# LAB: Digital In/Out - LED toggle with Push-Button

Date: 2021-10-07

Name(ID):Park JeongWoo

Partner Name:Lee JunGi

# I. 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 these APIs for the lab.

### Hardware

NUCLEO -F411RE

LEDs x 3, Resistor 330 ohm x 3, breadboard

### Software

Keil uVision IDE, CMSIS, EC_HAL

# II. Procedure

## Part 1. Create EC_HAL driver

Below are the examples of functions for Digital In and Out.

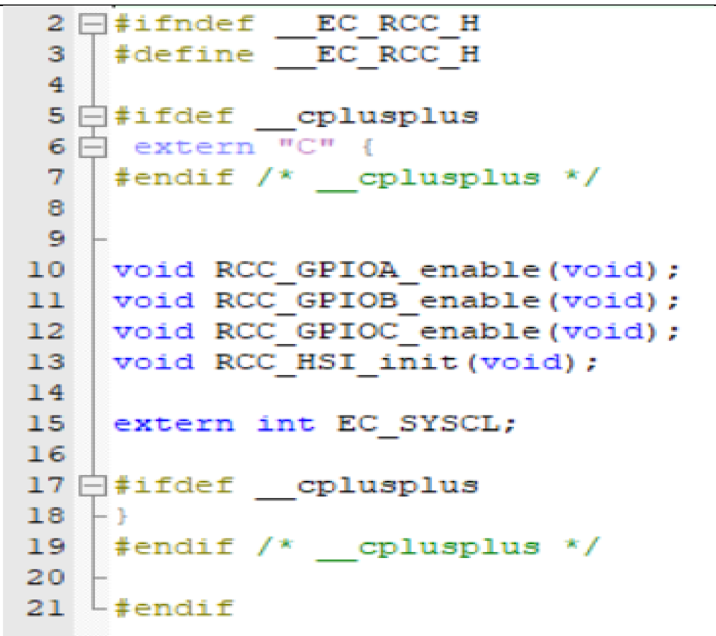
| Include File | Function |
|---|---|
| **ecRCC.h, c** | void RCC_HSI_init(void);<br>void RCC_GPIOA_enable(void); // This can go inside GPIO_init()<br>void RCC_GPIOB_enable(void);<br>void RCC_GPIOC_enable(void); |

**ecGPIO.h, c**

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_pudr(GPIO_TypeDef* Port, int pin, int pudr);

