
GEBZE TECHNICAL UNIVERSITY

ELECTRONICS ENGINEERING

# ELM335

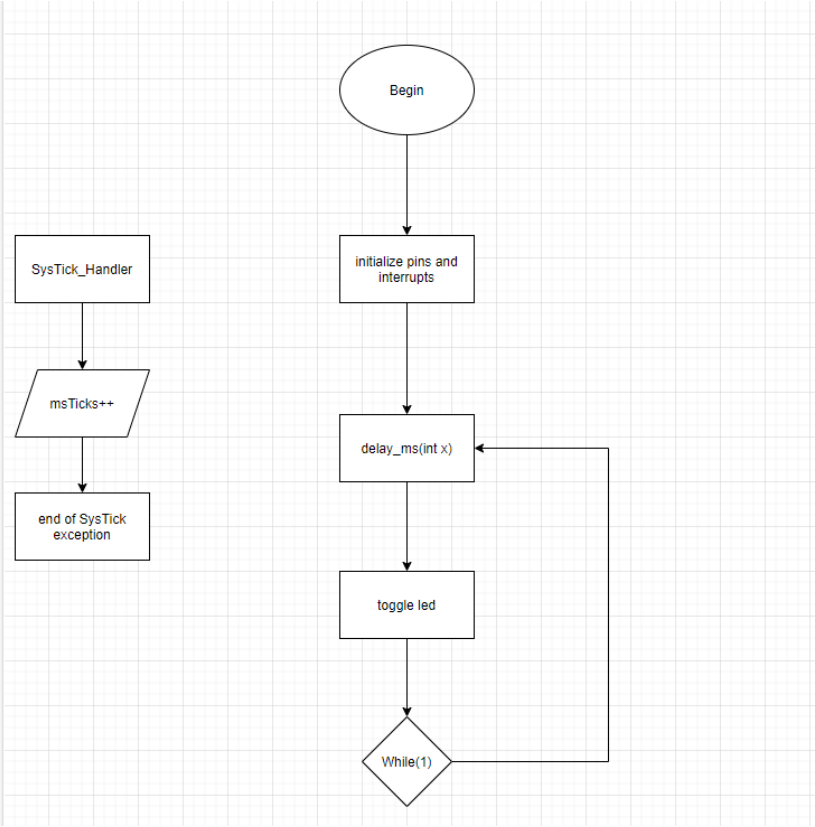
Microprocessors Laboratory

## LAB 3 Experiment Report

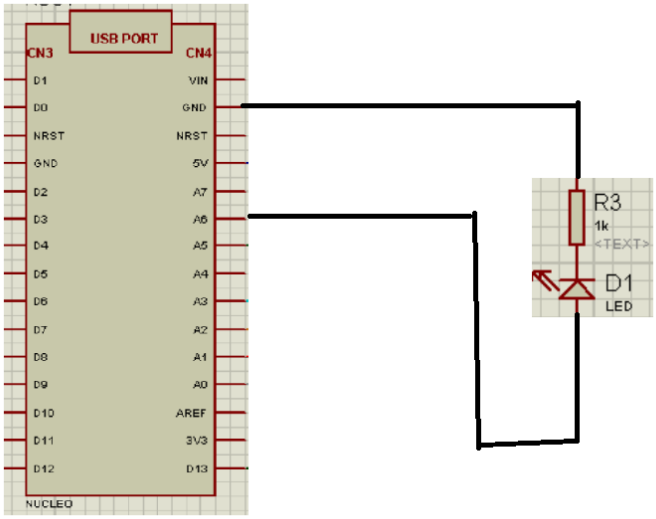

| Prepared by |
| --- |
| 151024008 - Abdürrahim Deniz KUMBARACI |
| 171024050 - Abdül Samet Karapınar |
| 171024008 - Yasin Özbek |

