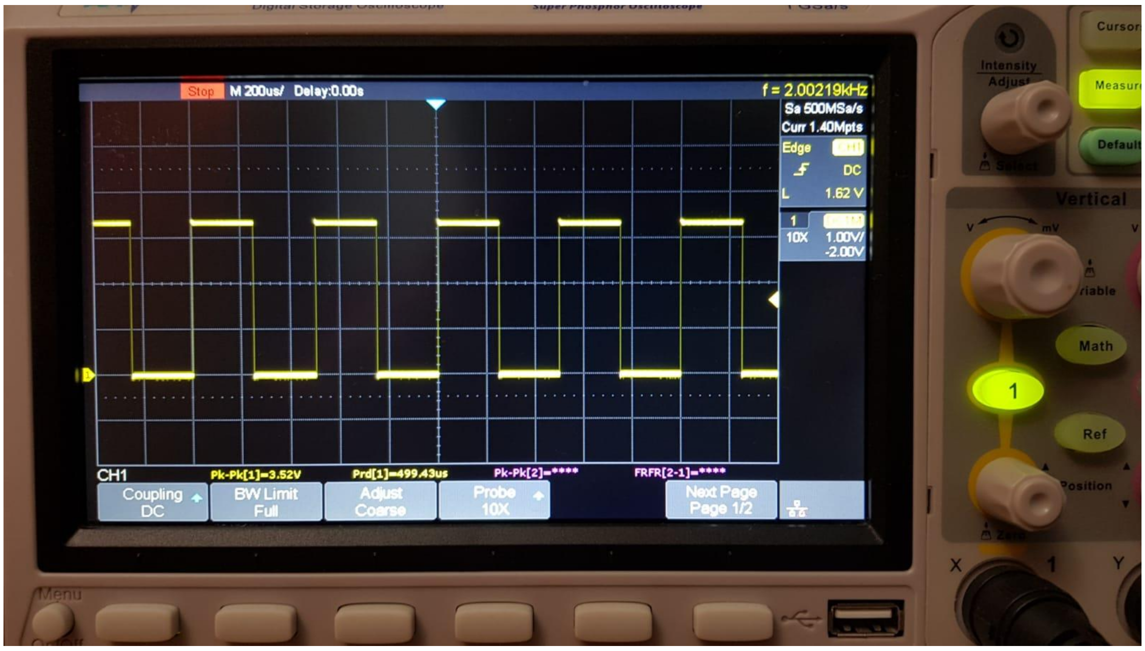


Figure 2.4) In main function delay oscilloscope display (optimization -O2) (period 499.43 us)

● Compare this approach to the without timer approach, explain the differences.
→ Both delay functions work correctly. But if we have more processes the delay function without using timer will not work correctly.

