GEBZE TECHNICAL UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF ELECTRONICS ENGINEERING

ELEC 335

Microprocessors Laboratory

Lab #3 Experiment Report

Hazırlayanlar
200102002025 – Umut Mehmet ERDEM
200102002051 – Arda DERİCİ
200102002061 – Serdar BAŞYEMENİCİ

1. Introduction

The aim of ELEC 335 Lab #3 is to provide a comprehensive understanding of timers and to implement this in practice. An attempt will be made to create an accurate delay function by using the “SysTick” exception and the blinking speeds of the LED will be changed with the button. A count-up timer will be implemented using the seven-segment display on the board. In this way, the term count up timer will be understood and implemented. The terms window and independent watchdog timer will be understood and implemented. Additionally, how to use the count up timer and watchdog timer in the same code will be considered and implemented.

2. Problems

2.1. Problem I

2.1.1. C Code and of the Problem I

```
/* author: Umut Mehmet Erdem | Arda Derici | Serdar Başyemenici
* problem1.c
*/

#include "stm32g0xx.h"
#include "stm32g0xx_it.h"

uint32_t counter; // parameter called counter is defined as unsigned int 32 bits
type.
#define LEDDELAY    1000 // 1 second

// presign of function
void led_toggle(void);
void SysInitial(void);
void delay_ms(uint32_t s);
void led_init(void);

int main(void) {

      SysInitial(); // system initial function is activated.

    while(1) {
      led_toggle(); // led is toggled by using XOR.
        delay_ms(LEDDELAY); // delay for LEDDELAY miliseconds
    }

    return 0;
}

void SysInitial(void){
      SysTick_Config(SystemCoreClock/1000); // SysTick_Handler() function's time is
defined in function parameter.
      led_init(); //led initilize
}
void SysTick_Handler(void)
{
  if(counter != 0){ // if counter is not 0.
        counter--; // counter is decreasing once in each cycle.
  }
}
```

```
void delay_ms(uint32_t s) {
	counter = s;
	while(counter);
}

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

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

void led_toggle(void){
	// Toggle LED
	GPIOC->ODR ^= (1U << 6);
}
```

