GEBZE TECHNICAL UNIVERSITY

ELECTRONIC ENGINEERING

ELM335

MICROPROCESSORS LAB

LAB 1 Report

Prepared By
1) 1901022038 – Selen Erdoğan
2) 200102002043 – Senanur Ağaç
3) 1901022050 - Merve Tutar

★ Objective

The purpose of making this lab is to give us a full understanding of timers. We used the C language while completing the projects. We used the blinky project from the stm32g0 repository as a starting point for the problems.

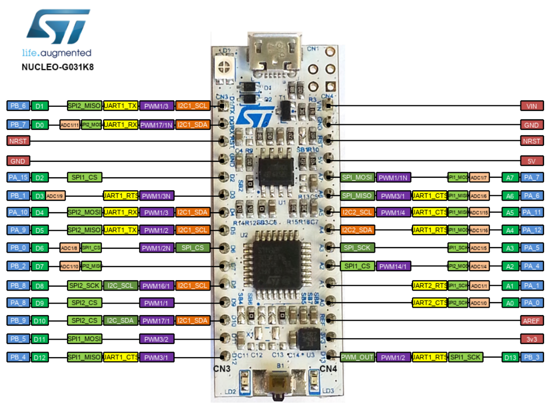
★ Problem 1

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

•How would you measure the accuracy of your delay using software methods?

•How would you measure the accuracy of your delay using hardware methods?

Explain each case, and (if possible) implement your solution.