## Souce code

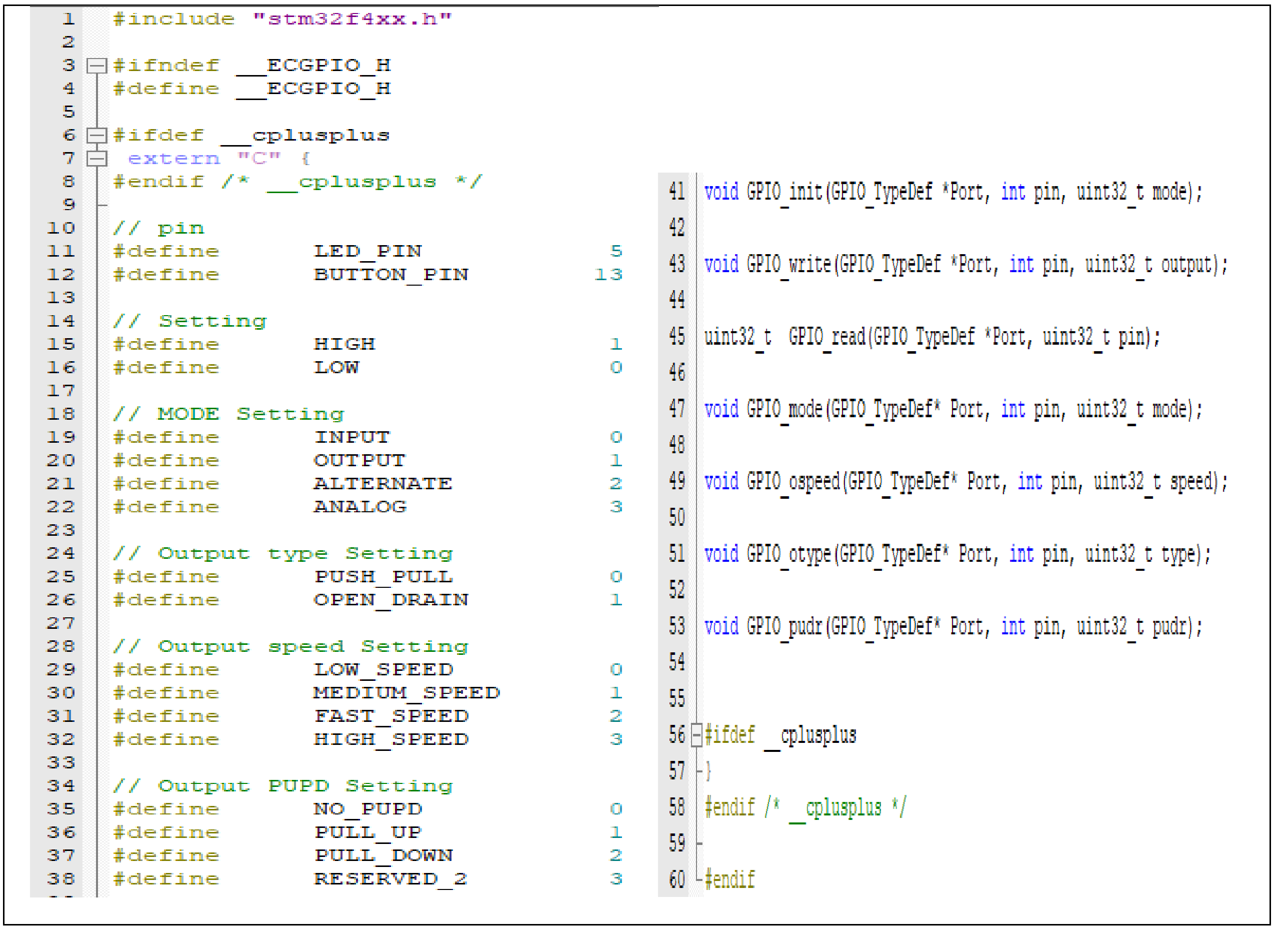
### ecRCC.h



```c
#ifndef __EC_RCC_H
#define __EC_RCC_H

#ifdef __cplusplus
 extern "C" {
#endif /* __cplusplus */


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

extern int EC_SYSCL;

#ifdef __cplusplus
}
#endif /* __cplusplus */

#endif
```