CODE:

```
// problem 2
#include "stm32g0xx.h"

#define LEDDELAY (800000/1000) //1ms
volatile uint32_t count;
```

```
void delay_ms(uint32_t time){
      count = time;
      while(count);
}

void SysTick_Handler(void){
      if(count > 0)
            count--;
}
void delay(volatile uint32_t s);
int main(void) {

      //init_timer1();
    /* Enable GPIOC clock */
    RCC->IOPENR |= (1 << 2);

    /* Setup PC6 as output */
    GPIOC->MODER &= ~(3U << 2*6);
    GPIOC->MODER |= (1 << 2*6);

    //(SystemCoreClock=16000000) / 100000 => 10us
    SysTick_Config(SystemCoreClock / 100000);

    while(1) {
        /* Toggle LED */
        GPIOC->ODR ^= (1 << 6);
        delay_ms(50); // 1ms
        //delay(LEDDELAY);
    }

    return 0;
}

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

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.)

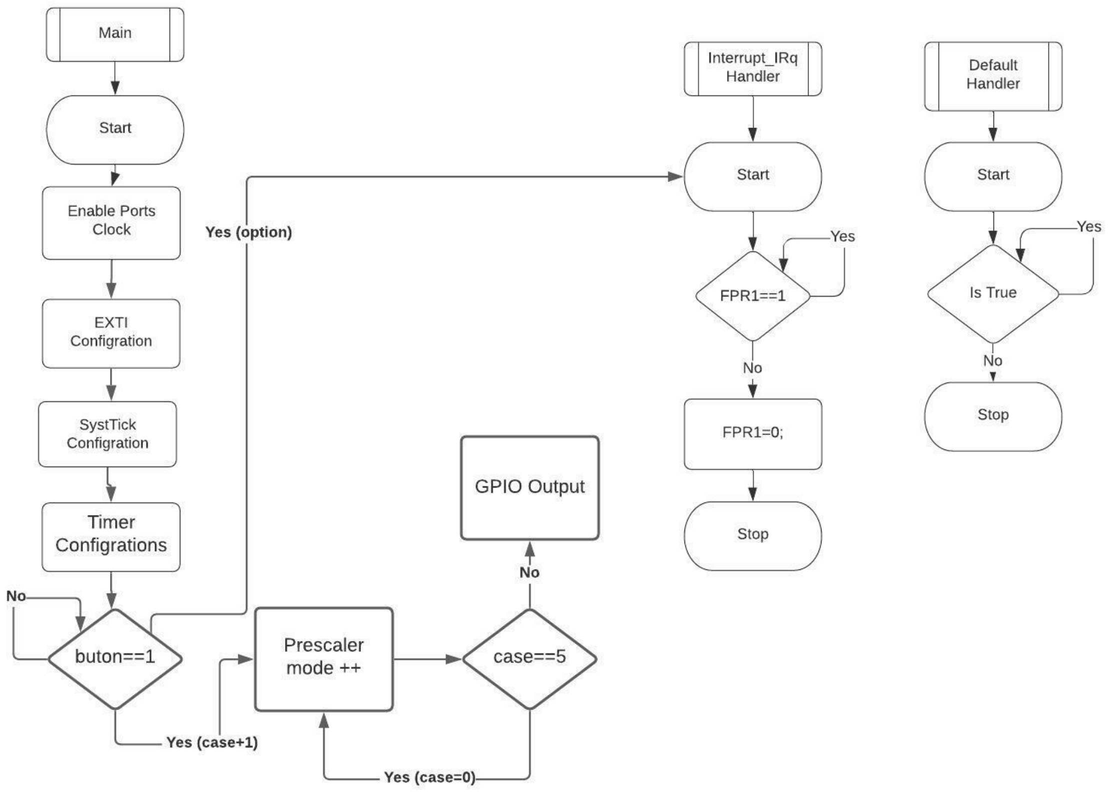


Figure 3.1) Flowchart of the problem 3

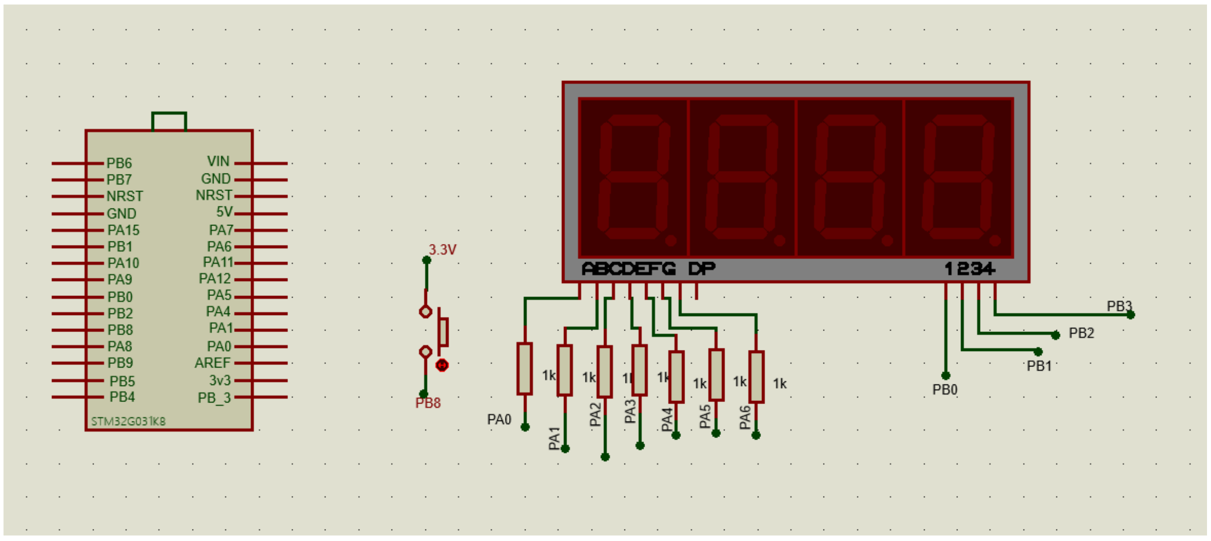


Figure 3.2) Circuit connection diagram of problem 3

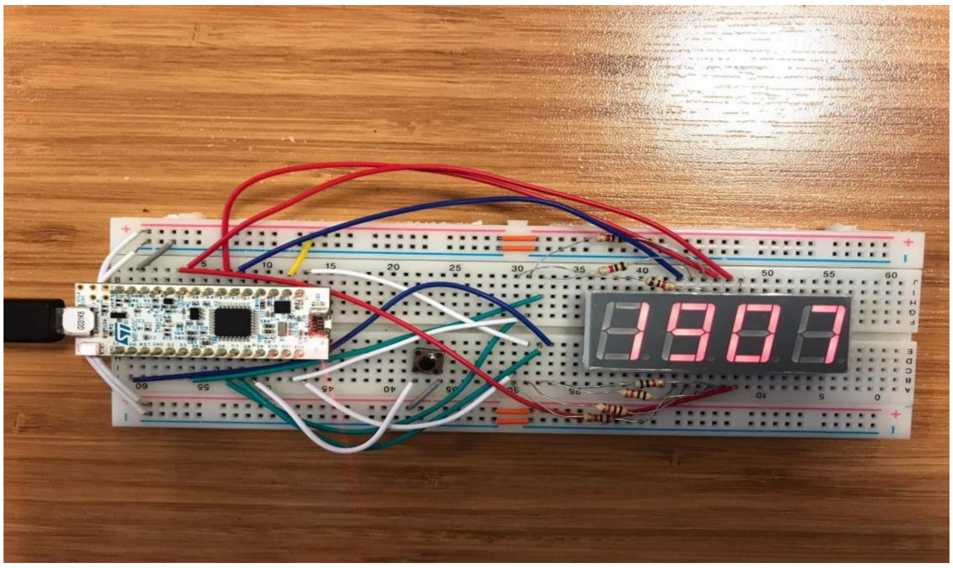
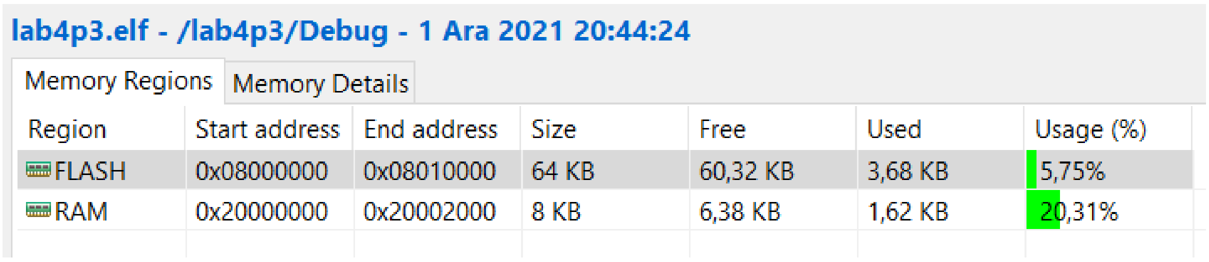


Figure 3.3) Circuit connections for problem 3

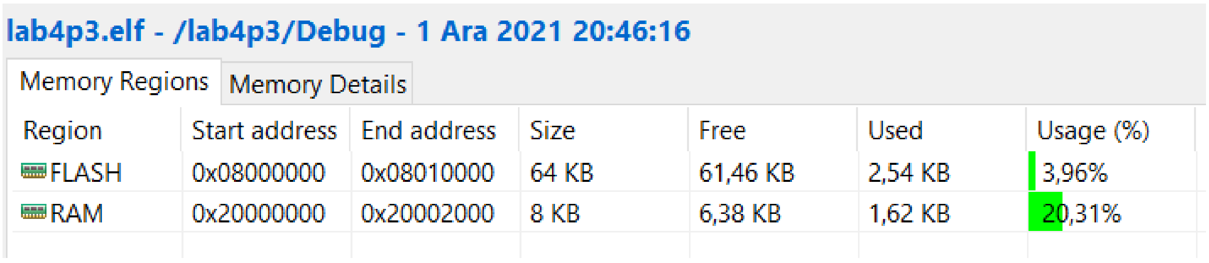