# Problem 1

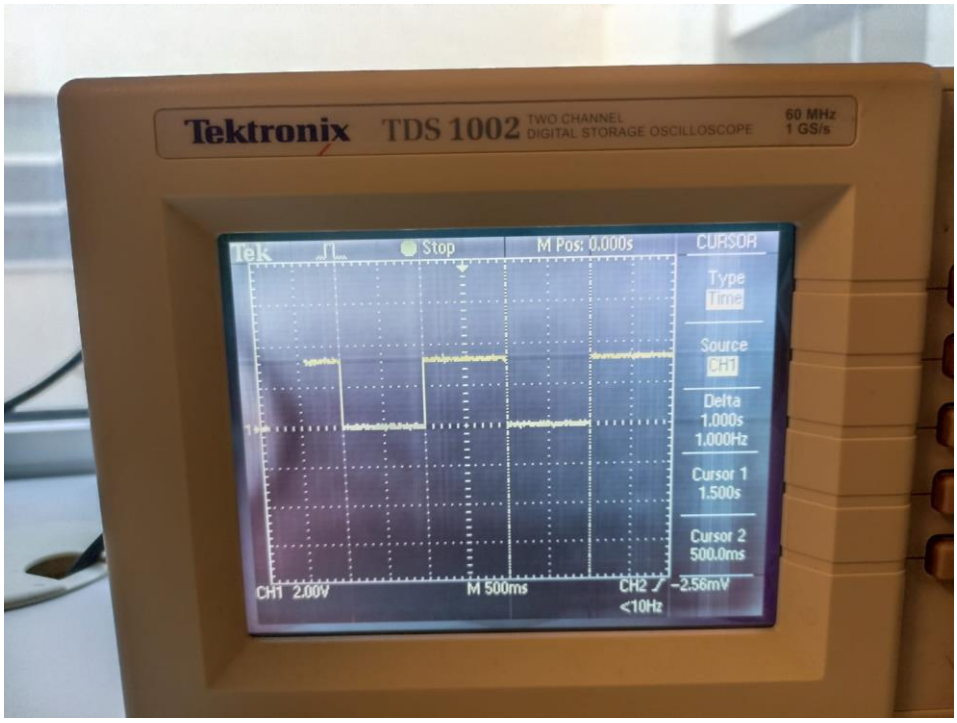
## Flow Chart

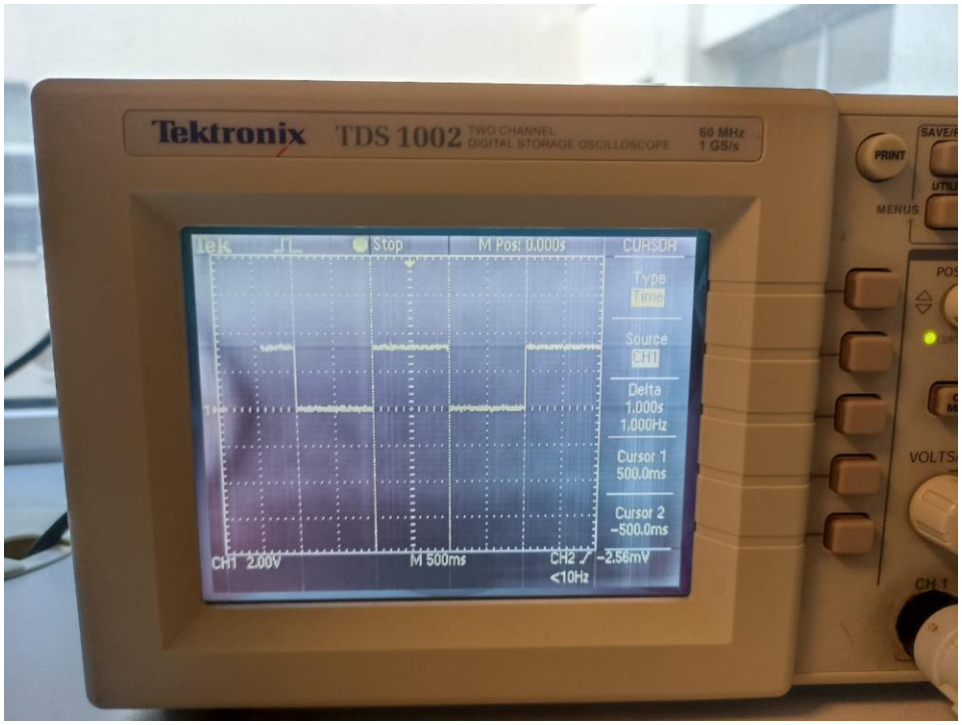



## Block Diagram



# Oscilloscope Photos



## Code

```
#include "stm32g0xx.h"

volatile uint32_t msTicks = 0;

int main(void);
void delay_ms(volatile uint32_t);

int main(void) {

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

    /* Setup PC6 as output */
    GPIOA->MODER &= ~(3U << 2*1);
    GPIOA->MODER |= (1U << 2*1);   //
                                   // output yaptik
    /* Turn on LED */
    GPIOA->ODR |= (1U << 1);

    SysTick_Config(SystemCoreClock / 1000); //Configure SysTick handler to work
per milisecond


          while(1) {
              delay_ms(1000);          //delay function set to 1000 for 1s delay

               GPIOA->ODR ^= (1U << 1); //Toggle led. XOR is used to change the
value .

          }
}
void delay_ms(uint32_t dlyTicks)//Delay ms declartion //kac ms
{
      uint32_t curTicks;               //Varriable to hold ms ticks current value

      curTicks = msTicks;
      while ((msTicks - curTicks) < dlyTicks) ;
}
void SysTick_Handler(void)
  msTicks++;
}
```

