LAB: GPIO Digital InOut

Date: 2022-10-02

Author/Partner: DongMin Kim / SeongJun Park

Github: https://github.com/DongminKim21800064/EC_dmkim-064

Demo Video: https://www.youtube.com/watch?v=ipuXnO2wPJw

Introduction

In this lab, you are required to create a simple program that toggle multiple LEDs with a push-button input. Create HAL drivers for GPIO digital in and out control and use your library.

Requirement

Hardware

- MCU
  - NUCLEO-F401RE
- Actuator/Sensor/Others:
  - LEDs x 4
  - Resistor 330 ohm x 4, breadboard

Software

- Keil uVision, CMSIS, EC_HAL library

Problem 1: Create EC_HAL library

Procedure

Create the library directory `\repos\EC\lib\`.

Save your header library files in this directory. See here for detail.

> DO NOT make duplicates of library files under each project folders

List of functions for Digital_In and Out .

Library file LINK such as github

ecRCC.h (provided)

```
void RCC_HSI_init(void);
void RCC_GPIOA_enable(void);
void RCC_GPIOB_enable(void);
void RCC_GPIOC_enable(void);
```

ecGPIO.h

```
void GPIO_init(GPIO_TypeDef *Port, int pin,  int mode);

void GPIO_write(GPIO_TypeDef *Port, int pin,  int output);

int  GPIO_read(GPIO_TypeDef *Port, int pin);

void GPIO_mode(GPIO_TypeDef* Port, int pin, int mode);

void GPIO_ospeed(GPIO_TypeDef* Port, int pin,  int speed);

void GPIO_otype(GPIO_TypeDef* Port, int pin,  int type);

void GPIO_pupd(GPIO_TypeDef* Port, int pin,  int pupd);
```

ecRCC.c

See Appendix

ecGPIO.c

See Appendix

Problem 2: Toggle LED with Button

Procedure

1. Create a new project under the directory `\respos\EC\LAB\`

- The project name is "**LAB_GPIO_DIO_LED".**
- Name the source file as "**LAB_GPIO_DIO_LED.c"**
- Use the example code provided here.

2. Include your library **ecGPIO.h, ecGPIO.c** in `\repos\EC\lib\`.

> You MUST write your name in the top of the source file, inside the comment section.

3. Toggle the LED by pushing button.

- Pushing button (LED ON), Pushing Button (LED OFF) and repeat

Configuration

Button (B1)	LED
Digital In	Digital Out
GPIOC, Pin 13	GPIOA, Pin 5
PULL-UP	Open-Drain, Pull-up, Medium Speed

Code

Your code goes here: https://github.com/DongminKim21800064/EC_dmkim-064/blob/main/lab/EC_LAB_GPIO_DIO_21800064_%EA%B9%80%EB%8F%99%EB%AF%BC/LAB_GPIO_DIO_LED.c

Explain your source code with necessary comments.

LAB_GPIO_DIO_LED.c

```c
#include "stm32f4xx.h"
#include "ecRCC.h"
#include "ecGPIO.h"
#include "ecSysTick.h"

#define LED_PIN     5
#define BUTTON_PIN 13

void setup(void);

int main(void) {

    // Initialiization --------------------------------------------------------
    setup();

     //Inifinite Loop ----------------------------------------------------------
    while(1){
        // When the button pressed
        if(GPIO_read(GPIOC, BUTTON_PIN) == 0){
            bit_toggle(GPIOA, LED_PIN);
        }
        // For debouncing
        delay_ms(50);
        SysTick_reset();
    }
}

// Initialiization
void setup(void)
{
    RCC_HSI_init();
    SysTick_init();
    GPIO_init(GPIOC, BUTTON_PIN, INPUT);        // calls RCC_GPIOC_enable()
    GPIO_pupd(GPIOC, BUTTON_PIN, EC_PU);        // PULL-UP

    GPIO_init(GPIOA, LED_PIN, OUTPUT);          // calls RCC_GPIOA_enable()
    GPIO_pupd(GPIOA, LED_PIN, EC_PU);           // PULL-UP
    GPIO_otype(GPIOA, LED_PIN, OPEN_DRAIN);     // OPEN-DRAIN
    GPIO_ospeed(GPIOA, LED_PIN,MEDIUM_SPEED);   // MEDIUM-SPEED
}
```

This is the main code of Problem 2. I included the necessary header file and defined the pin number of the LED and Button.