● What is the difference in code size when the optimization is enabled / disabled? How about the actual counter speed? Is there any change? If so, what would be the difference?
→ Counter speed is not changed.
● What is the code size percentage to the available ROM / RAM? How does optimization change this percentage?


lab4p3.elf - /lab4p3/Debug - 1 Ara 2021 20:44:24

Memory Regions | Memory Details

Region	Start address	End address	Size	Free	Used	Usage (%)
FLASH	0x08000000	0x08010000	64 KB	60,32 KB	3,68 KB	5,75%
RAM	0x20000000	0x20002000	8 KB	6,38 KB	1,62 KB	20,31%

Figure 3.4) -O0 optimization memory usage


lab4p3.elf - /lab4p3/Debug - 1 Ara 2021 20:46:16

Memory Regions | Memory Details

Region	Start address	End address	Size	Free	Used	Usage (%)
FLASH	0x08000000	0x08010000	64 KB	61,46 KB	2,54 KB	3,96%
RAM	0x20000000	0x20002000	8 KB	6,38 KB	1,62 KB	20,31%

Figure 3.5) -O2 optimization memory usage

→-O0 optimization used more memory than -O2 optimization.
● Is / Was there any brightness difference between the Seven Segments? If so, how did you fix it?
→No difference was observed in the brightness of the Seven Segments.

● Explain the difference between your implementation from the last lab? Which one is more convenient and scalable?
→ Implementing SSD with using timers is not affected by any optimization.

CODE:

```
//problem 3
#include "stm32g0xx.h"
#include "bsp.h"

#define BUTTON_DELAY    500
#define MAX_MINUTE      24
#define MAX_SECOND      60
enum count_timer_values{time1 = 999,      //mode1
                                    time2 = 499,      //mode2
                                    time3 = 99, //mode3
                                    time4 = 9,  //mode4
                                    time5 = 0   //mode5
};

volatile uint32_t SSD_number[] = {0x3F,   //0
                                                  0x06,     //1
                                                  0x5B,     //2
                                                  0x4F,     //3
                                                  0x66,     //4
                                                  0x6D,     //5
                                                  0x7D,     //6
                                                  0x07,     //7
                                                  0x7F,     //8
                                                  0x6F      //9
};

volatile uint32_t digit_display_number[] = {0x3F, 0x3F, 0x3F, 0x3F};
volatile uint32_t SSD_common_pins[] = {0x0010,0x0020,0x0040,0x0080};
volatile uint32_t temp_digit_display_number;
volatile uint32_t seconds;
volatile uint32_t minutes;
volatile uint32_t mode = 0;
volatile uint32_t counter_button = 0;
volatile uint32_t count_digit = 0;

void EXTI4_15_IRQHandler(){

      if ((counter_button >= BUTTON_DELAY) && ((EXTI->RPR1 >> 8) & 1)){
//Mode select
            seconds = 0;
            minutes = 0;

            if(mode >= 5){
                  mode = 0;
            }
            else{
                  mode++;
            }

            counter_button = 0;