## Comments and Questions

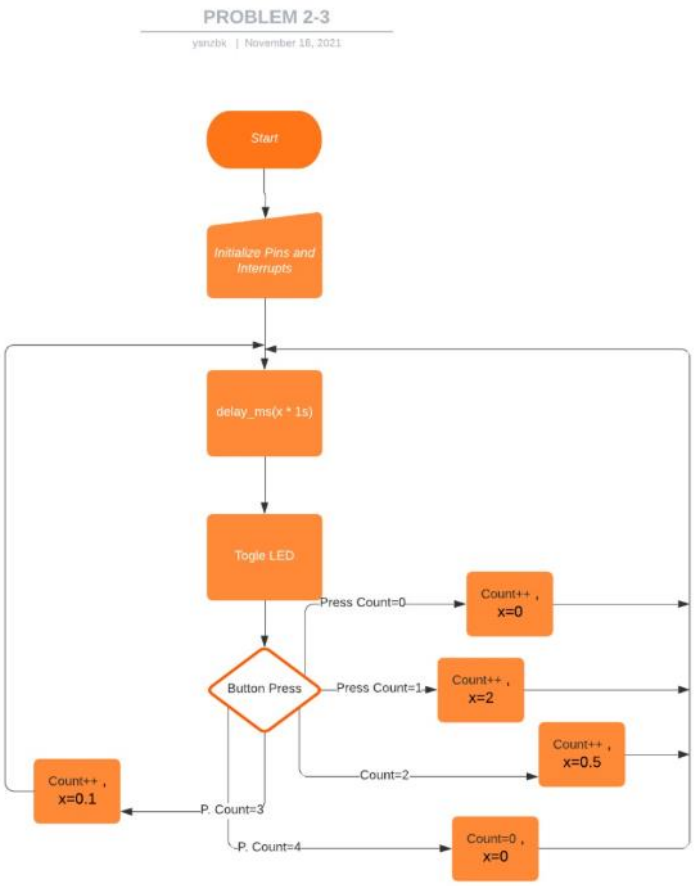
-Assembly Code size is 896 bytes

-C code size is 1240 bytes

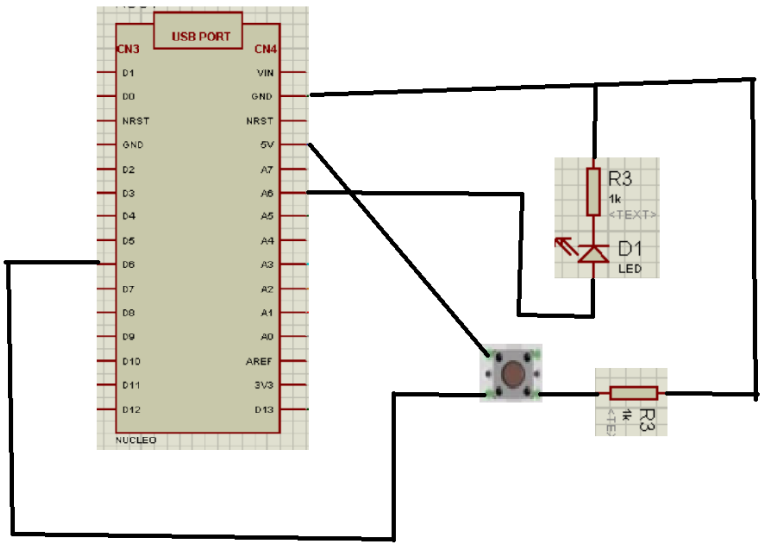
-In assembly we used  #400000 for delay but in C we use 1000 for 1 second interval

# Problem 2

## Flow Chart




## Block Diagram



## Code

```
#include "stm32g0xx.h"
#define LEDDELAY    1600000
static volatile int x = 0;  // durum degiskeni
void delay(volatile uint32_t);
int main(void) {
    /* Enable GPIOA AND GPIOB clock */
      RCC->IOPENR |= (3U << 0 );
      /* Setup PA6 as output */
    GPIOA->MODER &= ~(3U << 2*6);
    GPIOA->MODER |= (1U << 2*6);
    GPIOB->MODER &= ~(3U << 2*0);  // set PB0 as input
    /* Turn on LED */
    GPIOA->ODR &= ~(1U << 6); // led turn off
    while(1) {
         switch(x){
            case 0:
            delay(LEDDELAY);
               GPIOA->ODR &= ~(1U << 6);
               break;
            case 1:
            delay(LEDDELAY*2);   // 2 second
            GPIOA->ODR ^= (1U << 6);
            break;
            case 2:
            delay(LEDDELAY);     // 1 second
            GPIOA->ODR ^= (1U << 6);
            break;
            case 3:
              delay(LEDDELAY/2);   // 0.5 second
              GPIOA->ODR ^= (1U << 6);
              break;
            case 4:
            delay(LEDDELAY/10);  // 0.1 second
            GPIOA->ODR ^= (1U << 6);
            break;
            case 5:
            delay(LEDDELAY);
               GPIOA->ODR |= (1U << 6);
               break;
            default:
               x=0;
               break;
         } //end switch
    } //end while

    return 0;
} // end main
void delay(volatile uint32_t s)
{
    for(; s>0; s--)
    {
      if(GPIOB->IDR >> 0 == 1)
      {
            for(s=LEDDELAY/2 ; s>0; s--);
            x++;
                  if(x>5)
                        x=0;
```