2.2. Problem II

2.2.1. Flow Chart and Schematic Diagram

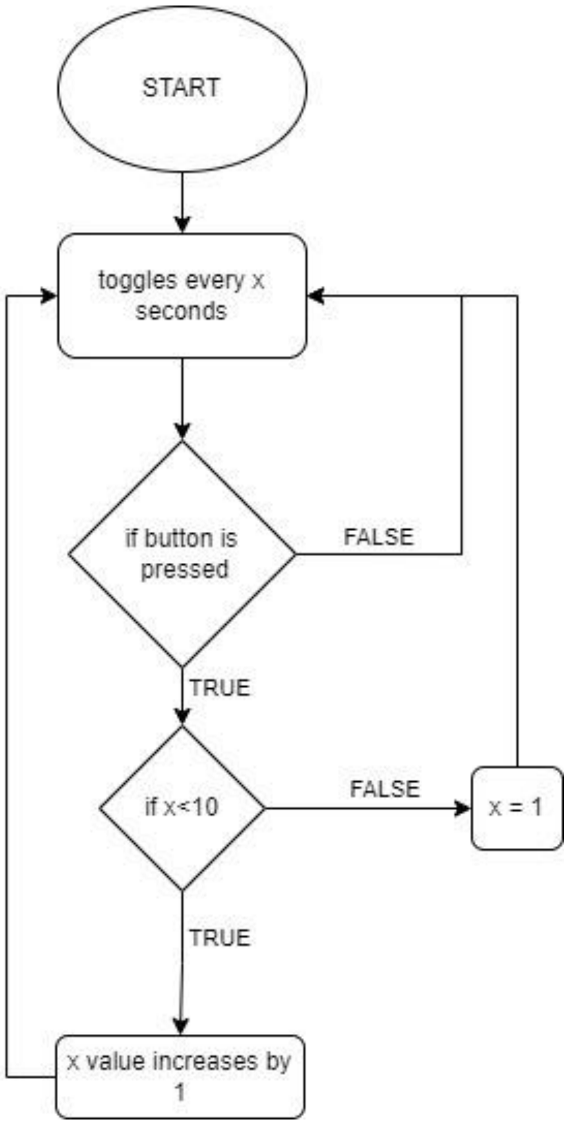

Figure 1 – flow chart for Problem II

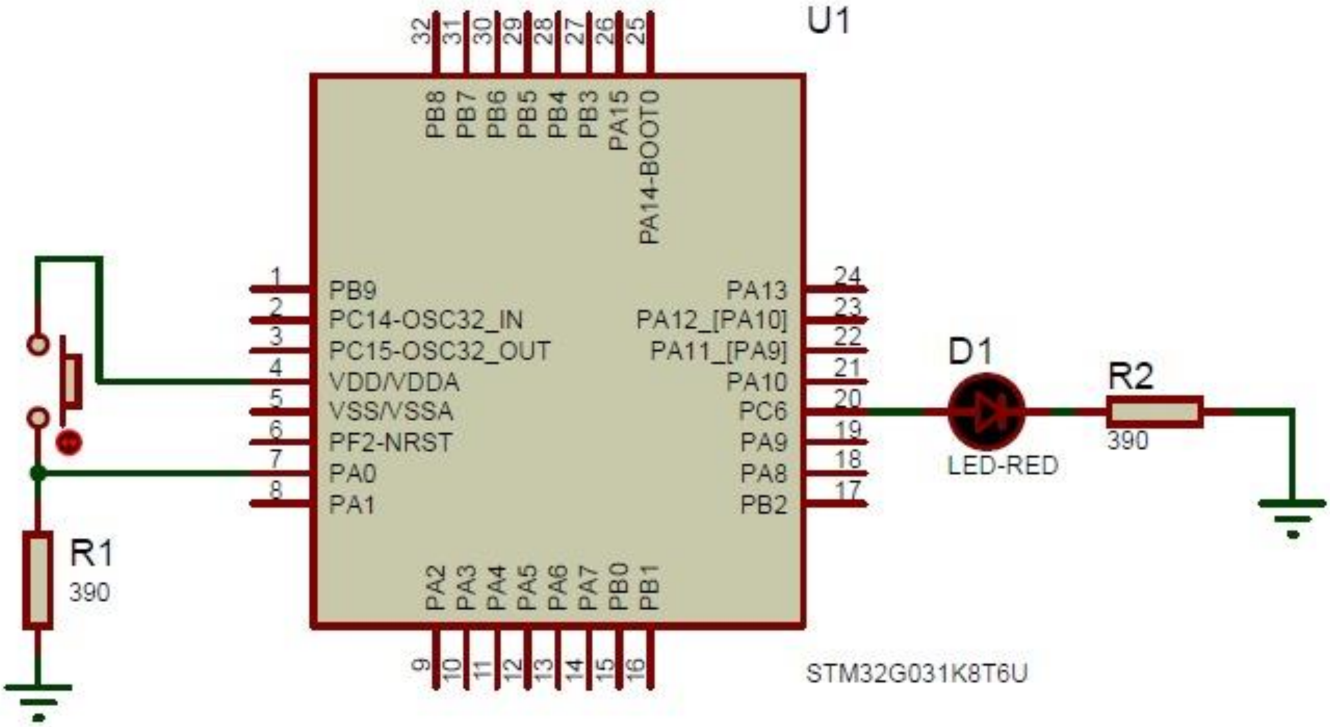

Figure 2 – schematic diagram for Problem II

2.2.2. C Code of the Problem II

```
/* author: Umut Mehmet Erdem | Arda Derici | Serdar Başyemenici
* problem2.c
*/
#include "stm32g0xx.h"

int PSC_val = 1;
uint8_t flag = 0;


void delay(uint32_t);
void LedInit(void);
void LedToggle(void);
void ButtonInit(void);
void InitTimer(void);
void EXTI0_1_IRQHandler(void);

int main(void) {

	LedInit();
	ButtonInit();
	InitTimer();

	TIM3->PSC = 1000*PSC_val; // in the beginning, PSC= 1000 for 1s.

	while(1){

		if(flag){ // When the button is pressed, the flag will be equal to 1 and 
will enter the if block.
			PSC_val++; //PSC_val is increasing once.
			delay(2000000);
			TIM3->PSC = 1000*PSC_val; // according to PSC_val, PSC is 
increasing.

			if(PSC_val == 11){ //PSC_val returns to 1 after 11
				PSC_val = 1;
			}
			flag = 0; // When the button is pressed again, the flag is set to 
0 to enter the if block.
		}
	}

	return 0;
}

void InitTimer(void){

	RCC->APBENR1 |= (1U<<1); //

	TIM3->CR1 = 0; //TIM3 control register 1 for enabling Counter enable
	TIM3->CR1 |= (1<<7); // for Auto-reload preload enable
	TIM3->CNT = 0; // TIMx counter

	TIM3->PSC = 1000; // TIMx prescaler
	TIM3->ARR = 16000; // TIMx auto-reload register
```

```
	TIM3->DIER |= (1<<0); // TIMx DMA/Interrupt enable register
	TIM3->CR1 |= (1<<0); // for Counter enable