In the **setup** function, I declared GPIO functions for requiring configurations. As you can see from the code, the Button pin set PULL-UP and the LED pin set PULL-UP, OPEN-DRAIN, and MEDIUM-SPEED.

In the **main** function, I made the infinite loop. When I pressed the button, LED will toggle. The details of **bit_toggle** are below.

bit_toggle

```
void bit_toggle(GPIO_TypeDef* Port, int pin){
    (Port->ODR) ^= (1UL << pin);
}
```

Enter the GPIOC port and toggle the position of the LED pin using XOR bitwise.

Discussion

1. Find out a typical solution for software debouncing and hardware debouncing.

   There's the word bouncing, which means that when a human-touch sensor, such as a real switch button, presses this sensor, and at the moment, the on/off is repeated several times in the sensor's close contact.

   Software

   A typical solution to solve Bouncing with software is giving a time delay. For example, if the time to Bouncing when the switch is pressed is 30 ms. the time delay is given to 50 ms. there is a time delay of 20 ms after the Bouncing is over, so the Bouncing can be solved.

   Hardware

   To eliminate Bouncing(chattering) in the switch button by hardware way, I can add capacitors that can act on the switch. The timing of chattering is different depending on the type of switch and the attached instrument, so the resistance is added to each application to control the timing to prevent chattering by adjusting the time constant by RC.

2. What method of debouncing did this NUCLEO board used for the push-button(B1)?

   The NUCLEO board used a method of solving Bouncing by giving time delay to software. I gave a delay to the main function using the functions "ecSysTick.c" and "ecSysTick.h" given in the class.

Problem 3: Toggle multi-LED with Button

Procedure

1. Create a new project under the directory `\respos\EC\LAB\`

- The project name is “**LAB_GPIO_DIO_multiLED”.**
- Name the source file as “**LAB_GPIO_DIO_multiLED.c”**

> You MUST write your name in the top of the source file, inside the comment section.

1. Include your library **ecGPIO.h, ecGPIO.c** in `\repos\lib\`.
2. Connect 4 LEDs externally with necessary load resistors.
   - As Button B1 is Pressed, light one LED at a time, in sequence.

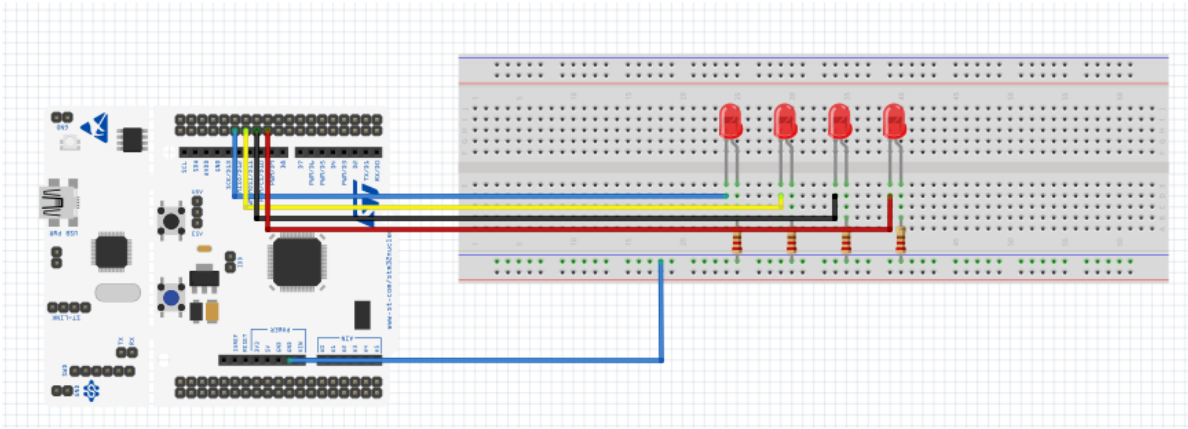
- Example: LED0--> LED1--> ...LED3--> ...LED0....