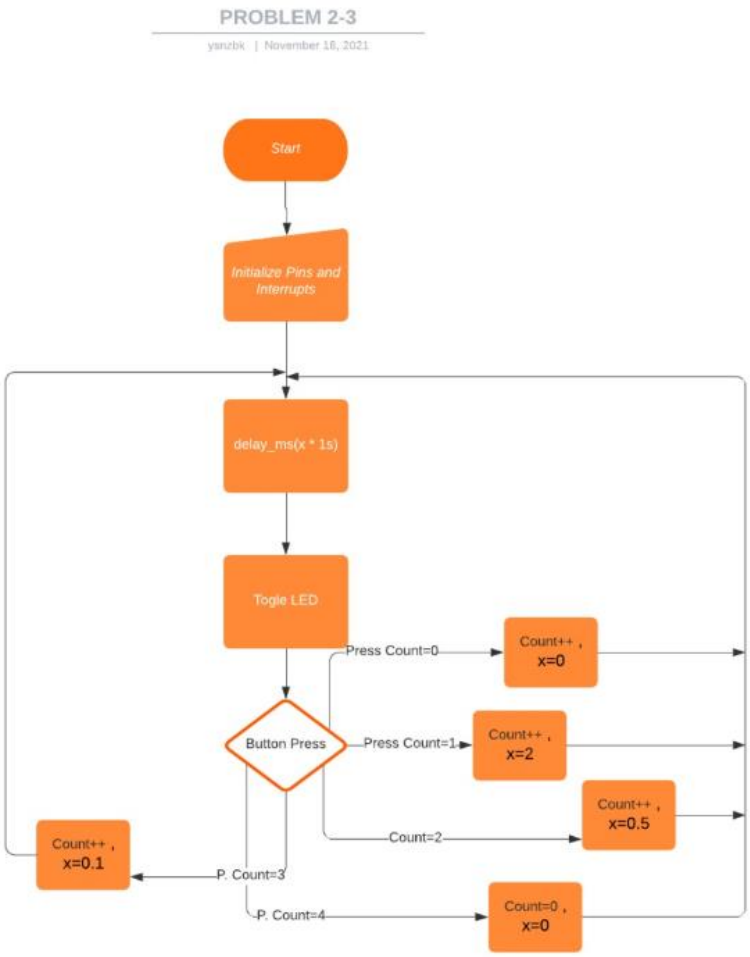
```
                        break;
            }
        }
}
```

## Comment and Questions

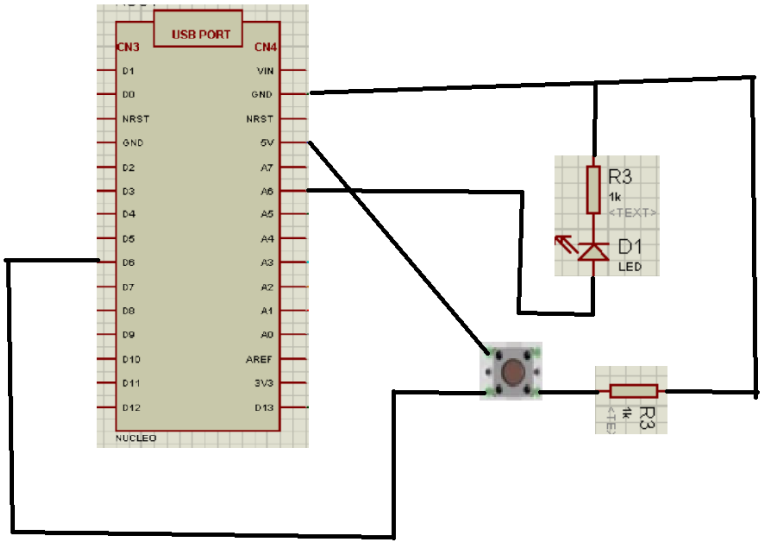
Its working fine. Interval is about 2 seconds , 1 seconds , 0.5 seconds and 0.1 seconds but we didnt understand it certainly because we couldnt use oscilloscope for this problem.