	NVIC_SetPriority(TIM3_IRQn,0); // TIM3 priority is 0
	NVIC_EnableIRQ(TIM3_IRQn); // interrupt is enabled.
}

void TIM3_IRQHandler (){
	LedToggle();
	TIM3->SR &= ~(1U << 0);
	/* TIMx status register - Update interrupt flag
	This bit is set by hardware on an update event. It is cleared by software.*/
}

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

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

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

    /* Clear PC6 */
    GPIOC->BRR |= (1U << 6);
}

void LedToggle(void){
	GPIOC->ODR ^= (1U << 6); // using XOR logic, output is changing.
}

void ButtonInit(){

	RCC->IOPENR |= (1U << 0U);
	GPIOA->MODER &= ~(3U << 0);
	GPIOA->PUPDR &= ~(3U << 0); // GPIO port pull-up/pull-down register
	GPIOA->PUPDR |= (1U << 0);

	RCC->APBENR2 |= (1U<<0); // SYSCFGRST: SYSCFG, COMP and VREFBUF reset
	/* EXTI Rising Trigger Selection Register 1
	 * Each bit enables/disables the rising edge trigger for the event and
interrupt on the
	corresponding line.
	 */
	EXTI->RTSR1 |= (1U<<0);
	/* EXTI Interrupt Mask Register 1
	 * Each bit enables/disables the rising edge trigger for the event and
interrupt on the
	corresponding line.
	 */
	EXTI->IMR1 |= (1U<<0);
```

```
	EXTI->FTSR1 |= (1U<<0); // EXTI falling trigger selection register 1
	EXTI->RTSR1 &= ~(1U<<0);

	NVIC_SetPriority(EXTI0_1_IRQn, 0);
	NVIC_EnableIRQ(EXTI0_1_IRQn);
}

void EXTI0_1_IRQHandler(void){
	flag=1; // when button pressed, flag sets 1.
	EXTI->FPR1 |= (1<<0); // EXTI Falling Pending Register 1
}
```