Configuration

Button (B1)	LED
Digital In	Digital Out
GPIOC, Pin 13	PA5, PA6, PA7, PB6
PULL-UP	Push-Pull, Pull-up, Medium Speed

Circuit Diagram

Circuit diagram



NUCLEO-F411RE with bread board for **Problem3** by Fritzing.

Code

Your code goes here: https://github.com/DongminKim21800064/EC_dmkim-064/blob/main/lab/EC_LAB_GPIO_DIO_21800064_%EA%B9%80%EB%8F%99%EB%AF%BC/LAB_GPIO_DIO_Multiple_LED.c

Explain your source code with necessary comments.

LAB_GPIO_DIO_Multiple_LED.c

```
#include "stm32f4xx.h"
#include "ecRCC.h"
#include "ecGPIO.h"
#include "ecSysTick.h"

void setup(void);

int main(void) {
     uint32_t i=0;
   // Initialiization --------------------------------------------------------
   multipleLED_init();
```

```
   // Inifinite Loop ------------------------------------------------------------
   while(1){
      if(GPIO_read(GPIOC, BUTTON_PIN) == 0){
         i++;
         if(i==5) i = 0;
            }
   multipleLED(i);
        }
}
```

First of all, I initialized the setting of multiple LED. In the infinite loop, when the button is pressed, integer type **i** will increase. The function **multipleLED** get the value of **i**, then sequentially LED will turn on. We just use 0 to 4 in **i** . Therefore when the i==5, reset the **i** to 0.

multipleLED_init

```
void multipleLED_init(void)
{
    RCC_HSI_init();
    SysTick_init();

    GPIO_init(GPIOC, BUTTON_PIN, INPUT);  // calls ButtonPin_enable()
    GPIO_init(GPIOA, 5, OUTPUT);          // calls LED_enable()
    GPIO_init(GPIOA, 6, OUTPUT);
    GPIO_init(GPIOA, 7, OUTPUT);
    GPIO_init(GPIOB, 6, OUTPUT);

    // Digital in ----------------------------------------------------------------
    GPIO_pupd(GPIOC, BUTTON_PIN, EC_PU);

    // Digital out ---------------------------------------------------------------
    GPIO_pupd(GPIOA, 5, EC_PU);
    GPIO_otype(GPIOA, 5, PUSH_PULL);
    GPIO_ospeed(GPIOA, 5, MEDIUM_SPEED);

    GPIO_pupd(GPIOA, 6, EC_PU);
    GPIO_otype(GPIOA, 6, PUSH_PULL);
    GPIO_ospeed(GPIOA, 6, MEDIUM_SPEED);

    GPIO_pupd(GPIOA, 7, EC_PU);
    GPIO_otype(GPIOA, 7, PUSH_PULL);
    GPIO_ospeed(GPIOA, 7, MEDIUM_SPEED);

    GPIO_pupd(GPIOB, 6, EC_PU);
    GPIO_otype(GPIOB, 6, PUSH_PULL);
    GPIO_ospeed(GPIOB, 6, MEDIUM_SPEED);

}
```

To require the Configuration, set the all pins Push-Pull, Pull-up and Medium Speed.

multipleLED

```
void multipleLED(uint32_t  num){
```

```
    int count = 0;
    int number[5][4] = {
        {0,0,0,0},
        {1,0,0,0},
        {0,1,0,0},
        {0,0,1,0},
        {0,0,0,1},
    };
        GPIO_write(GPIOA, 5, number[num][0]);
        GPIO_write(GPIOA, 6, number[num][1]);
        GPIO_write(GPIOA, 7, number[num][2]);
        GPIO_write(GPIOB, 6, number[num][3]);

        delay_ms(50);

        count++;
        if (count >10) count =0;
        SysTick_reset();
}
```

To reduce the code data, I declared the structure type "number".