# Problem 3

## Flow Chart



## Block Diagram




## Comments and Questions

Since We used interrupt approach, While loop is much smaller in both code and instuction size.So It did not use as many instructions per cycle. And since the functions to call when the interrupt is triggered is defined seperately, it is much more cleaner in code and also more modular in nature therefore more scaleable.

## Code

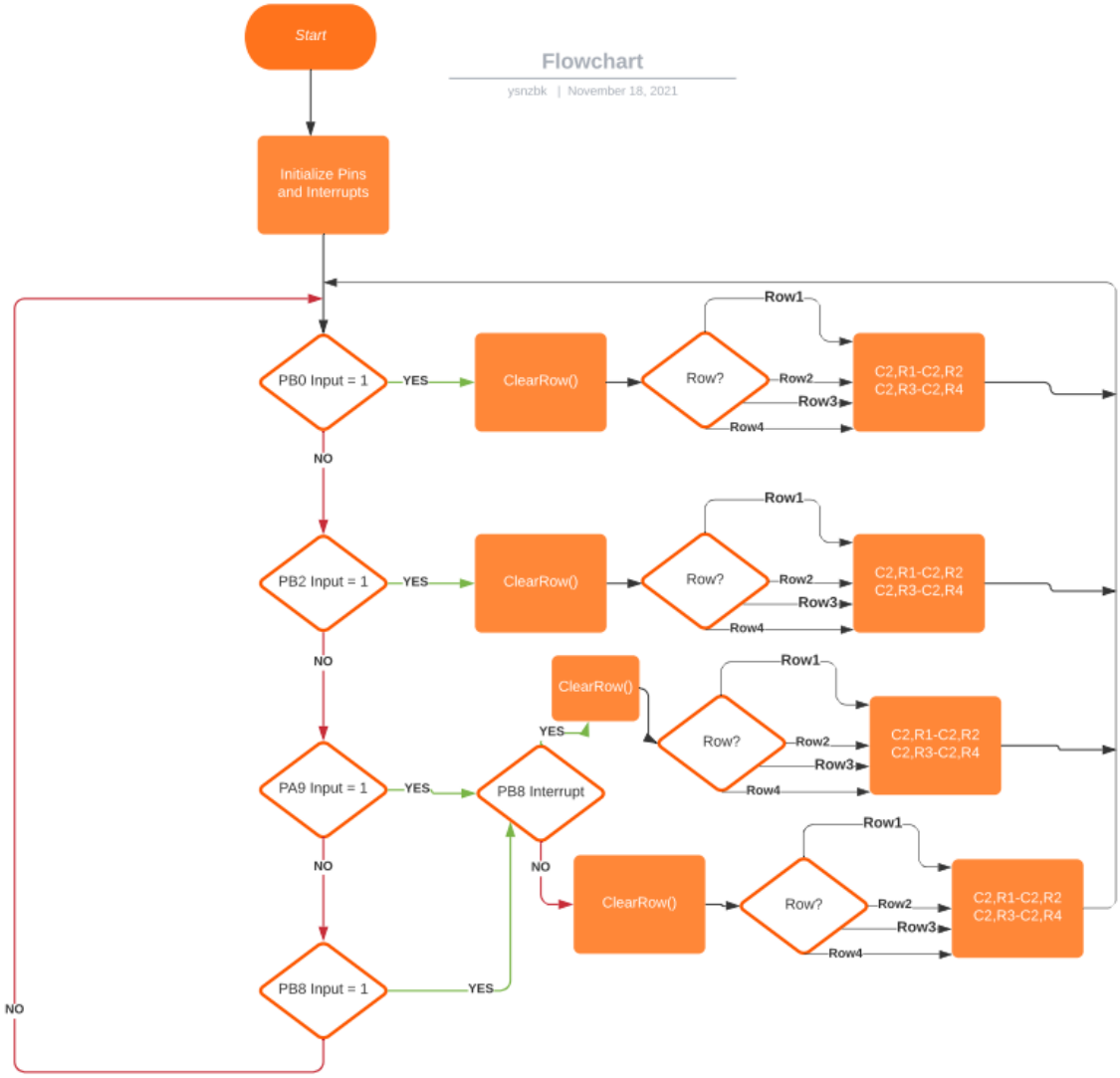
```
#include "stm32g0xx.h"
#define DELAYTWO    50000000
#define DELAYONE    25000000
#define DELAYHALF   12500000
#define DELAYTENTH 2500000
void delay(volatile uint32_t);
        void EXTI4_15_IRQHandler(void)
        {
        // DEBOUNCING ****
  // Define variables for debouncer
                        volatile char ButoonPress = 0;
                        volatile int ButtonPressDB = 0;
                        volatile int Treshold = 500;
                // Define vaiable for program mode
                        volatile int ProgramPress = 0;
                if (GPIOA->IDR & 1){
                if(ButoonPress == 0){
                        ButtonPressDB=ButtonPressDB+1;
                        if(ButtonPressDB > Treshold){
                                ProgramPress = ProgramPress+1;
                        }
                }
        }
        // Debouncing end ****
                // Led Functions
        if (ProgramPress>5){
                ProgramPress = 0;
        }
```

```
if (ProgramPress == 0){
        GPIOB->ODR ^=(1U <<4 );
}
if (ProgramPress == 1){
        GPIOB->ODR |= (1U << 4);
        delay(DELAYTWO);
        GPIOB->ODR ^=(1U << 4);
        delay(DELAYTWO);
}
if (ProgramPress == 2){
                GPIOB->ODR |= (1U << 4);
                delay(DELAYONE);
                GPIOB->ODR ^=(1U << 4);
                delay(DELAYONE)
}
if (ProgramPress == 3){
                GPIOB->ODR |= (1U << 4);
                delay(DELAYHALF);
                GPIOB->ODR ^=(1U << 4);
                delay(DELAYHALF);
        }
if (ProgramPress == 4){
                GPIOB->ODR |= (1U << 4);
                delay(DELAYTENTH);
                GPIOB->ODR ^=(1U << 4);
                delay(DELAYTENTH);
        }
if (ProgramPress == 5){
                GPIOB->ODR |= (1U << 4);
```