      }
      EXTI->RPR1 |= (1U << 8);
```

```
}
void TIM1_BRK_UP_TRG_COM_IRQHandler(){


	digit_display_number[0] = SSD_number[(MAX_MINUTE-1-minutes)/10];
	digit_display_number[1] = SSD_number[(MAX_MINUTE-1-minutes)%10];
	digit_display_number[2] = SSD_number[(MAX_SECOND-1-seconds)/10];
	digit_display_number[3] = SSD_number[(MAX_SECOND-1-seconds)%10];

	seconds++;
	if(seconds == MAX_SECOND){
		seconds = 0;
		minutes++;
		if(minutes == MAX_MINUTE){
			minutes = 0;
		}
	}
	BSP_IO_pin_write(1,7,2);

	TIM1->SR &= ~(1U << 0);
}

void SysTick_Handler(){
	switch(mode){
		case 0:
			TIM1->CR1 &= ~(1U << 0);
			break;
		case 1:
			TIM1->CR1 |= (1U << 0);
			TIM1->PSC = time1;
			break;
		case 2:
			TIM1->PSC = time2;
			break;
		case 3:
			TIM1->PSC = time3;
			break;
		case 4:
			TIM1->PSC = time4;
			break;
		case 5:
			TIM1->PSC = time5;
			break;
	}
	if(count_digit >= 4){
		count_digit = 0;
	}

	GPIOB->ODR = digit_display_number[count_digit];
	GPIOA->ODR = ~SSD_common_pins[count_digit];

	count_digit++;
```

```
      if(counter_button <= BUTTON_DELAY){
            counter_button++;
      }

}


int main(void) {

      for(uint32_t i = 0;i<8;i++){  //SSD led pins PB0-PB7 output initial
(PB0-PB6 => A-G) (PB7 => second LED)
            BSP_IO_pin_init(1,i,1,0);
      }
      for(uint32_t i = 4;i<8;i++){  //SSD common pins PA4-PA7 output
initial (PA4-PA7 => D1-D4)
            BSP_IO_pin_init(0,i,1,0);
      }
      BSP_IO_pin_init(1,8,1,2);     //button pin PB8 input pulldown
initial
      BSP_EXTI_init(1,8,1,1);       //button pin PB8 interrupt initial
      NVIC_DisableIRQ(EXTI4_15_IRQn);
      NVIC_SetPriority(EXTI4_15_IRQn,1);
      NVIC_EnableIRQ(EXTI4_15_IRQn);
      BSP_TIMx_init(1,16000,time1,0);     //TIM1 initial
      BSP_SysTick_init(1);          //SysTick 1ms initial

      for(;;);