This code set 5 modes.

When the num == 0, all LEDs are turned off.

When the num == 1, PA5 LED is turned on and others turned off .

When the num == 2, PA6 LED is turned on and others turned off .

When the num == 3, PA7 LED is turned on and others turned off .

When the num == 4, PB6 LED is turned on and others turned off .

Results and Analysis

Results

Experiment images and results

> Show experiment images /results

Demo Video

https://www.youtube.com/watch?v=ipuXnO2wPJw

Analysis

In this experiment, I try LED toggle using Nucleo F411RE.
First, when I press the button once, I will execute the code that the LED turns off and on and toggles. The button is set to Input, the LED is set to Output, the button is set to Pull-up, the LED is set to Pull-up, Open drain, and Medium Speed.
Press Reset Mode to begin. Every time I press the button, I can see that the LED is off and on.
I intentionally caused a delay in the code. This is to prevent the **Bouncing** in which the button switch is turned on and off at high-speed several times in a short period of time when it attaches to or falls into a mechanical contact.

The second thing is LAB multi LED. In the first experiment, I only looked at the LEDs connected to the PA5. This time, I will turn on and off the four LEDs sequentially.
As in the first experiment, I proceed with PULL-UP button and push-Pull, pull-up, and medium speed LED. Nothing turns on in the Reset state. You can see that it turns on one by one.

Discussion

1. Find out a typical solution for software debouncing and hardware debouncing. What method of debouncing did this NUCLEO board used for the push-button(B1)?

- There's the word bouncing, which means that when a human-touch sensor, such as a real switch button, presses this sensor, and at the moment, the on/off is repeated several times in the sensor's close contact.

Software

- A typical solution to solve Bouncing with software is giving a time delay. For example, if the time to Bouncing when the switch is pressed is 30 ms. the time delay is given to 50 ms. there is a time delay of 20 ms after the Bouncing is over, so the Bouncing can be solved.

Hardware

- To eliminate Bouncing(chattering) in the switch button by hardware way, I can add capacitors that can act on the switch. The timing of chattering is different depending on the type of switch and the attached instrument, so the resistance is added to each application to control the timing to prevent chattering by adjusting the time constant by RC.
- The NUCLEO board used a method of solving Bouncing by giving time delay to software. I gave a delay to the main function using the functions "ecSysTick.c" and "ecSysTick.h" given in the class.

Reference

Complete list of all references used (github, blog, paper, etc)

- https://github.com/ykkimhgu/EC-student/ lecture provided.
- https://kocoafab.cc/tutorial/view/655 for drawing circuit with "Fritzing" program.
- https://studymake.tistory.com/166 for understanding debouncing.
- https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=albert0512&logNo=221522825682 for understanding set appropriate delay time.

Troubleshooting

1. Q : I needed an accurate understanding of PULL-Pull and and OPEN DRAIN

   A : Push/Pull literally means to push up or down to the ground, which means to conduct an internal switch toward the power source or the ground.
   Open-Drain is a similar concept to TTL's Open-Collector, which creates a high value by saying that P-MOSFET is always Off and that the drain of the lower N-MosFet is OPEN.

2. Q : How to set appropriate delay time?

   A : Continue to read the current button state and the immediately preceding button state continuously. If the current state is different from the previous state (when pressed and released), save the current time in **lastDebounceTime**. Make sure that the button is still pressed after the last time the status of the button has changed by **debounceDelay**. If the button is pressed and the action has not already been executed, the action is executed.