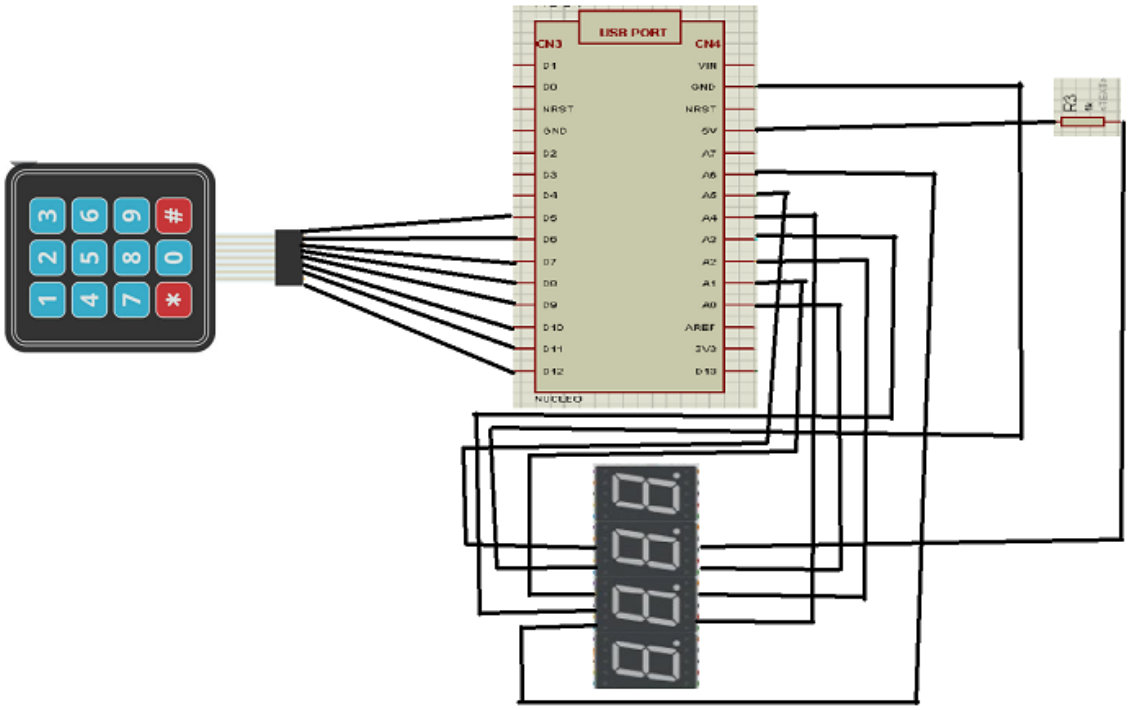
```
                    }
}
int main(void) {
/* Enable the ports for I/O */
// Function Parameters
        RCC -> IOPENR |= (1U <<0 ); //port A Clock enable
        RCC -> IOPENR |= (1U << 1); // Port B Clock Enable
        GPIOA -> MODER &= ~ (3U << 0); // PA0 register is set to input
        GPIOA -> PUPDR &= ~ (2U << 0); // Pull down for avoiding noise
        GPIOB -> MODER &= ~ (3U << 2*4);
        GPIOB -> MODER |= (1U << 2*4); // PB4 Register is set to output
        // INTERRUPTS
        EXTI -> EXTICR[0] |=(1U << 4);
        EXTI -> RTSR1 |= (1U << 4);
        EXTI -> IMR1 |= (1U << 4);
        NVIC_SetPriority(EXTI4_15_IRQn , 0);
        NVIC_EnableIRQ(EXTI4_15_IRQn);
   while(1) {
   return 0;
}
   }
   void delay(volatile uint32_t time){
      for(; time>0; time--);
   }
```