    return 0;
}
```

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.

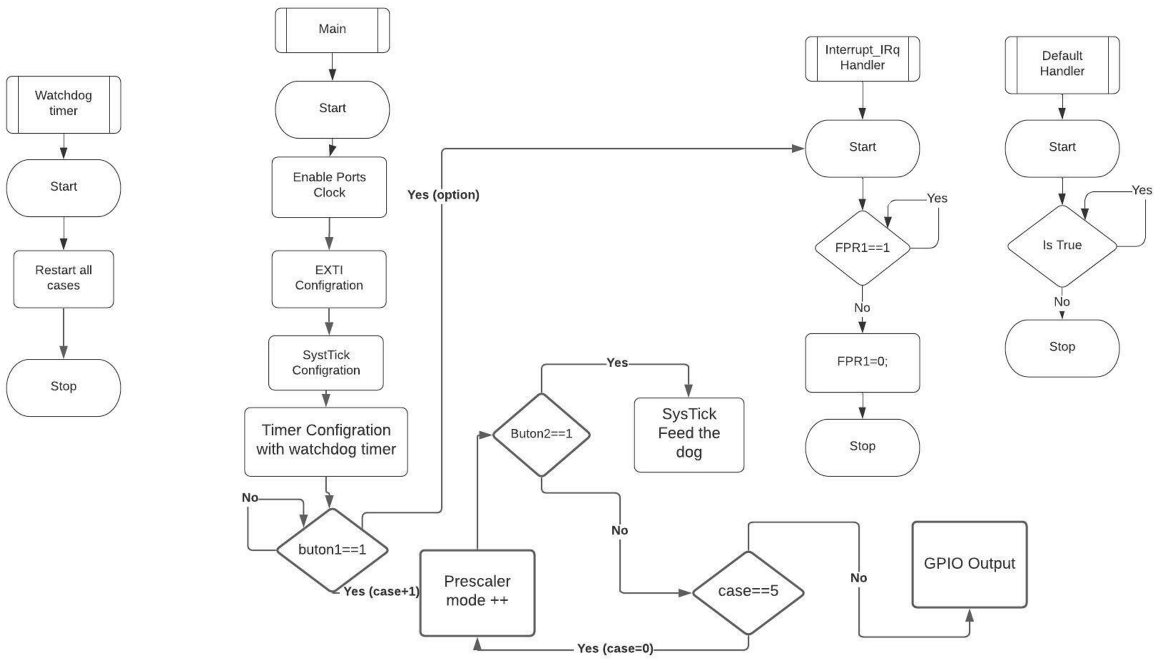


Figure 4.1) Flowchart of the problem 4

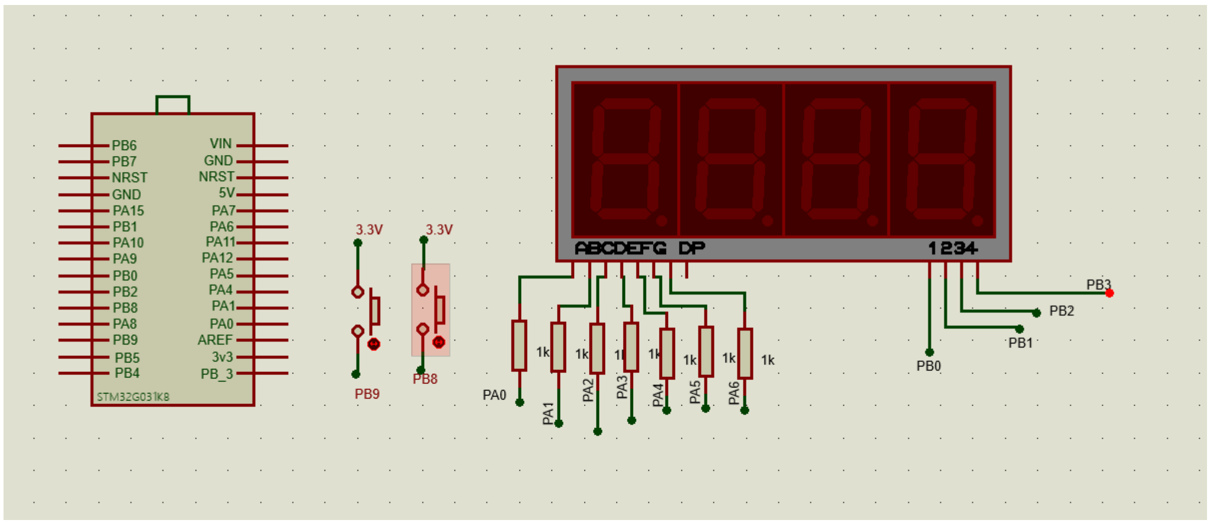


Figure 4.2) Circuit connection diagram for problem 4

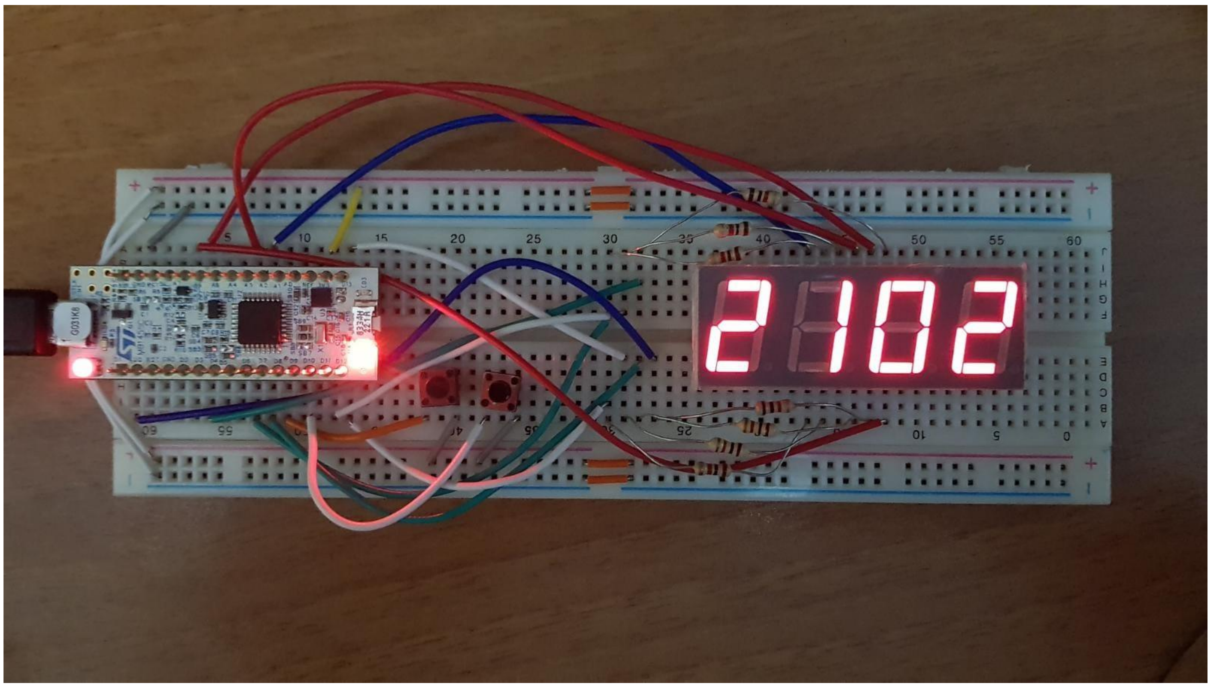


Figure 4.3) Circuit connections for problem 4

● How would you work with watchdog timers when there is nothing on the main routine?
● Create an external interrupt with a button that will turn off all the timer interrupts. (Not the exceptions and watchdog timer) This should fire up the watchdog timer since it will not be fed.
→ In the main function, only the variables are assigned. Watchdog timer is updated in the Systick timer handler function.If Systick timer does not go to interrupt, watchdog timer goes to interrupt and reset. Also, a button is connected to PB9. When the button is pressed all timers will be closed and the watchdog timer will go to interrupt. PB9 button works like a reset button.

CODE:

```
//problem 4
#include "stm32g0xx.h"
#include "bsp.h"

#define BUTTON_DELAY    500
#define MAX_MINUTE      24
#define MAX_SECOND      60
enum count_timer_values{time1 = 999,      //mode1
                                    time2 = 499,      //mode2
                                    time3 = 99, //mode3
                                    time4 = 9,  //mode4
                                    time5 = 0   //mode5
};

volatile uint32_t SSD_number[] = {0x3F,   //0
                                          0x06,     //1
                                          0x5B,     //2
                                          0x4F,     //3
                                          0x66,     //4
                                          0x6D,     //5
```

```
                                                    0x7D,     //6
                                                    0x07,     //7
                                                    0x7F,     //8
                                                    0x6F      //9
};

volatile uint32_t digit_display_number[] = {0x3F, 0x3F, 0x3F, 0x3F};
volatile uint32_t SSD_common_pins[] = {0x0010,0x0020,0x0040,0x0080};
volatile uint32_t temp_digit_display_number;
volatile uint32_t seconds;
volatile uint32_t minutes;
volatile uint32_t mode = 0;
volatile uint32_t counter_button = 0;
volatile uint32_t count_digit = 0;

void EXTI4_15_IRQHandler(){

      if ((counter_button >= BUTTON_DELAY) && ((EXTI->RPR1 >> 8) & 1)){
//Mode select
            seconds = 0;
            minutes = 0;

            if(mode >= 5){
                  mode = 0;
            }
            else{
                  mode++;
            }

            counter_button = 0;
      }
      if ((counter_button >= BUTTON_DELAY) && ((EXTI->RPR1 >> 9) & 1)){
//Mode select
            for(uint32_t i=0;i<4;i++){
                  digit_display_number[i] = SSD_number[8];
            }
            NVIC_DisableIRQ(TIM1_BRK_UP_TRG_COM_IRQn);
            SysTick->CTRL = 0;
            NVIC_DisableIRQ(SysTick_IRQn);
            counter_button = 0;
      }
      EXTI->RPR1 |= (3U << 8);
}
void TIM1_BRK_UP_TRG_COM_IRQHandler(){


      digit_display_number[0] = SSD_number[(MAX_MINUTE-1-minutes)/10];
      digit_display_number[1] = SSD_number[(MAX_MINUTE-1-minutes)%10];
      digit_display_number[2] = SSD_number[(MAX_SECOND-1-seconds)/10];
      digit_display_number[3] = SSD_number[(MAX_SECOND-1-seconds)%10];

      seconds++;
      if(seconds == MAX_SECOND){
            seconds = 0;
            minutes++;
            if(minutes == MAX_MINUTE){
                  minutes = 0;
            }
      }
      BSP_IO_pin_write(1,7,2);

      TIM1->SR &= ~(1U << 0);
}

void SysTick_Handler(){
      switch(mode){
            case 0:
                  TIM1->CR1 &= ~(1U << 0);
                  break;
            case 1:
                  TIM1->CR1 |= (1U << 0);
                  TIM1->PSC = time1;
                  break;
            case 2:
                  TIM1->PSC = time2;
                  break;
            case 3:
                  TIM1->PSC = time3;
                  break;
            case 4:
                  TIM1->PSC = time4;
                  break;
            case 5:
                  TIM1->PSC = time5;
                  break;
      }
      if(count_digit >= 4){
            count_digit = 0;
      }

      GPIOB->ODR = digit_display_number[count_digit];
      GPIOA->ODR = ~SSD_common_pins[count_digit];

      count_digit++;

      if(counter_button <= BUTTON_DELAY){
            counter_button++;
      }

      BSP_IWDG_feed();
}


int main(void) {
      BSP_IWDG_init();
      for(uint32_t i = 0;i<8;i++){  //SSD led pins PB0-PB7 output initial
```

```
(PB0-PB6 => A-G) (PB7 => second LED)
            BSP_IO_pin_init(1,i,1,0);
      }
      for(uint32_t i = 4;i<8;i++){  //SSD common pins PA4-PA7 output
initial (PA4-PA7 => D1-D4)
            BSP_IO_pin_init(0,i,1,0);
      }
      BSP_IO_pin_init(1,8,1,2);     //button pin PB8 input pulldown
initial
      BSP_EXTI_init(1,8,1,1);       //button pin PB8 interrupt initial
      BSP_IO_pin_init(1,9,1,2);     //button pin PB8 input pulldown
initial
            BSP_EXTI_init(1,9,1,1);       //button pin PB8 interrupt
initial
      NVIC_DisableIRQ(EXTI4_15_IRQn);
      NVIC_SetPriority(EXTI4_15_IRQn,1);
      NVIC_EnableIRQ(EXTI4_15_IRQn);
      BSP_TIMx_init(1,16000,time1,0);     //TIM1 initial
      BSP_SysTick_init(1);          //SysTick 1ms initial

      for(;;);

    return 0;
}
```