2.3. Problem III

2.3.1. Flow Chart and Schematic Diagram

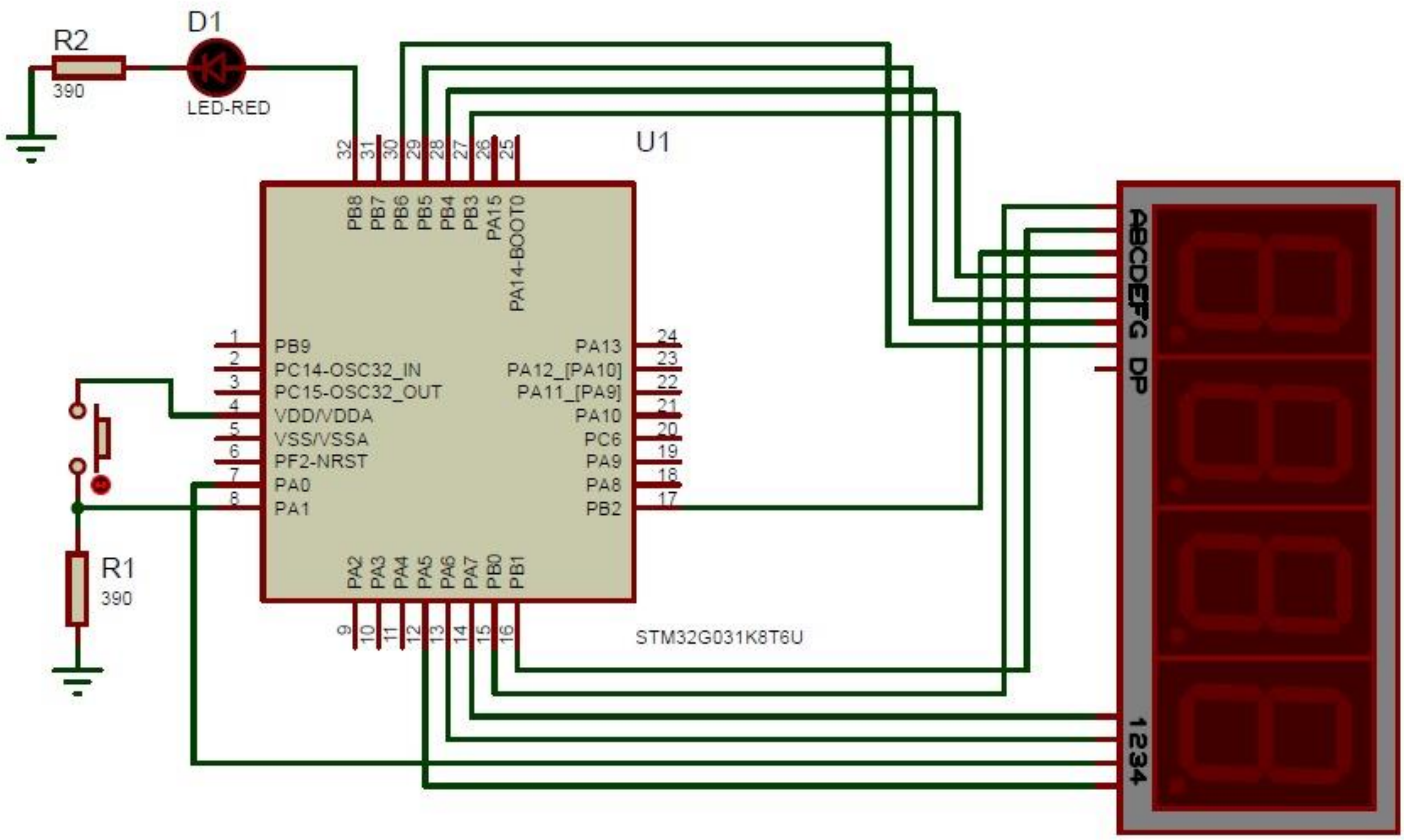

Figure 3 – schematic diagram for Problem III

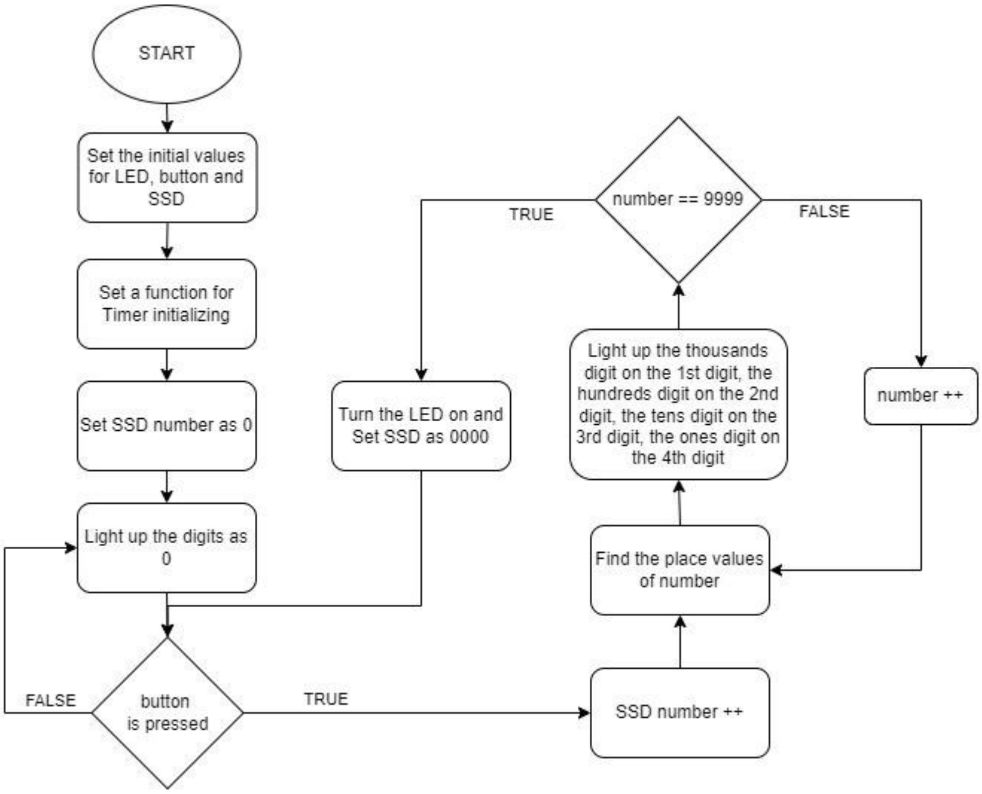

Figure 4 – flow chart for Problem III

2.3.2. C Code of the Problem III

```
/* author: Umut Mehmet Erdem | Arda Derici | Serdar Başyemenici
* problem3.c
*/
#include "stm32g0xx.h"

void clearSSD(void){ // Clear display
	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){ // choose number we want and its leds are turned on.
	clearSSD();
	switch(x){
		case 0:
			GPIOB->ODR &= ~(0x3F); // A,B,C,D,E,F is on
			break;
		case 1:
			GPIOB->ODR &= ~(0x6); // B,C is on
			break;
		case 2:
			GPIOB->ODR &= ~(0x5B); // A,B,D,E,G is on
			break;
		case 3:
			GPIOB->ODR &= ~(0x4F); // A,B,C,D,G is on
			break;
		case 4:
			GPIOB->ODR &= ~(0x66); // B,C,F,G is on
			break;
```

```
            case 5:
                  GPIOB->ODR &= ~(0x6D); // A,C,D,F,G is on
                  break;
            case 6:
                  GPIOB->ODR &= ~(0x7D); // A,C,D,E,F,G is on
                  break;
            case 7:
                  GPIOB->ODR &= ~(0x7); // A,B,C is on
                  break;
            case 8:
                  GPIOB->ODR &= ~(0x7F); // A,B,C,D,E,F,G is on
                  break;
            case 9:
                  GPIOB->ODR &= ~(0x6F); //A,B,C,D,F,G is on; E is off
                  break;
      }
}

void counter(void){
      SetZero(); // leds show us 0000 value.
      delay(1000000);
      for(int i=0; i<=9999; i++){ // count up timer until 10000.
            int thousand, hundred, decimal, unit;
            thousand=(i/1000); // thousand digit of i
            hundred=((i-thousand*1000)/100); // hundred digit of i
            decimal=((i- thousand*1000 - hundred*100)/10); // decimal digit of i
            unit=(i- thousand*1000 - hundred*100 - decimal*10); // unit digit of i

            /* unit digit we want is set to 1 and the others are set to 0*/
            GPIOA ->ODR &= ~(1U << 7); // off D1 - PA7
            GPIOA ->ODR &= ~(1U << 6); // off D2 - PA6
            GPIOA ->ODR &= ~(1U << 0); // off D3 - PA0
            GPIOA ->ODR |= (1U << 5); // on D4 - PA5
            setSSD(unit);
            delay(300);

            /* decimal digit we want is set to 1 and the others are set to 0*/
            GPIOA ->ODR &= ~(1U << 7); // D1 - PA7