★ Solution 1

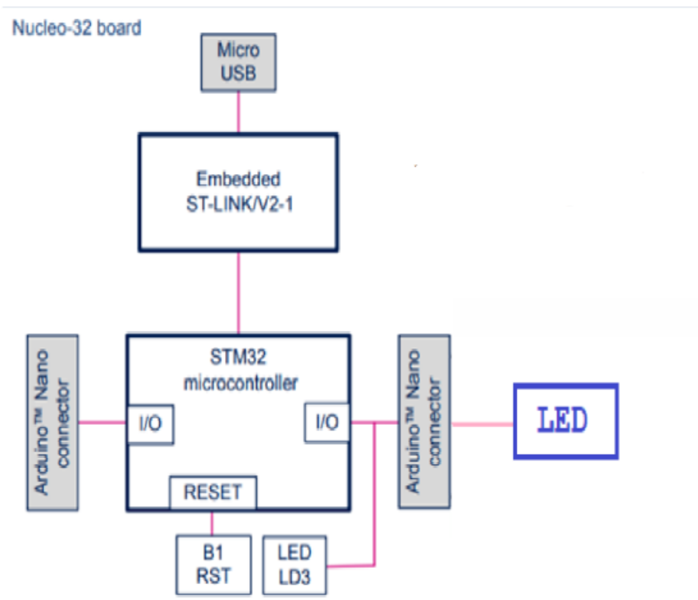


Figure 1. Block Diagram

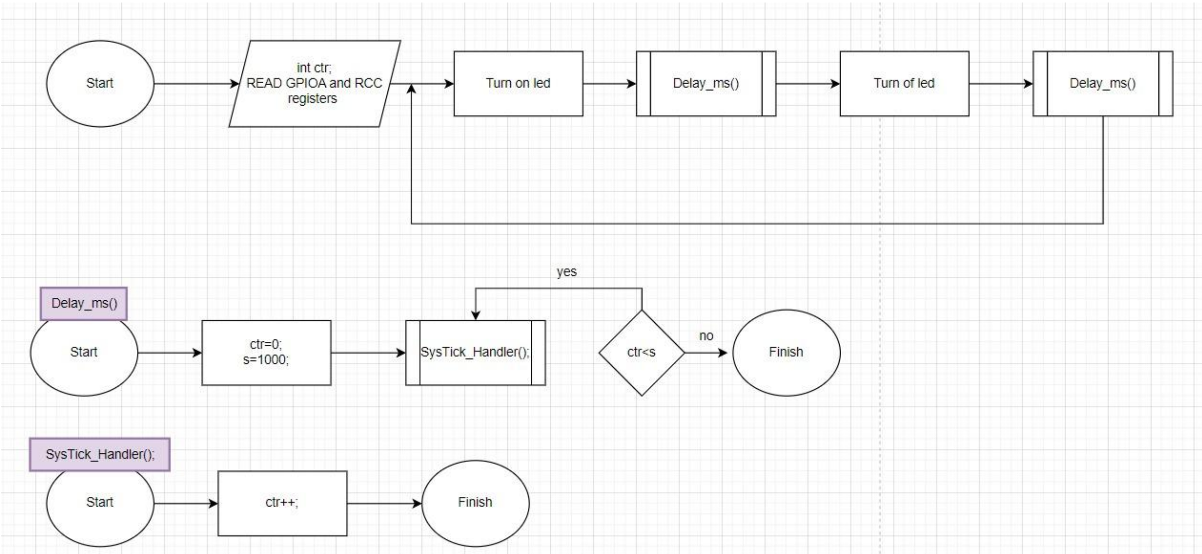


Figure2. Flowchart

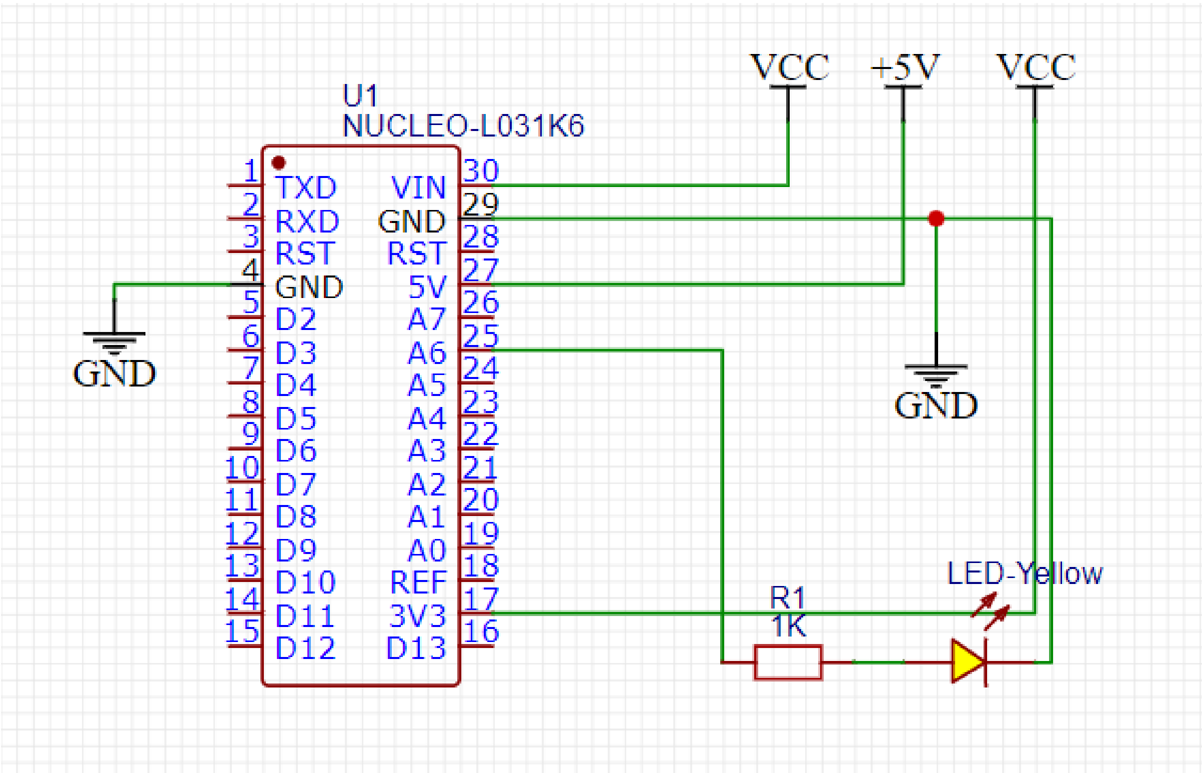


Figure 3. Schematic

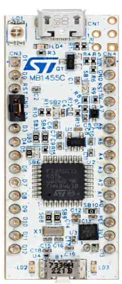
```
#include "stm32g0xx.h"
#define LEDDELAY 1600000
static int ctr;
void delay(volatile uint32_t);
```

```
void SysTick_Handler(void){
ctr++;
}
int main(void) {
SysTick_Config(16000); //16000000/1000 = 16000 => 1ms interrupt
 /* Enable GPIOA clock */
 RCC->IOPENR |= (1U << 0);
 /* Setup PA6 as output */
 GPIOA->MODER &= ~(3U << 2*6);
 GPIOA->MODER |= (1U << 2*6);//gp output mode 01
 while(1) {
 GPIOA->ODR |= (1U << 6);//turn on led pa6
 delay_ms(1000);
 GPIOA->BRR |= (1U << 6);//turn off led pa6
 delay_ms(1000);
 }
 return 0;
}
void delay_ms(volatile uint32_t s) {
 for (ctr=0 ; ctr < s; );//Thousand times 1ms makes a second delay
}
```

- ***Explanation Part***

With the SysTick exception, it was observed that the processor will be used to determine the operating speed, ie the cutting speed. Later it was seen that this command can be executed with SysTick_Config (16000). The reason for this was the operating speed of our processor. 16000000/1000 = 16000 => 1ms interrupt. Finally, SysTick_Handler (); With the function, we caught this interrupt and created a 1 ms delay. Since the counting process performed in this function is repeated 1000 times, it will save a cutting operation of 1 second. This cutting process was observed with the command of the led to be on and off by providing the control of this situation with a led. In this way, the led will turn on for 1 second and turn off for 1 second, and a correct delay function will be obtained thanks to the SysTick exception.

- **Problem 2**



In this problem, you will work with general purpose timers. Set up a timer with lowest priority that will be used to toggle on-board LED at 1 second intervals. Change the blinking speed using an external button. Each button press should increase the blinking speed by 1 second up to a maximum of 10 seconds. Next button press after 10 should revert it back to 1 second. All the functionality should be inside your interrupts.