**ecRCC.c** :   See Appendix

### ecGPIO.h

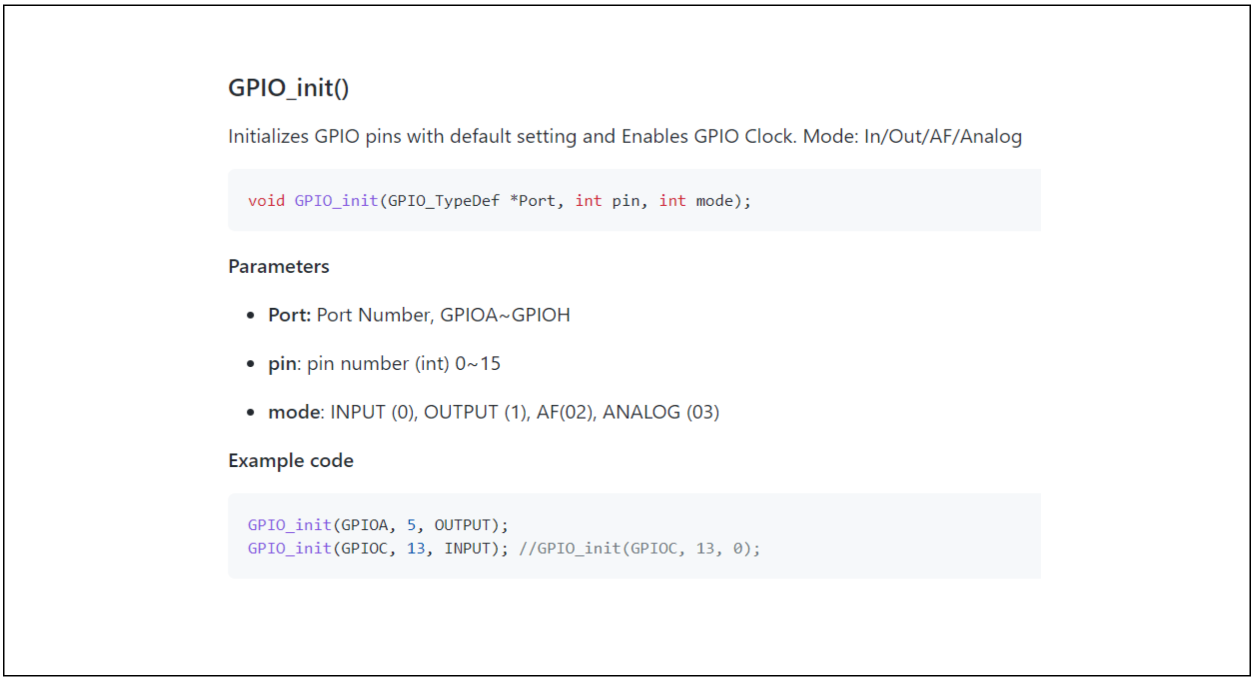


```
#include "stm32f4xx.h"

#ifndef __ECGPIO_H
#define __ECGPIO_H

#ifdef __cplusplus
 extern "C" {
#endif /* __cplusplus */

// pin
#define         LED_PIN             5
#define         BUTTON_PIN         13

// Setting
#define         HIGH                1
#define         LOW                 0

// MODE Setting
#define         INPUT               0
#define         OUTPUT              1
#define         ALTERNATE           2
#define         ANALOG              3

// Output type Setting
#define         PUSH_PULL           0
#define         OPEN_DRAIN          1

// Output speed Setting
#define         LOW_SPEED           0
#define         MEDIUM_SPEED        1
#define         FAST_SPEED          2
#define         HIGH_SPEED          3

// Output PUPD Setting
#define         NO_PUPD             0
#define         PULL_UP             1
#define         PULL_DOWN           2
#define         RESERVED_2          3
```

```
void GPIO_init(GPIO_TypeDef *Port, int pin, uint32_t mode);

void GPIO_write(GPIO_TypeDef *Port, int pin, uint32_t output);

uint32_t  GPIO_read(GPIO_TypeDef *Port, uint32_t pin);

void GPIO_mode(GPIO_TypeDef* Port, int pin, uint32_t mode);

void GPIO_ospeed(GPIO_TypeDef* Port, int pin, uint32_t speed);

void GPIO_otype(GPIO_TypeDef* Port, int pin, uint32_t type);

void GPIO_pudr(GPIO_TypeDef* Port, int pin, uint32_t pudr);


#ifdef __cplusplus
}
#endif /* __cplusplus */

#endif
```

**ecGPIO.c** :   See Appendix

## Documenation of Library



### GPIO_init()

Initializes GPIO pins with default setting and Enables GPIO Clock. Mode: In/Out/AF/Analog