            GPIOA ->ODR &= ~(1U << 6); // D2 - PA7
            GPIOA ->ODR |= (1U << 0); // D3 - PA7
            GPIOA ->ODR &= ~(1U << 5);
            setSSD(decimal);
            delay(300);

            /* hundred digit we want is set to 1 and the others are set to 0*/
            GPIOA ->ODR &= ~(1U << 7); // D1 - PA7
            GPIOA ->ODR |= (1U << 6); // D2 - PA7
            GPIOA ->ODR &= ~(1U << 0); // D3 - PA7
            GPIOA ->ODR &= ~(1U << 5);
            setSSD(hundred);
            delay(300);

            /* thousand digit we want is set to 1 and the others are set to 0*/
            GPIOA ->ODR |= (1U << 7); // D1 - PA7
            GPIOA ->ODR &= ~(1U << 6); // D2 - PA7
            GPIOA ->ODR &= ~(1U << 0); // D3 - PA7
            GPIOA ->ODR &= ~(1U << 5);
            setSSD(thousand);
            delay(300);
      }

      GPIOB->ODR |= (1U << 8); // PB8 - D8 LED turn on
      delay(1000000);
      GPIOB->BRR |= (1U << 8); // led turn off
      SetZero();
      delay(1000000);
}
```

```
void ButtonInit(void){
	/* rising edge, selection register and mask register */
	// PA1 is button
	EXTI->RTSR1 |= (1U << 1); // Rising Trigger Selection Register
	EXTI->EXTICR[0] |= (0U << 8*1); // External Interrupt Configuration Register |
for port selection
	EXTI->IMR1 |= (1U << 1); // Interrupt Mask Register

	/* enable NVIC and set interrupt priority */
	NVIC_SetPriority(EXTI0_1_IRQn, 0);
	NVIC_EnableIRQ(EXTI0_1_IRQn);
}

void EXTI0_1_IRQHandler(void){ // EXTI for button
	counter(); // when button pressed, counter is started.
	EXTI->RPR1 |= (1U << 1); // Rising Pending Register
}

void GPIOA_Init(){
	/* enable required GPIOA registers and RCC register */
	/*PA7 -> D1 digit, PA6 -> D2 digit, PA0 -> D3 digit, PA5 -> D4 digit,*/
	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 GPIOB_Init(){
	/* enable required GPIOB registers and RCC register */
	/*PB0-PB6 output pins are assigned from A to G respectively*/
	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 SetZero(){
	GPIOA ->ODR |= (1U << 7); // D1 digit -> PA7
	GPIOA ->ODR |= (1U << 6); // D2 digit -> PA6
	GPIOA ->ODR |= (1U << 0); // D3 digit -> PA0
	GPIOA ->ODR |= (1U << 5); // D4 digit -> PA5
	setSSD(0);
}

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

int main(void) {
	GPIOA_Init();
	GPIOB_Init();
	SetZero();
	ButtonInit();

	while(1) {
	}

	return 0;
}
```