★ Solution 2

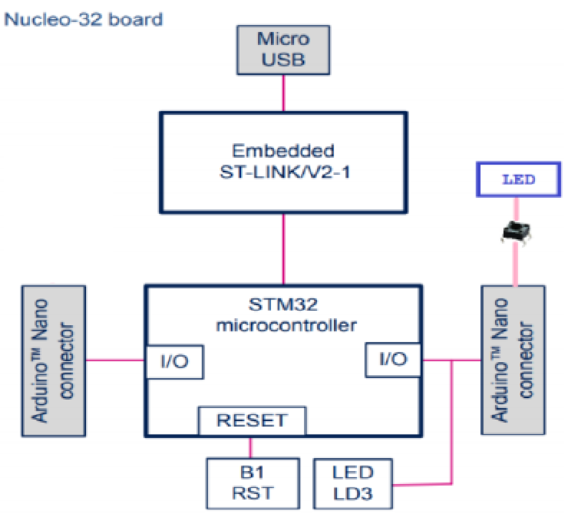


Figure 4. Block Diagram

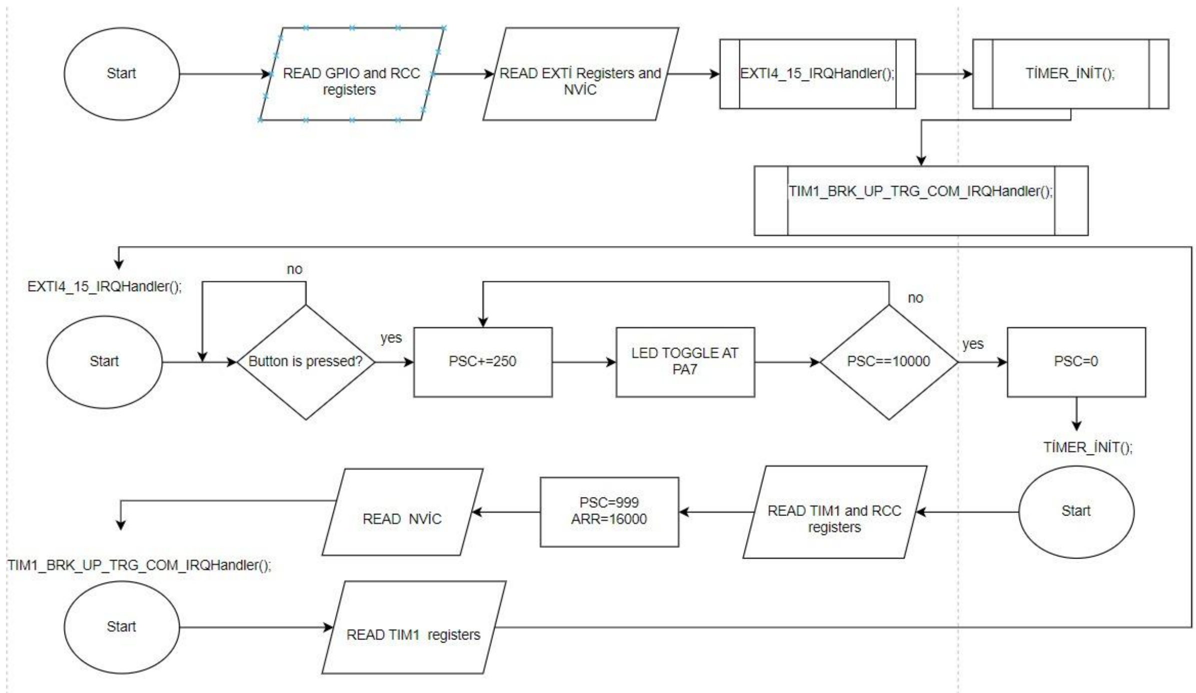


Figure 5. Flowchart

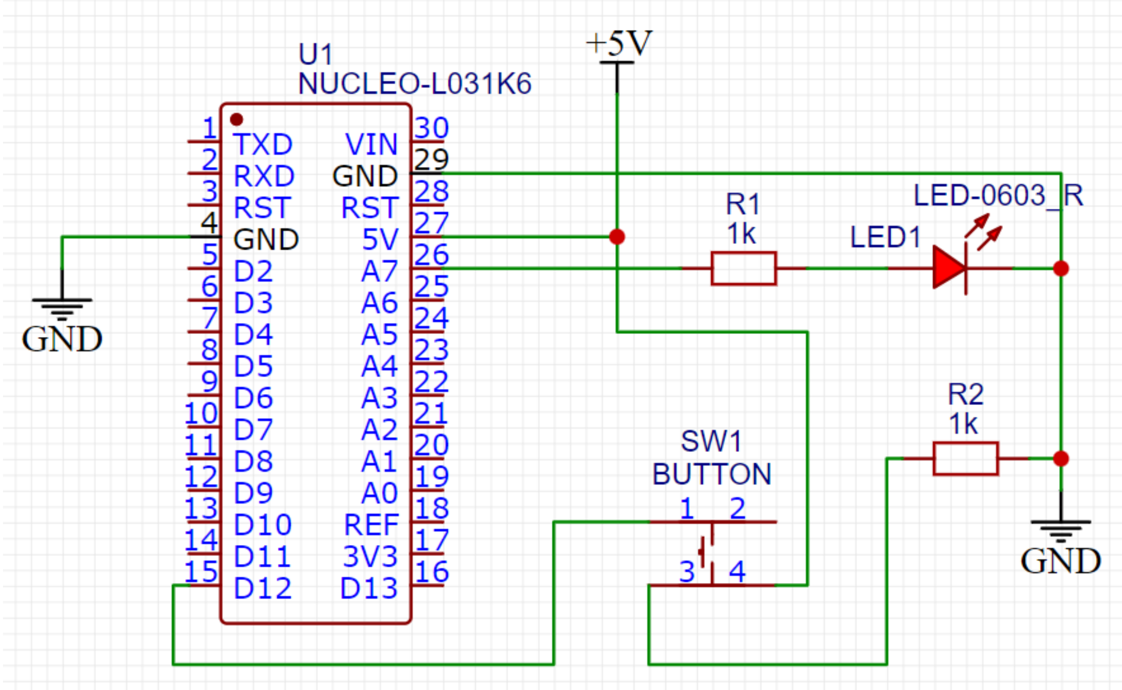


Figure 6. Schematic

```
#include "stm32g0xx.h"
void timer_init(){
/* enable TIM1 module clock */
RCC->APBENR2 |= (1U << 11); //golden rule for TIM1 module
//I chose TIM1 module 16 bit - 128MHz
TIM1->CR1 = 0; //control register is 0 for just in case
TIM1->CR1 |= (1 << 7); //arpe register is buffered
TIM1->CNT = 0; // counter register is 0
//1 second interrupt
TIM1->PSC = 999;
TIM1->ARR = 16000; //auto reload register
TIM1->DIER |= (1 << 0); // dier register is 1 because update interrupt enable
TIM1->CR1 |= (1 << 0); //control register is 1
NVIC_SetPriority(TIM1_BRK_UP_TRG_COM_IRQn, 1);
NVIC_EnableIRQ(TIM1_BRK_UP_TRG_COM_IRQn);
}
void EXTI4_15_IRQHandler(void){
TIM1->PSC += 250; //for catch all interrupt
if(TIM1->PSC > 10000)
TIM1->PSC = 0;
EXTI->RPR1 |= (1U << 3); // rising edge pending register
}
void TIM1_BRK_UP_TRG_COM_IRQHandler(void){
GPIOA->ODR ^= (1U << 7); // pa7 toggle
TIM1->SR &= ~(1U << 0); //status register
}
int main(void) {
/* Enable GPIOA clock */
RCC->IOPENR |= (1U << 0);
```

```
/* Setup PA7 as output */
GPIOA->MODER &= ~(3U << 2*7);
GPIOA->MODER |= (1U << 2*7);
/* clear LED */
GPIOA->BRR |= (1U << 7); // for all case
/* Enable GPIOB clock */
RCC->IOPENR |= (1U << 1);
/*Setup PB4 as input*//*button*/
GPIOB->MODER &= ~(3U << 2*4); // input mode 00
/* rising edge, selection register and mask register */
EXTI->RTSR1 |= (1U << 4);
EXTI->EXTICR[1] |= (1U << 8*0);
EXTI->IMR1 |= (1U << 4);
/* enable NVIC and set interrupt priority */
NVIC_SetPriority(EXTI4_15_IRQn, 0);
NVIC_EnableIRQ(EXTI4_15_IRQn);
timer_init();
 while(1) {
 }
 return 0;
}
```