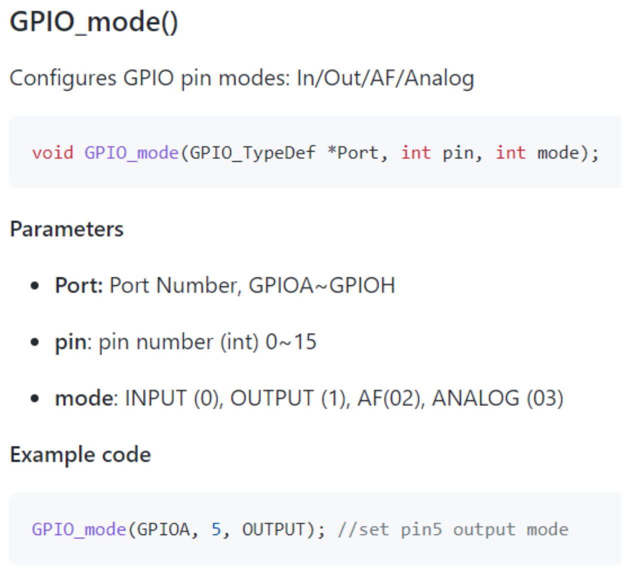
```
void GPIO_init(GPIO_TypeDef *Port, int pin, int mode);
```

**Parameters**

- **Port:** Port Number, GPIOA~GPIOH
- **pin**: pin number (int) 0~15
- **mode**: INPUT (0), OUTPUT (1), AF(02), ANALOG (03)

**Example code**

```
GPIO_init(GPIOA, 5, OUTPUT);
GPIO_init(GPIOC, 13, INPUT); //GPIO_init(GPIOC, 13, 0);
```



## GPIO_mode()

Configures GPIO pin modes: In/Out/AF/Analog

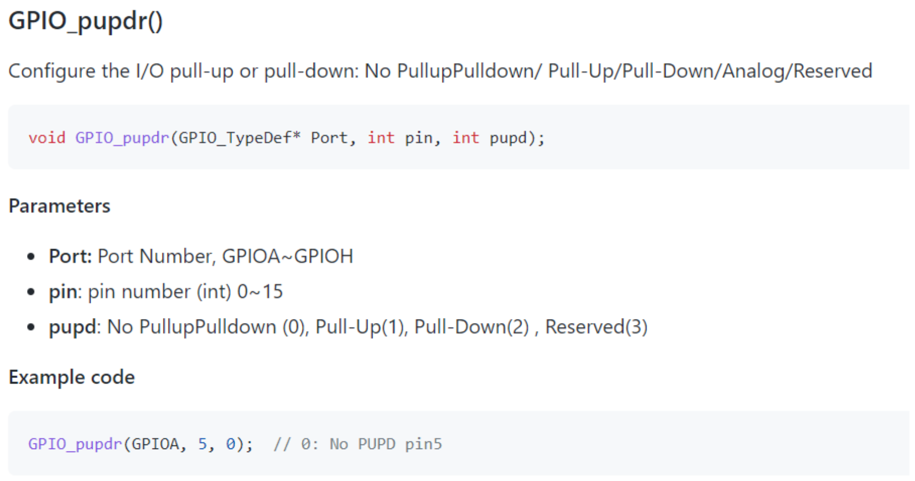
```
void GPIO_mode(GPIO_TypeDef *Port, int pin, int mode);
```

**Parameters**

- **Port:** Port Number, GPIOA~GPIOH
- **pin**: pin number (int) 0~15
- **mode**: INPUT (0), OUTPUT (1), AF(02), ANALOG (03)

**Example code**

```
GPIO_mode(GPIOA, 5, OUTPUT); //set pin5 output mode
```



## GPIO_pupdr()

Configure the I/O pull-up or pull-down: No PullupPulldown/ Pull-Up/Pull-Down/Analog/Reserved