2.4. Problem IV

2.4.1. C Code of the Problem IV

```
/* author: Umut Mehmet Erdem | Arda Derici | Serdar Başyemenici
* problem4.c
*/

#include "stm32g0xx.h"

void IWDGFunction(void){
	/*IWDG status register
	 * WVU: Watchdog counter window value update
	 * RVU: Watchdog counter reload value update
	 * PVU: Watchdog prescaler value update
	 */
	IWDG->SR = 0x7;
}

int main(void) {

	/* PC6 output set
	 * turned on the LED before initializing the watchdog.
	 * This will provide a blinking effect, if the system resets and the execution
starts from the beginning.
	*/
	RCC->IOPENR = RCC_IOPENR_GPIOCEN;
	GPIOC->MODER &= ~(3U << 2*6);
	GPIOC->MODER |= (1U << 2*6);
	GPIOC->ODR |= (1U << 6);

	/* IWDG->KR : the key value 0x5555 to enable access to the IWDG_PR, IWDG_RLR
and IWDG_WINR registers
	 * Writing the key value 0xCCCC starts the watchdog
	 * IWDG->PR : Prescaler divider are written by software to select the
prescaler divider feeding the counter clock.
	 * IWDG->RLR : They are written by software to define the value to be loaded
in the watchdog counter each time the value
		0xAAAA is written in the IWDG key register (IWDG_KR). The watchdog
counter counts down from this value.
		The timeout period is a function of this value and the clock prescaler.
		IWDG->WINR : they contain the high limit of the window value to be
compared with the downcounter.
	 */

	IWDG->KR = 0x5555; // IWDG key register
	IWDG->PR = 0x5; // IWDG prescaler register - 32 devider
	IWDG->RLR = 0xFFF; // IWDG reload register - Watchdog counter reload value -
4095
	IWDG->WINR = 0xFFF; // IWDG window register - Watchdog counter window value -
4095
	IWDG->KR = 0xCCCC;

	while(1) {
		IWDGFunction();
	}

	return 0;
}
```

3. Conclusions

In ELEC 335 Lab#3, the working mechanism of interrupts and timers was understood and applied. Also, the connection and operating mechanism of the 4 digit 7 segment display was learned and implemented in practice. Worked with watchdog timers. The independent watchdog timer was set and its behavior was observed in a simple blinking example. The appropriate reset time was calculated and implemented.

4. References

[1] https://github.com/fcayci/stm32g0

[2] https://www.st.com/resource/en/reference_manual/rm0444-stm32g0x1-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

[3] https://www.st.com/resource/en/datasheet/stm32g031k8.pdf

[4] https://www.st.com/resource/en/schematic_pack/mb1455-g031k8-c01_schematic.pdf

[5] https://www.st.com/resource/en/user_manual/um2591-stm32g0-nucleo32-board-mb1455-stmicroelectronics.pdf

[6] drawio.com