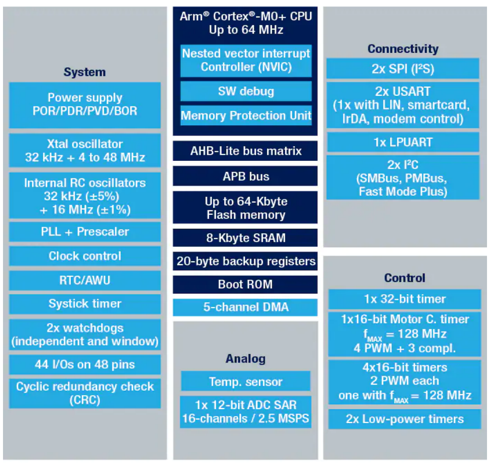
★ Explanation Part

TIM1-> PSC = 999;

TIM1-> ARR = 16000;

Thanks to its commands, a 1-second delay was created. The delay generation time had to be increased each time an external button was pressed. For this reason, it was commanded to TIM1-> PSC register by trial and error method that the led should light for 1 more seconds each time the button is pressed. The 250 value used here gave us enough time to get the required time since it was asked to both add and write. When the button is pressed for the 11th time, the burning speed of the led will return to the initial state so that a value of 0 is assigned to the PSC register and it will increase according to the number of presses made from the button.

★ Problem 3



In this problem, connect your seven-segmen tdisplay and implement a count up timer. It should sit at zero, once the external button is pressed, it should count up to the maxvalue (i.e.9999) then once it overflows to 0, it should stop,and light up an LED (on-board or external). Pressing thebutton again should clear the LED and count again. For this problem, your main loop should not have anything. All the functionality should be inside your interrupts