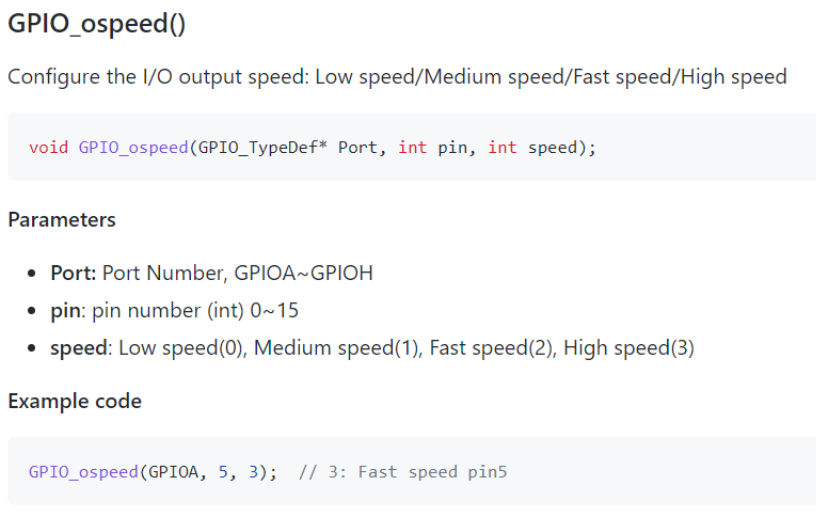
```
void GPIO_pupdr(GPIO_TypeDef* Port, int pin, int pupd);
```

**Parameters**

- **Port:** Port Number, GPIOA~GPIOH
- **pin**: pin number (int) 0~15
- **pupd**: No PullupPulldown (0), Pull-Up(1), Pull-Down(2) , Reserved(3)

**Example code**

```
GPIO_pupdr(GPIOA, 5, 0);  // 0: No PUPD pin5
```



## GPIO_ospeed()

Configure the I/O output speed: Low speed/Medium speed/Fast speed/High speed

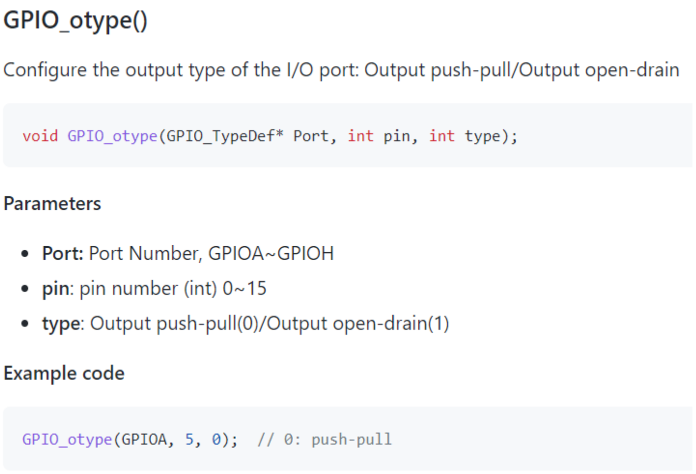
```
void GPIO_ospeed(GPIO_TypeDef* Port, int pin, int speed);
```

**Parameters**

- **Port:** Port Number, GPIOA~GPIOH
- **pin**: pin number (int) 0~15
- **speed**: Low speed(0), Medium speed(1), Fast speed(2), High speed(3)

**Example code**

```
GPIO_ospeed(GPIOA, 5, 3);  // 3: Fast speed pin5
```



### GPIO_otype()

Configure the output type of the I/O port: Output push-pull/Output open-drain

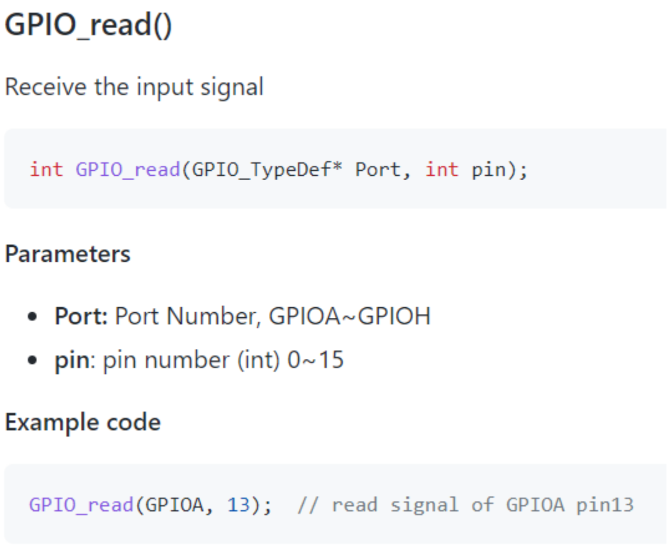
```
void GPIO_otype(GPIO_TypeDef* Port, int pin, int type);
```