# Problem 4

## Flow Chart



## Block Diagram




## Code

```
#include "stm32g0xx.h"

void clearSSD(void);
void setSSD(int);

void clearRowsKeypad(void);
void setRowsKeypad(void);


void EXTI0_1_IRQHandler(void){
 clearRowsKeypad();

 GPIOA->ODR ^= (1U << 8);   //  PA8
 if((GPIOB->IDR >> 0) & 1){
       // #
       setSSD(10);
 }
 GPIOA->ODR ^= (1U << 8);

 GPIOB->ODR ^= (1U << 9);  // PB9
 if((GPIOB->IDR >> 0) & 1){
       // 9
       setSSD(9);
 }
 GPIOB->ODR ^= (1U << 9);
```

```
 GPIOB->ODR ^= (1U << 5);
 if((GPIOB->IDR >> 0) & 1){
       // 6
       setSSD(6);
 }
 GPIOB->ODR ^= (1U << 5);

 GPIOB->ODR ^= (1U << 4);
 if((GPIOB->IDR >> 0) & 1){
       // #
       setSSD(3);
 }
 GPIOB->ODR ^= (1U << 4);

 EXTI->RPR1 |= (1U << 0);
 setRowsKeypad();

}

void EXTI2_3_IRQHandler(void){ // interrupt from PB2
clearRowsKeypad();

GPIOA->ODR ^= (1U << 8);   //  PA8
if((GPIOB->IDR >> 0) & 1){
       // #
       setSSD(0);
}
GPIOA->ODR ^= (1U << 8);

GPIOB->ODR ^= (1U << 9);  // PB9
if((GPIOB->IDR >> 0) & 1){
       // 8
       setSSD(8);
}
GPIOB->ODR ^= (1U << 9);

GPIOB->ODR ^= (1U << 5);
if((GPIOB->IDR >> 0) & 1){
       // 5
       setSSD(5);
}
GPIOB->ODR ^= (1U << 5);

GPIOB->ODR ^= (1U << 4);
if((GPIOB->IDR >> 0) & 1){
       // 2
       setSSD(2);
}
GPIOB->ODR ^= (1U << 4);

EXTI->RPR1 |= (1U << 0);

setRowsKeypad();
```

```
}

void EXTI4_15_IRQHandler(void){
if( EXTI->RTSR1 |= (1U << 8) != 0 ){              /*interrupt from PB8 */
      clearRowsKeypad();

      GPIOA->ODR ^= (1U << 8);   //  PA8
      if((GPIOB->IDR >> 8) & 1){
            // #
            setSSD(10);
      }
      GPIOA->ODR ^= (1U << 8);

      GPIOB->ODR ^= (1U << 9);  // PB9
      if((GPIOB->IDR >> 8) & 1){
            // 9
            setSSD(7);
      }
      GPIOB->ODR ^= (1U << 9);

      GPIOB->ODR ^= (1U << 5);
      if((GPIOB->IDR >> 8) & 1){
            // 6
            setSSD(4);
      }
      GPIOB->ODR ^= (1U << 5);

      GPIOB->ODR ^= (1U << 4);
      if((GPIOB->IDR >> 8) & 1){
            // #
            setSSD(1);
      }
      GPIOB->ODR ^= (1U << 4);


      EXTI->RPR1 |= (1U << 8);  // Clear interrupt flag
      setRowsKeypad();
}

if( EXTI->RTSR1 &= (1U << 9) != 0 ){   /*interrupt from PA9 */
      clearRowsKeypad();

      GPIOA->ODR ^= (1U << 8);   //  PA8
      if((GPIOA->IDR >> 9) & 1){
            // #
            setSSD(10);
      }
      GPIOA->ODR ^= (1U << 8);

      GPIOB->ODR ^= (1U << 9);  // PB9
      if((GPIOA->IDR >> 9) & 1){
            // 9
            setSSD(10);
      }
```

```
	GPIOB->ODR ^= (1U << 9);

	GPIOB->ODR ^= (1U << 5);
	if((GPIOA->IDR >> 9) & 1){
		// 6
		setSSD(10);
	}
	GPIOB->ODR ^= (1U << 5);

	GPIOB->ODR ^= (1U << 4);
	if((GPIOA->IDR >> 9) & 1){
		// #
		setSSD(10);
	}
	GPIOB->ODR ^= (1U << 4);


	EXTI->RPR1 |= (1U << 9);  // Clear interrupt flag
	setRowsKeypad();
}



}


int main(void) {

    /* Enable GPIOC clock */
    RCC->IOPENR |= (1U << 0);
    RCC->IOPENR |= (1U << 1);