★ **Solution 3**

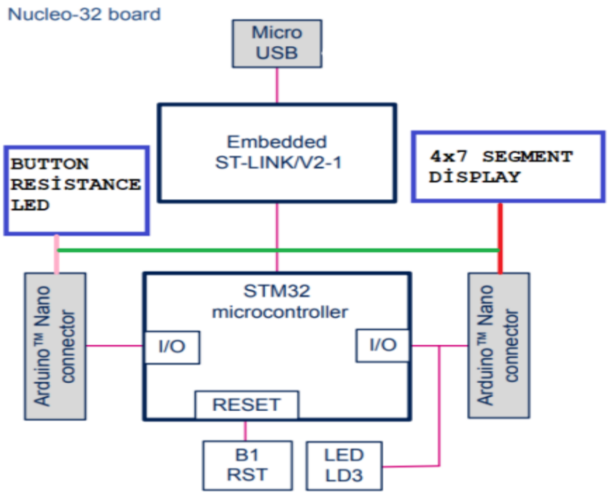


Figure7. Block Diagram

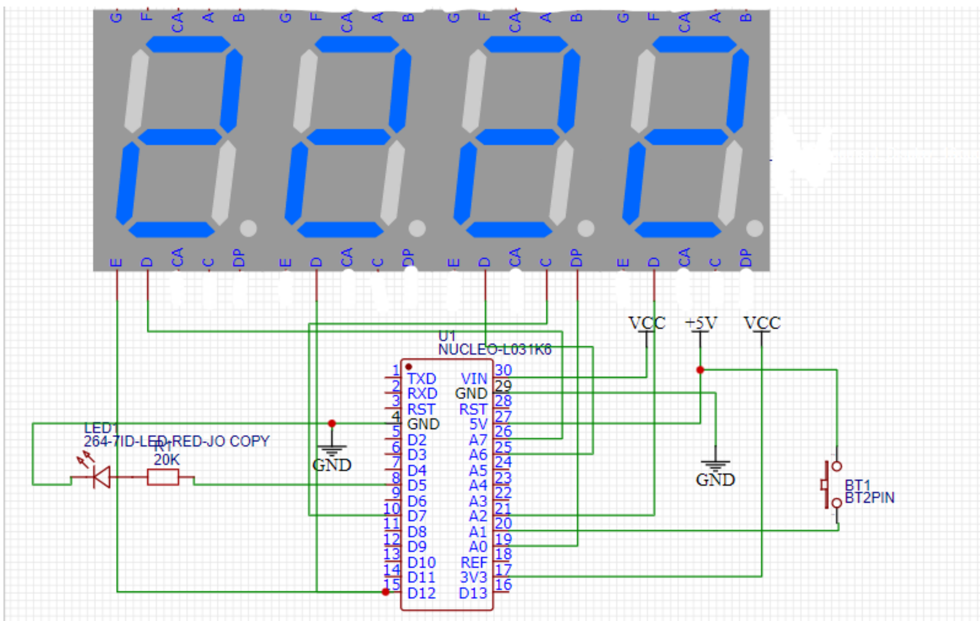


Figure8. Schematic

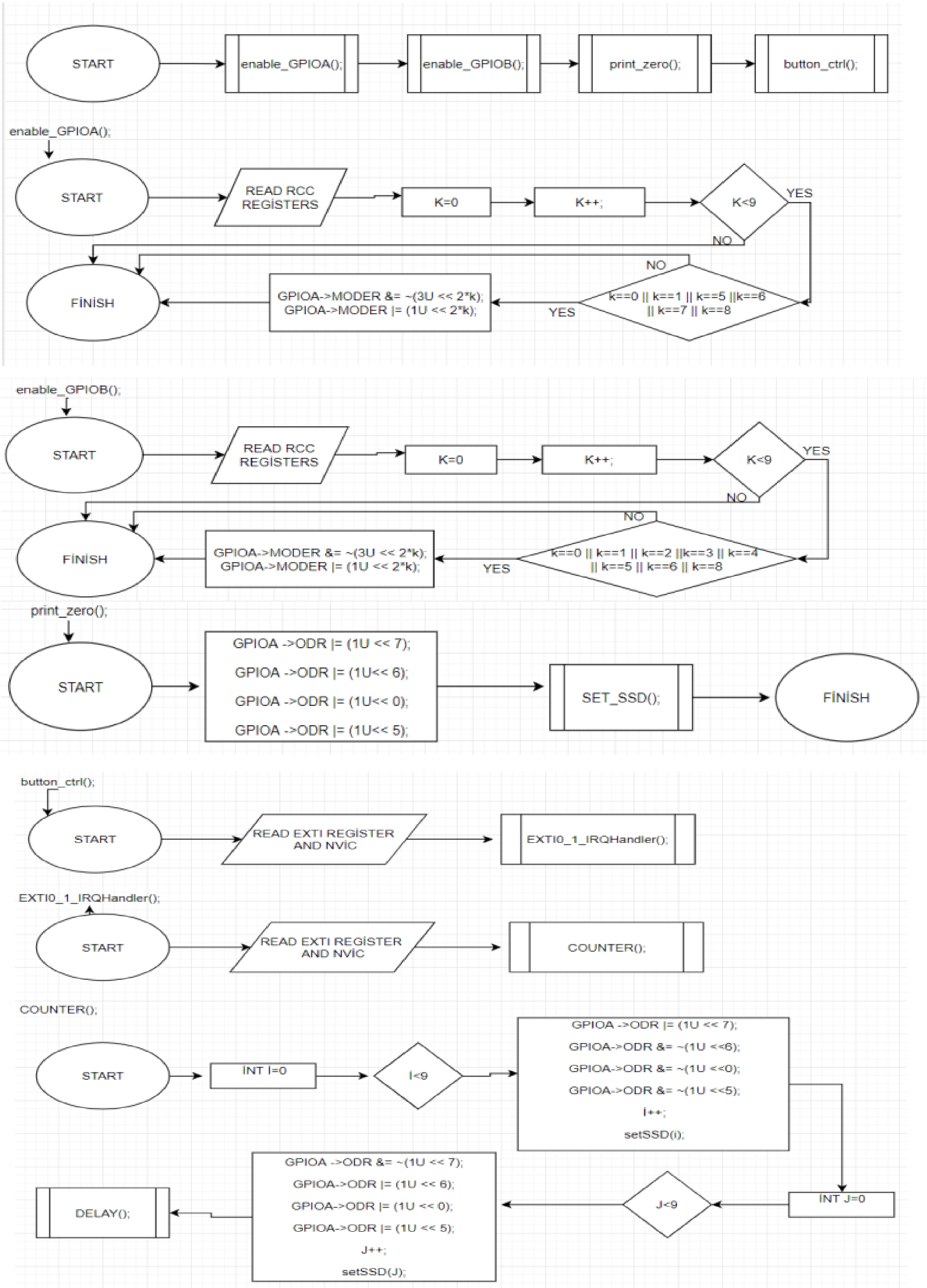


Figure9. Flowchart

- ★ ***Explanation Part***

void clearSSD (void); Remains of the LEDs from previous situations were cleaned with.
void setSSD (int x); We activated the LEDs with and now we are
writing C code, so SSD LEDs are activated by using for loop comfortably except for Assembly.
void counter (void); Thanks to this, the first digit became the second counter and the last 3 digits became the millisecond counter.
In the selection of this function, the set_SSD function is created and the counter and SSD connection are made.
void button_ctrl (void); Assign the button pin to be connected with
void EXTI0_1_IRQHandler (void); By calling a counter in the function, the counter SSD and button connection was made, and thus it was ensured that the counter would not start counting without
taking a command from the button.
The pins are enabled thanks to the void enable_GPIOA () and void enable_GPIOB () functions. Thanks to for loop in C language, these operations were carried out easily by logical elimination. The same pins in project1 were used as the circuit diagram and pin connections. In this way, the same connections were established over a connection scheme I am used to and the circuit was completed. By assigning a resistor for each led, the leds were protected, thus preventing the leds from losing their function.

```
#include "stm32g0xx.h"
void clearSSD(void){
GPIOB -> ODR |= (1U << 0); //PB0 A
GPIOB -> ODR |= (1U << 1); //PB1 B
GPIOB -> ODR |= (1U << 2); //PB2 C
GPIOB -> ODR |= (1U << 3); //PB3 D
GPIOB -> ODR |= (1U << 4); //PB4 E
GPIOB -> ODR |= (1U << 5); //PB5 F
GPIOB -> ODR |= (1U << 6); //PB6 G
}
void setSSD(int x){
clearSSD();
switch(x){
case 0:
for(int i=0; i<6; i++){// A B C D E F
GPIOB->ODR &= ~(1U << i);
}
 break;
case 1:
```

```
for(int i=1; i<3; i++){ //B C
GPIOB->ODR &= ~(1U << i);
}
 break;
case 2:
for(int i=0; i<7; i++){// A B D E G
if (i==0 || i==1 || i==3 || i==4 || i==6){
GPIOB->ODR &= ~(1U << i);
}
}
 break;
case 3:
for(int i=0; i<7; i++){// A B C D G
if (i==0 || i==1 || i==2 || i==3 || i==6){
GPIOB->ODR &= ~(1U << i);
}
}
break;
case 4:
for(int i=0; i<7; i++){// B C F G
if (i==1 || i==2 || i==5 || i==6){
GPIOB->ODR &= ~(1U << i);
}
}
break;
case 5:
for(int i=0; i<7; i++){// A C D F G
if (i==0 || i==2 || i==3 || i==5 || i==6){
GPIOB->ODR &= ~(1U << i);
}
}
break;
case 6:
GPIOB->ODR &= ~(1U << 0);// A C D E F G
for(int i=2; i<7; i++){
GPIOB->ODR &= ~(1U << i);
}
 break;
case 7:
for(int i=0; i<3; i++){// A B C
GPIOB->ODR &= ~(1U << i);
}
break;
case 8:
for(int i=0; i<7; i++){// A B C D E F G
GPIOB->ODR &= ~(1U << i);
}
 break;
case 9:
GPIOB->ODR &= ~(1U << 0); //A
GPIOB->ODR &= ~(1U << 1); //B
```