Then, put **reading** in **buttonState** as a sign that it has been moved. If you release your hand from the button, the current time is saved again in **lastDebounceTime**. After that, check if the button is still not pressed when the **debounceDelay** has passed. If the button is not pressed continuously and the buttonState is in the HIGH state, fix the buttonState to **LOW (reading)**. Since the hand is released from the button **(reading==LOW)**, the operation is not executed.

3. Q : Check the code's compile without LED or other outputs.

   A : Enter CTRL + F5 and observe the values step by step.

Appendix

ecRCC.c

```
#include "stm32f4xx.h"
#include "ecRCC.h"

volatile int EC_SYSCLK=16000000;

void RCC_HSI_init() {
    // Enable High Speed Internal Clock (HSI = 16 MHz)
  //RCC->CR |= ((uint32_t)RCC_CR_HSION);
    RCC->CR |= 0x00000001U;

  // wait until HSI is ready
  //while ( (RCC->CR & (uint32_t) RCC_CR_HSIRDY) == 0 ) {;}
    while ( (RCC->CR & 0x00000002U) == 0 ) {;}

  // Select HSI as system clock source
  RCC->CFGR &= (uint32_t)(~RCC_CFGR_SW);                                  // not
essential
  RCC->CFGR |= (uint32_t)RCC_CFGR_SW_HSI;                                 //00:
HSI16 oscillator used as system clock

    // Wait till HSI is used as system clock source
  while ((RCC->CFGR & (uint32_t)RCC_CFGR_SWS) != 0 );

    //EC_SYSTEM_CLK=16000000;
        //EC_SYSCLK=16000000;
        EC_SYSCLK=16000000;
}

void RCC_PLL_init() {
    // To correctly read data from FLASH memory, the number of wait states
(LATENCY)
  // must be correctly programmed according to the frequency of the CPU clock
  // (HCLK) and the supply voltage of the device.
    FLASH->ACR &= ~FLASH_ACR_LATENCY;
    FLASH->ACR |=  FLASH_ACR_LATENCY_2WS;

    // Enable the Internal High Speed oscillator (HSI)
    RCC->CR |= RCC_CR_HSION;
    while((RCC->CR & RCC_CR_HSIRDY) == 0);

    // Disable PLL for configuration
```

```
    RCC->CR    &= ~RCC_CR_PLLON;

    // Select clock source to PLL
    RCC->PLLCFGR &= ~RCC_PLLCFGR_PLLSRC;        // Set source for PLL: clear
bits
    RCC->PLLCFGR |= RCC_PLLCFGR_PLLSRC_HSI; // Set source for PLL: 0 =HSI, 1 =
HSE

    // Make PLL as 84 MHz
    // f(VCO clock) = f(PLL clock input) * (PLLN / PLLM) = 16MHz * 84/8 = 168
MHz
    // f(PLL_R) = f(VCO clock) / PLLP = 168MHz/2 = 84MHz
    RCC->PLLCFGR = (RCC->PLLCFGR & ~RCC_PLLCFGR_PLLN) | 84U << 6;
    RCC->PLLCFGR = (RCC->PLLCFGR & ~RCC_PLLCFGR_PLLM) | 8U ;
    RCC->PLLCFGR &= ~RCC_PLLCFGR_PLLP;  // 00: PLLP = 2, 01: PLLP = 4, 10: PLLP
= 6, 11: PLLP = 8


    // Enable PLL after configuration
    RCC->CR   |= RCC_CR_PLLON;
    while((RCC->CR & RCC_CR_PLLRDY)>>25 != 0);

    // Select PLL as system clock
    RCC->CFGR &= ~RCC_CFGR_SW;
    RCC->CFGR |= RCC_CFGR_SW_PLL;

    // Wait until System Clock has been selected
    while ((RCC->CFGR & RCC_CFGR_SWS) != 8UL);

    // The maximum frequency of the AHB and APB2 is 100MHz,
    // The maximum frequency of the APB1 is 50 MHz.
    RCC->CFGR &= ~RCC_CFGR_HPRE;        // AHB prescaler = 1; SYSCLK not divided
(84MHz)
    RCC->CFGR &= ~RCC_CFGR_PPRE1;       // APB high-speed prescaler (APB1) = 2,
HCLK divided by 2 (42MHz)
    RCC->CFGR |=  RCC_CFGR_PPRE1_2;
    RCC->CFGR &= ~RCC_CFGR_PPRE2;       // APB high-speed prescaler (APB2) = 1,
HCLK not divided    (84MHz)

    EC_SYSCLK=84000000;
}


void RCC_GPIOA_enable()
{
    // HSI is used as system clock
    RCC_HSI_init();
    // RCC Peripheral Clock Enable Register
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;
}

void RCC_GPIOB_enable()
{
    // HSI is used as system clock
    RCC_HSI_init();
    // RCC Peripheral Clock Enable Register
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOBEN;
}
```

```
void RCC_GPIOC_enable()
{
    // HSI is used as system clock
    RCC_HSI_init();
    // RCC Peripheral Clock Enable Register
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOCEN;
}
```