**Parameters**

- **Port:** Port Number, GPIOA~GPIOH
- **pin**: pin number (int) 0~15
- **type**: Output push-pull(0)/Output open-drain(1)

**Example code**

```
GPIO_otype(GPIOA, 5, 0);  // 0: push-pull
```



### GPIO_read()

Receive the input signal

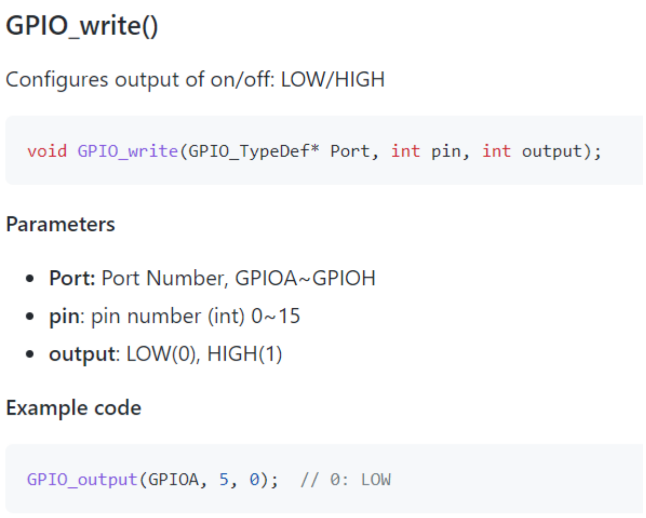
```
int GPIO_read(GPIO_TypeDef* Port, int pin);
```

**Parameters**

- **Port:** Port Number, GPIOA~GPIOH
- **pin**: pin number (int) 0~15

**Example code**

```
GPIO_read(GPIOA, 13);  // read signal of GPIOA pin13
```



### GPIO_write()

Configures output of on/off: LOW/HIGH

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

**Parameters**

- **Port:** Port Number, GPIOA~GPIOH
- **pin**: pin number (int) 0~15
- **output**: LOW(0), HIGH(1)

**Example code**

```
GPIO_output(GPIOA, 5, 0);  // 0: LOW
```

# Part 2. Toggle LED with push – button input

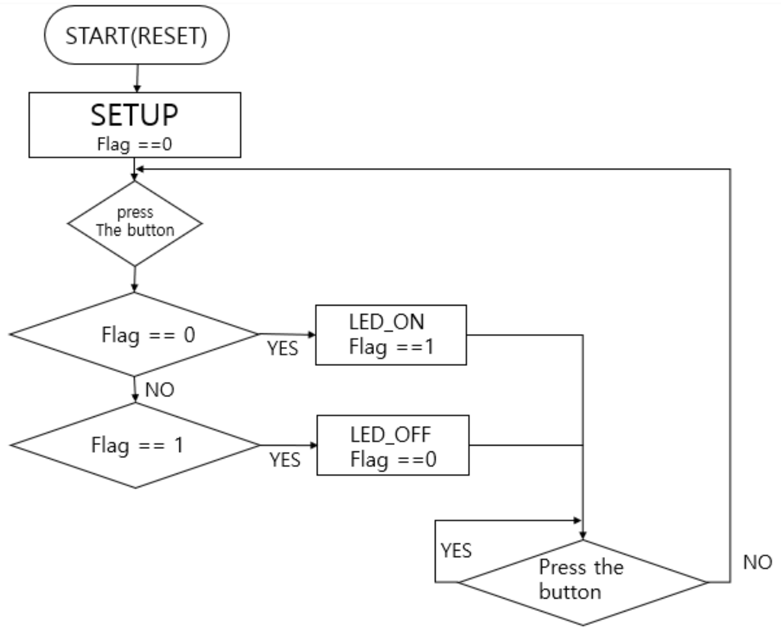
Create a new project named as “LAB_GPIO_DigitialI nOut_LED”.