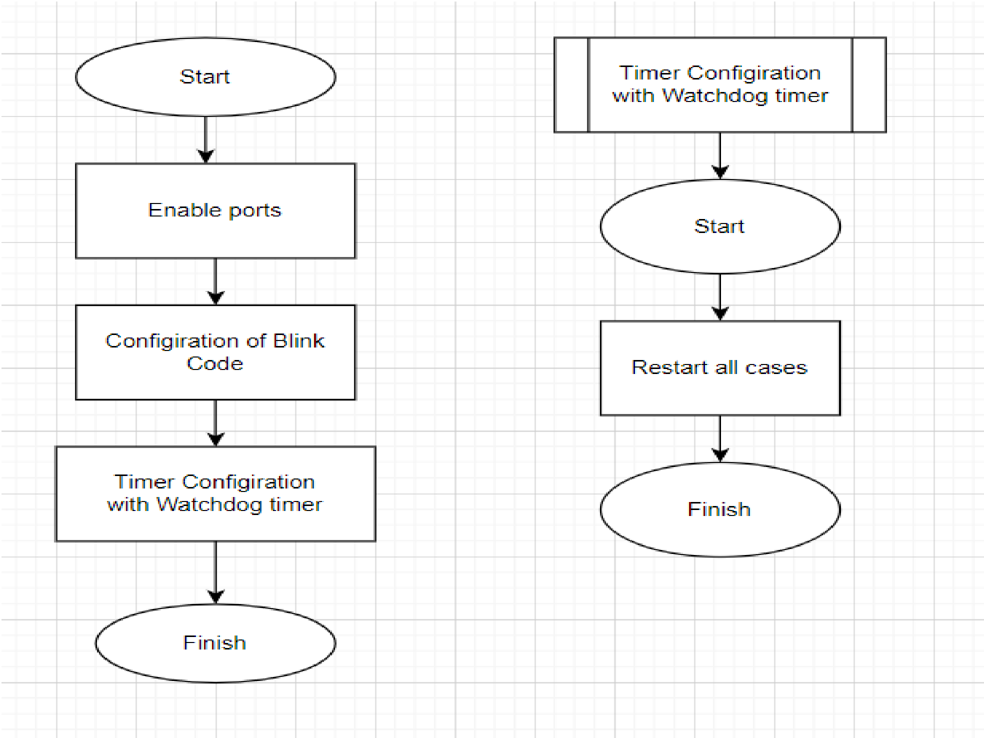
```
GPIOB->ODR &= ~(1U << 2); //C
GPIOB->ODR &= ~(1U << 3); //D
GPIOB->ODR &= ~(1U << 5); //F
GPIOB->ODR &= ~(1U << 6); //G
 break;
}
}
void counter(void){
print_zero();
delay(1000000);
for(int i=0; i<=9; i++){//second counter
GPIOA ->ODR |= (1U << 7); // D1 - PA7
GPIOA ->ODR &= ~(1U << 6); // D2 - PA6
GPIOA ->ODR &= ~(1U << 0); // D3 - PA0
GPIOA ->ODR &= ~(1U << 5); // D4 - PA5
setSSD(i);
delay(1000000);
for(int j=0; j<=9; j++){//milisecond counter
if(i<9)
GPIOA ->ODR &= ~(1U << 7); // D1 - PA7
GPIOA ->ODR |= (1U << 6); // D2 - PA6
GPIOA ->ODR |= (1U << 0); // D3 - PA0
GPIOA ->ODR |= (1U << 5); // D4 - PA5
setSSD(j);
delay(50000);
}
}
GPIOB->ODR |= (1U << 8); // PB8 - D8 LED turn on
delay(1000000);
GPIOB->BRR |= (1U << 8); // led turn off
print_zero();
delay(1000000);
}
void button_ctrl(void){//PA1 İS BUTTON
/* rising edge, selection register and mask register */
EXTI->RTSR1 |= (1U << 1);
EXTI->EXTICR[0] |= (0U << 8*1);
EXTI->IMR1 |= (1U << 1);
/* enable NVIC and set interrupt priority */
NVIC_SetPriority(EXTI0_1_IRQn, 0);
NVIC_EnableIRQ(EXTI0_1_IRQn);
}
void EXTI0_1_IRQHandler(void){
counter();
EXTI->RPR1 |= (1U << 1); // rising edge
}
void enable_GPIOA(){
/* enable required GPIOA registers and RCC register */
RCC->IOPENR |= (1U << 0);
for(int k=0; k<9; k++){
if (k==0 || k==1 || k==5 || k==6 || k==7 || k==8){
```

```
GPIOA->MODER &= ~(3U << 2*k);
GPIOA->MODER |= (1U << 2*k);
}
}
}
void enable_GPIOB(){
/* enable required GPIOB registers and RCC register */
RCC->IOPENR |= (1U << 1);
for(int k=0; k<9; k++){
if (k==0 || k==1 || k==2 || k==3 || k==4 || k==5 || k==6 || k==8){
GPIOB->MODER &= ~(3U << 2*k);
GPIOB->MODER |= (1U << 2*k);
}
}
}
void print_zero(){
GPIOA ->ODR |= (1U << 7); // D1 digit to PA7
 GPIOA ->ODR |= (1U << 6); // D2 digit to PA6
 GPIOA ->ODR |= (1U << 0); // D3 digit to PA0
 GPIOA ->ODR |= (1U << 5); // D4 digit to PA5
 setSSD(0);
}
int main(void) {
 enable_GPIOA();
 enable_GPIOB();
 print_zero();
 button_ctrl();
 while(1) {
 }
 return 0;
}
void delay(volatile uint32_t s) {
 for(; s>0; s--);
}
```