ecGPIO.c

```
#include "stm32f4xx.h"
#include "stm32f411xe.h"
#include "ecGPIO.h"
#include "ecSysTick.h"

void bit_toggle(GPIO_TypeDef* Port, int pin){
    (Port->ODR) ^= (1UL << pin);
}

void GPIO_init(GPIO_TypeDef *Port, int pin, int mode){
    // mode  : Input(0), Output(1), AlterFunc(2), Analog(3)
    if (Port == GPIOA)
        RCC_GPIOA_enable();
    if (Port == GPIOB)
        RCC_GPIOB_enable();
    if (Port == GPIOC)
        RCC_GPIOC_enable();

    GPIO_mode(Port, pin, mode);

}


// GPIO Mode          : Input(00), Output(01), AlterFunc(10), Analog(11)
void GPIO_mode(GPIO_TypeDef *Port, int pin, int mode){
   Port->MODER &= ~(3UL<<(2*pin));
   Port->MODER |= mode <<(2*pin);
}


// GPIO Speed          : Low speed (00), Medium speed (01), Fast speed (10), High
speed (11)
void GPIO_ospeed(GPIO_TypeDef *Port, int pin, int speed){
      Port->OSPEEDR &=  ~(3UL<<(2*pin));
        Port->OSPEEDR |=    speed <<(2*pin);
//10:Fast Speed
}

// GPIO Output Type: Output push-pull (0, reset), Output open drain (1)
void GPIO_otype(GPIO_TypeDef *Port, int pin, int type){
        Port->OTYPER &=         ~(type<<(pin));                              //
0:Push-Pull
```

```
}

// GPIO Push-Pull    : No pull-up, pull-down (00), Pull-up (01), Pull-down (10),
Reserved (11)
void GPIO_pupd(GPIO_TypeDef *Port, int pin, int pupd){
    Port->PUPDR &=  ~(3UL<<(2*pin));
        Port->PUPDR  |=     pupd<<(2*pin);                                  //
10: Pull-UP
}

int GPIO_read(GPIO_TypeDef *Port, int pin){

    unsigned int btVal=0;
            //Read bit value of Button
            btVal=(Port->IDR>>pin) & 1UL;
    return btVal;       //[TO-DO] YOUR CODE GOES HERE
}



void GPIO_write(GPIO_TypeDef *Port, int pin, int Output){
    Port->ODR &= ~(1UL<<(pin));
  Port->ODR |= (Output <<(pin));
}


void multipleLED_init(void)
{
    RCC_HSI_init();
    SysTick_init();

    GPIO_init(GPIOC, BUTTON_PIN, INPUT);  // calls RCC_GPIOC_enable()
    GPIO_init(GPIOA, 5, OUTPUT);    // calls RCC_GPIOA_enable()
    GPIO_init(GPIOA, 6, OUTPUT);
    GPIO_init(GPIOA, 7, OUTPUT);
    GPIO_init(GPIOB, 6, OUTPUT);

    // Digital in ------------------------------------------------------------
    GPIO_pupd(GPIOC, BUTTON_PIN, EC_PU);

    // Digital out -----------------------------------------------------------
    GPIO_pupd(GPIOA, 5, EC_PU);
    GPIO_otype(GPIOA, 5, PUSH_PULL);
    GPIO_ospeed(GPIOA, 5, MEDIUM_SPEED);

    GPIO_pupd(GPIOA, 6, EC_PU);
    GPIO_otype(GPIOA, 6, PUSH_PULL);
    GPIO_ospeed(GPIOA, 6, MEDIUM_SPEED);

    GPIO_pupd(GPIOA, 7, EC_PU);