    /* Setup PC6 as output */
    GPIOA->MODER &= ~(3U << 2*8);  // PA8 D9
    GPIOA->MODER |= (1U << 2*8);

    GPIOB->MODER &= ~(3U << 2*9);  // PB9 D10
    GPIOB->MODER |= (1U << 2*9);

    GPIOB->MODER &= ~(3U << 2*5);  // PB5 D11
    GPIOB->MODER |= (1U << 2*5);

    GPIOB->MODER &= ~(3U << 2*4);  // PB4 D12
    GPIOB->MODER |= (1U << 2*4);


    GPIOA->MODER &= ~(3U << 2*9); // PA9 D5
    GPIOA->PUPDR |= (2U << 2*9);

    GPIOB->MODER &= ~(3U << 2*0); // PB0 D6
    GPIOB->PUPDR |= (2U << 2*0);

    GPIOB->MODER &= ~(3U << 2*7); // PB2 D7
    GPIOB->PUPDR |= (2U << 2*7);
```

```
GPIOB->MODER &= ~(3U << 2*8);  // PB8 D8
GPIOB->PUPDR |= (2U << 2*8);


EXTI->EXTICR[2] |= (0U << 8*1);// PA9 interrupt
EXTI->EXTICR[0] |= (1U << 0);  // PB0 interrupt
EXTI->EXTICR[0] |= (1U << 8*2);// PB2 interrupt
EXTI->EXTICR[2] |= (1U << 0);  // PB8 interrupt

EXTI->RTSR1 |= (1U << 9);
EXTI->RTSR1 |= (1U << 0);     // rising edge
EXTI->RTSR1 |= (1U << 2);
EXTI->RTSR1 |= (1U << 8);

EXTI->IMR1  |= (1U << 9);
EXTI->IMR1  |= (1U << 0);
EXTI->IMR1  |= (1U << 2);
EXTI->IMR1  |= (1U << 8);

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

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

NVIC_SetPriority(EXTI4_15_IRQn , 0);
NVIC_EnableIRQ(EXTI4_15_IRQn);



GPIOA->MODER &= ~(3U << 2*0);  // PA0 A0
GPIOA->MODER |= (1U << 2*0);

GPIOA->MODER &= ~(3U << 2*1);  // PA1 A1
GPIOA->MODER |= (1U << 2*1);

GPIOA->MODER &= ~(3U << 2*4);  // PA4 A2
GPIOA->MODER |= (1U << 2*4);

GPIOA->MODER &= ~(3U << 2*5);  // PA5 A3
GPIOA->MODER |= (1U << 2*5);

GPIOA->MODER &= ~(3U << 2*12);  // PA12 A4
GPIOA->MODER |= (1U << 2*12);

GPIOA->MODER &= ~(3U << 2*11);  // PA11 A5
GPIOA->MODER |= (1U << 2*11);

GPIOA->MODER &= ~(3U << 2*6);  // PA6 A6
GPIOA->MODER |= (1U << 2*6);
```

```
    // Tum Satirlari Hazirla
    GPIOA->ODR |= (1U << 8);
    GPIOB->ODR |= (1U << 9);
    GPIOB->ODR |= (1U << 5);
    GPIOB->ODR |= (1U << 4);


    clearSSD();

    while(1) {

    }

    return 0;
}

void clearSSD(void){
    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
  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 << 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;
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


    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 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


                  GPIOA->ODR |= (1U << 12); // PA12 E

                break;
              case 10:
                GPIOA->ODR &= ~(1U << 6); //PA6 G
                break;


      }
}

void clearRowsKeypad(void){

    GPIOA->ODR &= ~(1U << 8);
    GPIOB->ODR &= ~(1U << 9);
    GPIOB->ODR &= ~(1U << 5);
    GPIOB->ODR &= ~(1U << 4);

}

void setRowsKeypad(void){

    GPIOA->ODR |= (1U << 8);
    GPIOB->ODR |= (1U << 9);
    GPIOB->ODR |= (1U << 5);
    GPIOB->ODR |= (1U << 4);
}
```

## Comments and Questions

In this problem we tried to solve problem but we did just for the 3,6,9 and #. Unfortunately its not working correctly.