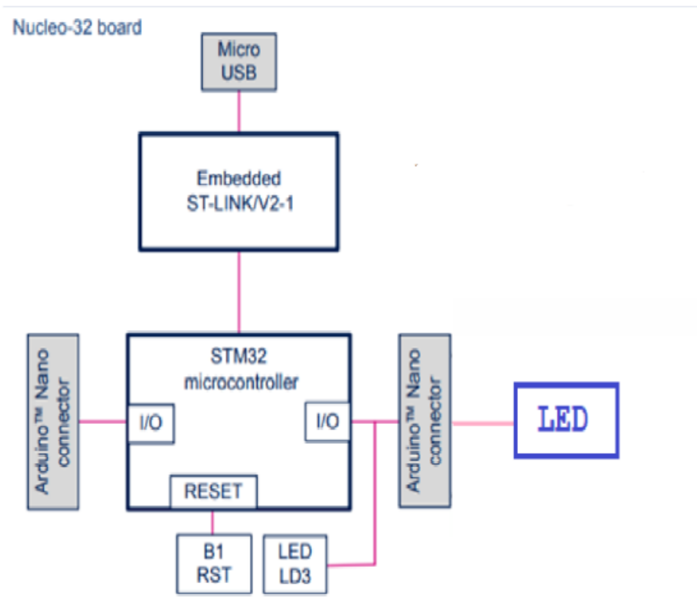
★ **Problem 4**

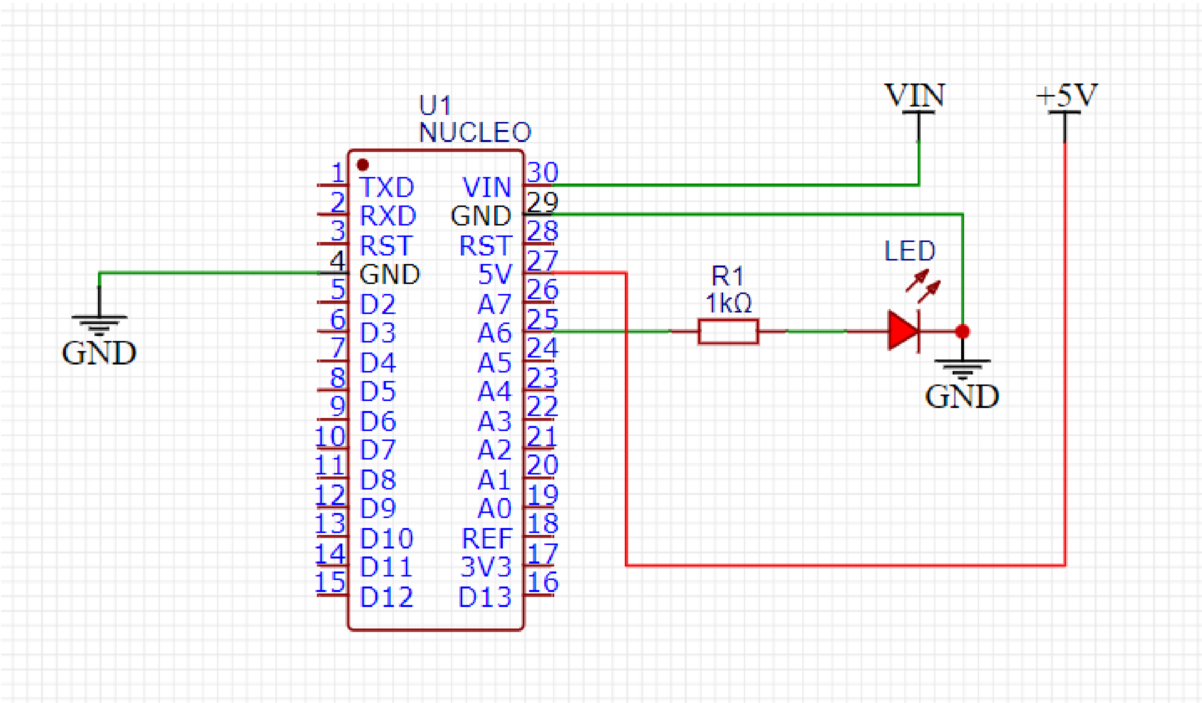
In this problem, you will work with watchdog timers. Setup either window or independent watchdog timer and observe its behavior in the simple blinky example from the repo. Calculate the appropriate reset time and implement it. Add the necessary handler routine for resetting the device.

★ **Solution 4**


Start
Enable ports
Configiration of Blink Code
Timer Configiration with Watchdog timer
Finish
Timer Configiration with Watchdog timer
Start
Restart all cases
Finish

★


Nucleo-32 board
Micro USB
Embedded ST-LINK/V2-1
STM32 microcontroller
I/O
I/O
RESET
Arduino™ Nano connector
Arduino™ Nano connector
LED
B1 RST
LED LD3




```
//problem 4

#include "stm32g0xx.h"
#include "bsp.h"
void timer_init(){
/* enable TIM1 module clock */
RCC->APBENR2 |= (1U << 11); //golden rule for TIM1 module
//I chose TIM1 module 16 bit - 128MHz
TIM1->CR1 = 0; //control register is 0 for just in case
TIM1->CR1 |= (1 << 7); //arpe register is buffered
TIM1->CNT = 0; // counter register is 0
//1 second interrupt
TIM1->PSC = 999;
TIM1->ARR = 16000; //auto reload register
TIM1->DIER |= (1 << 0); // dier register is 1 because update interrupt enable
TIM1->CR1 |= (1 << 0); //control register is 1
NVIC_SetPriority(TIM1_BRK_UP_TRG_COM_IRQn, 1);
NVIC_EnableIRQ(TIM1_BRK_UP_TRG_COM_IRQn);


}
void EXTI4_15_IRQHandler(void){
TIM1->PSC += 250; //for catch all interrupt
if(TIM1->PSC > 10000)
TIM1->PSC = 0;
EXTI->RPR1 |= (1U << 3); // rising edge pending register
```

```
}
void TIM1_BRK_UP_TRG_COM_IRQHandler(void){
GPIOA->ODR ^= (1U << 7); // pa7 toggle
TIM1->SR &= ~(1U << 0); //status register
BSP_IWDG_init();

}
int main(void) {
        BSP_IWDG_init();

/* Enable GPIOA clock */
RCC->IOPENR |= (1U << 0);
/* Setup PA7 as output */
GPIOA->MODER &= ~(3U << 2*7);
GPIOA->MODER |= (1U << 2*7);
/* clear LED */
GPIOA->BRR |= (1U << 7); // for all case
/* Enable GPIOB clock */
RCC->IOPENR |= (1U << 1);
/*Setup PB4 as input*//*button*/
GPIOB->MODER &= ~(3U << 2*4); // input mode 00
/* rising edge, selection register and mask register */
EXTI->RTSR1 |= (1U << 4);
EXTI->EXTICR[1] |= (1U << 8*0);
EXTI->IMR1 |= (1U << 4);
/* enable NVIC and set interrupt priority */
NVIC_SetPriority(EXTI4_15_IRQn, 0);
NVIC_EnableIRQ(EXTI4_15_IRQn);
timer_init();
 while(1) {
 }
 return 0;
}
```

★ **Problem 5**

In this problem, you will implement your watchdog timer in Problem 3. Figure out a way to properly incorporate into your code when it all works with timers. Explain your solution and implementation and make sure you covered all possible scenarios.