Name the source file as  “LAB_GPIO_DigitialInOut_LED.c”,

## observation of the output

After compilation, pressing and releasing the button turns on the LED, and again pressing and releasing the LED turns off. It can be seen that the LED reacts  immediately and toggles as the button is pressed and released.

## Flow Chart

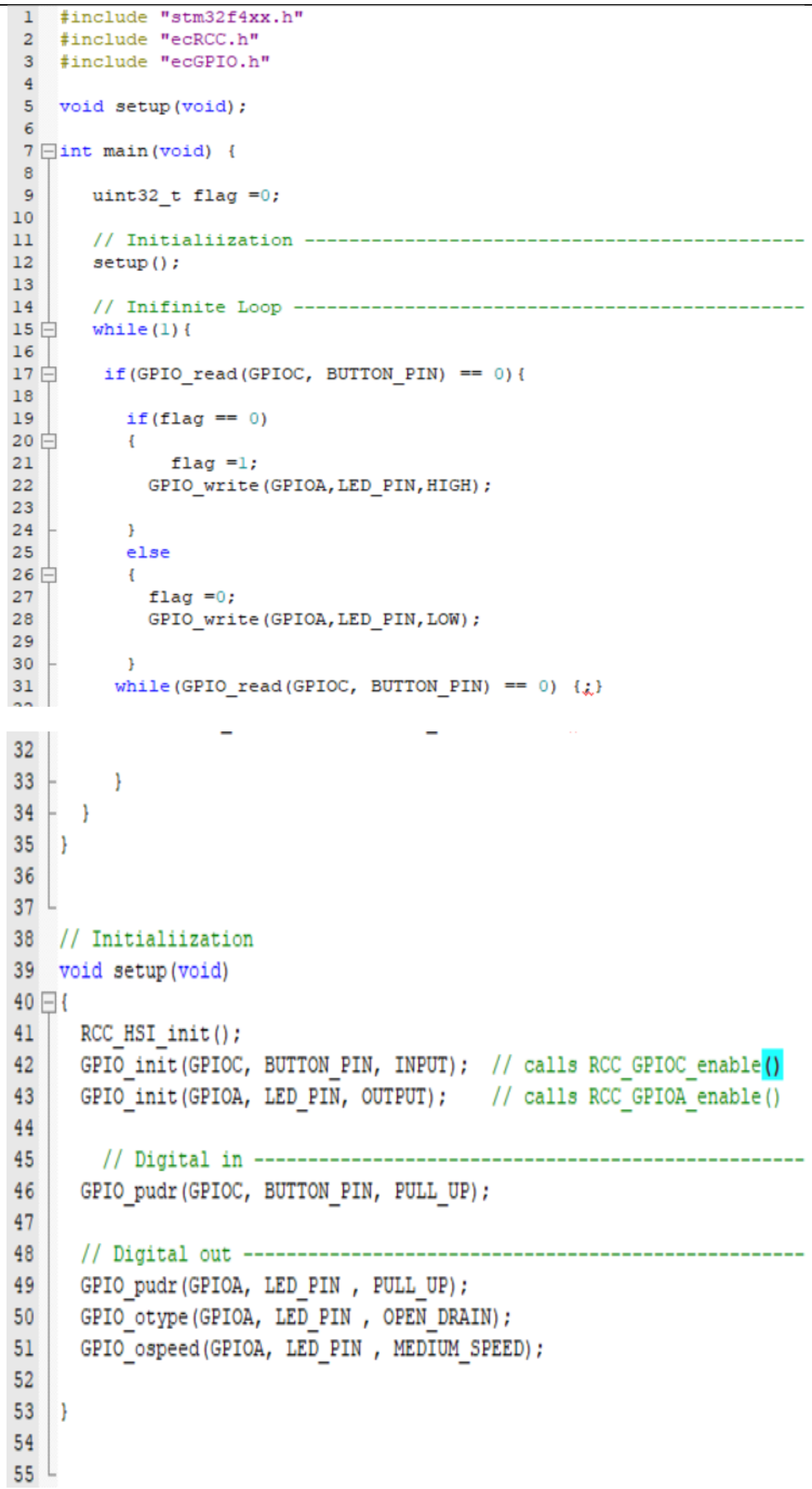


## Configuration Input and Output pins

| Digital In: Button | Digital Out: LED |
|---|---|
| GPIOC, Pin 13<br>Digital Input<br>Set PULL-UP | GPIOA, Pin 5<br>Digital Output<br>Drain<br>Pull-up<br>Medium Speed |

## Souce code

**LAB_GPIO_DigitialInOut_LED.c**



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

void setup(void);

int main(void) {

   uint32_t flag =0;

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

   // Inifinite Loop ----------------------------------------------
   while(1){

    if(GPIO_read(GPIOC, BUTTON_PIN) == 0){

      if(flag == 0)
      {
          flag =1;
        GPIO_write(GPIOA,LED_PIN,HIGH);

      }
      else
      {
        flag =0;
        GPIO_write(GPIOA,LED_PIN,LOW);

      }
     while(GPIO_read(GPIOC, BUTTON_PIN) == 0) {;}

     }
  }
}


// Initialiization
void setup(void)
{
  RCC_HSI_init();
  GPIO_init(GPIOC, BUTTON_PIN, INPUT);  // calls RCC_GPIOC_enable()
  GPIO_init(GPIOA, LED_PIN, OUTPUT);    // calls RCC_GPIOA_enable()

    // Digital in ---------------------------------------------------
  GPIO_pudr(GPIOC, BUTTON_PIN, PULL_UP);

  // Digital out ----------------------------------------------------
  GPIO_pudr(GPIOA, LED_PIN , PULL_UP);