    GPIO_otype(GPIOA, 7, PUSH_PULL);
    GPIO_ospeed(GPIOA, 7, MEDIUM_SPEED);

    GPIO_pupd(GPIOB, 6, EC_PU);
    GPIO_otype(GPIOB, 6, PUSH_PULL);
    GPIO_ospeed(GPIOB, 6, MEDIUM_SPEED);

}
```

```
void multipleLED(uint32_t  num){
    int count = 0;
    int number[5][4] = {
        {0,0,0,0}, // all LEDs are turned off
        {1,0,0,0}, // PA5 LED is turned on and others turned off
        {0,1,0,0}, // PA6 LED is turned on and others turned off
        {0,0,1,0}, // PA7 LED is turned on and others turned off
        {0,0,0,1}, // PB6 LED is turned on and others turned off
    };
        GPIO_write(GPIOA, 5, number[num][0]);
        GPIO_write(GPIOA, 6, number[num][1]);
        GPIO_write(GPIOA, 7, number[num][2]);
        GPIO_write(GPIOB, 6, number[num][3]);

        delay_ms(50);

        count++;
        if (count >10) count =0;
        SysTick_reset();
}

void sevensegment_init(void){

    GPIO_init(GPIOA, 8, OUTPUT);        // A
    GPIO_init(GPIOB, 10, OUTPUT);       // B
    GPIO_init(GPIOA, 7, OUTPUT);        // C
    GPIO_init(GPIOA, 6, OUTPUT);        // D
    GPIO_init(GPIOA, 5, OUTPUT);        // E
    GPIO_init(GPIOA, 9, OUTPUT);        // F
    GPIO_init(GPIOC, 7, OUTPUT);        // G
    GPIO_init(GPIOB, 6, OUTPUT);        // DP

    //Set PULL-UP Mode
    GPIO_pupd(GPIOC, BUTTON_PIN, EC_PU);            // PULL-UP

}

void sevensegment_decode(uint8_t  num){

    // 7-segments Reversed TruthTable
        int number[10][8] = {
                                    // A B C D E F G DP
                                        {0,0,0,0,0,0,1,1},          //zero
                    {1,0,0,1,1,1,1,1},          //one
                    {0,0,1,0,0,1,0,1},          //two
                    {0,0,0,0,1,1,0,1},          //three
                    {1,0,0,1,1,0,0,1},          //four
                    {0,1,0,0,1,0,0,1},          //five
                    {0,1,0,0,0,0,0,1},          //six
                    {0,0,0,1,1,1,1,1},          //seven
                    {0,0,0,0,0,0,0,1},          //eight
                    {0,0,0,0,1,0,0,1},          //nine
                        };

        GPIO_write(GPIOA, 8, number[num][0]);  // A
        GPIO_write(GPIOB, 10, number[num][1]); // B
```

```
        GPIO_write(GPIOA, 7, number[num][2]);  // C
        GPIO_write(GPIOA, 6, number[num][3]);  // D
        GPIO_write(GPIOA, 5, number[num][4]);  // E
        GPIO_write(GPIOA, 9, number[num][5]);  // F
        GPIO_write(GPIOC, 7, number[num][6]);  // G
        GPIO_write(GPIOB, 6, number[num][7]);  // DP


    }
```