★ **Solution 5**

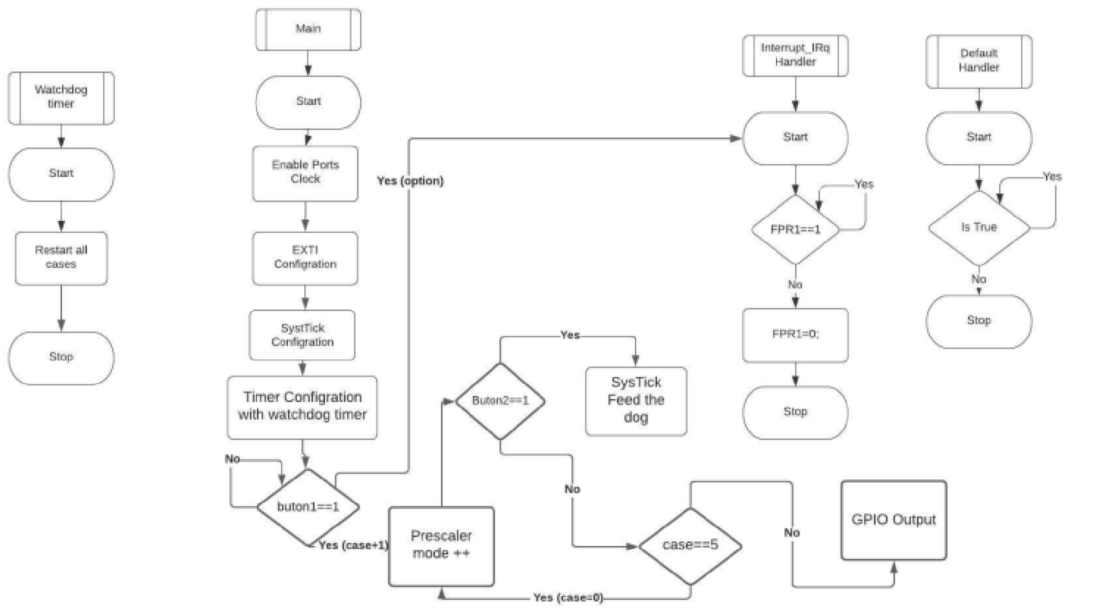


Figure10. Flowchart

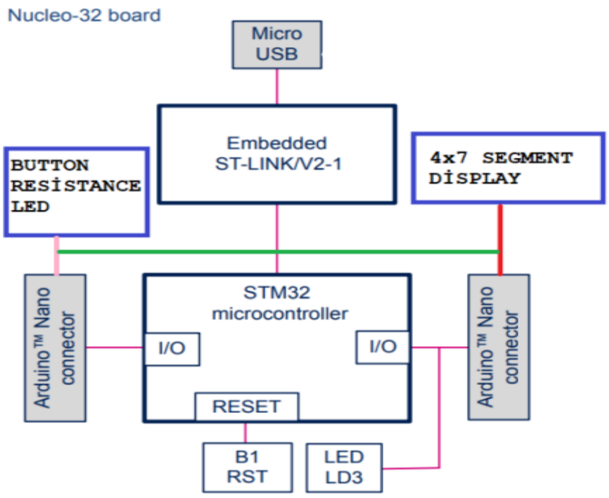


Figure11. Block Diagram

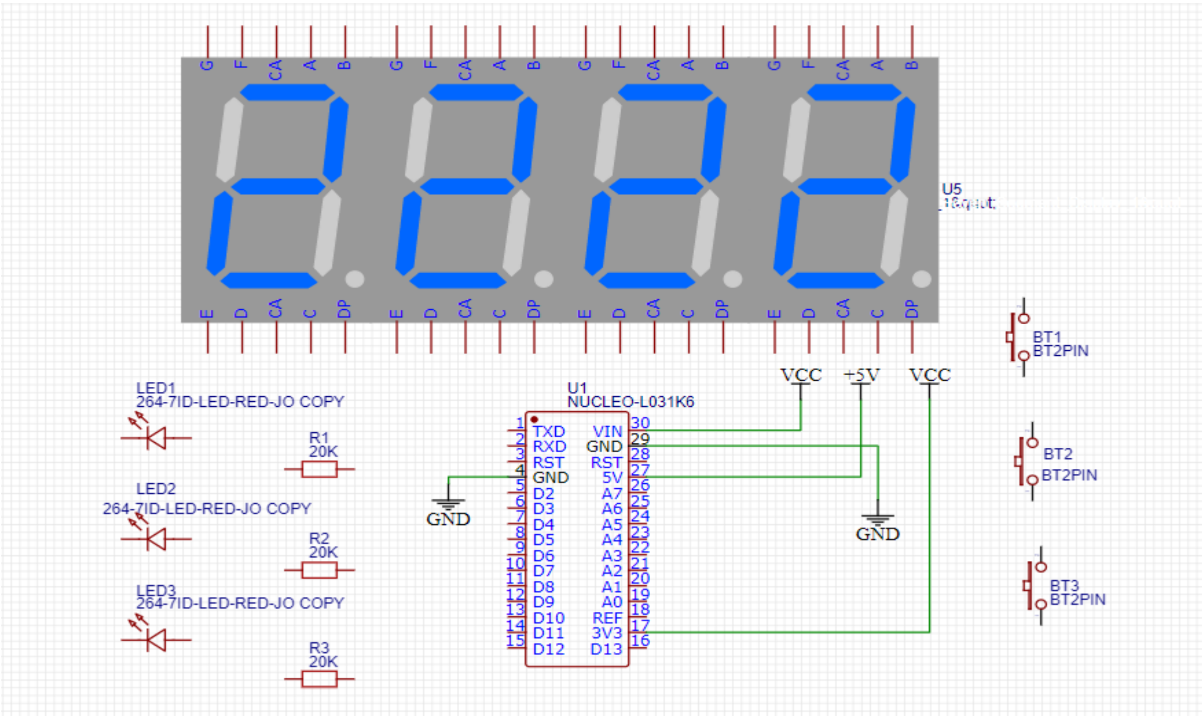


Figure 12. Schematic

```
//problem 5
#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 outputinitial (PA4-PA7 =>
D1-D4)
		BSP_IO_pin_init(0,i,1,0);
	}
	BSP_IO_pin_init(1,8,1,2); //button pin PB8 input pulldowninitial
	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;
}
```

```
//BSP.H Header file
//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;
        }
}
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;
}
```

- ★ *Explanation Code*

We 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 Systicktimer handler function.If Systick timer does not go to interrupt, watchdog timer goes to interrupt and reset. When the button is pressed all timers will be closed and the watchdog timer will go to interrupt.