  GPIO_otype(GPIOA, LED_PIN , OPEN_DRAIN);
  GPIO_ospeed(GPIOA, LED_PIN , MEDIUM_SPEED);

}

```

## Discussion

1) **What the differences between open-drain and Push-pull for output pin?**

Push-Pull is directly connected to Vcc and GRD through switching using two transistors and operates on its own. However, since Open-Drain uses one transistor, it cannot operate on its own and requires additional circuits. At this time, the required circuit is Pull-up or Pull-down.

Since Push-Pull uses a voltage of Vcc as it is output, it can be confirmed the LED is bright, and Open-Drain has no "Vcc" and voltage drop occurs due to resistance of Pull-up or Pull-down, and thus it can be confirmed the LED is dark

2) **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)?**

A typical solution to solve Bouncing with software is to solve Bouncing by giving time delay. For example, if the time to Bouncing when the switch is pressed is 10 ms. the time delay is given to 15 ms. there is a time delay of 5 ms after the Bouncing is over, so the Bouncing can be solved. The Bouncing with hardware is usually solved using resistance and capacitors. As a representative example, there is a method of solving Bouncing using schmitt.

The NUCLEO board used a method of solving Bouncing by giving time delay to software. I implemented time delay through "If-Statement" and "While-Statement".

# Part 3. Multiple LEDs On/Off in Sequence

Connect 4 LEDs externally. You must connect load resistor in series to LEDs as seen in the example diagram.

As Button B1 is Pressed, light one LED at a time, in sequence.

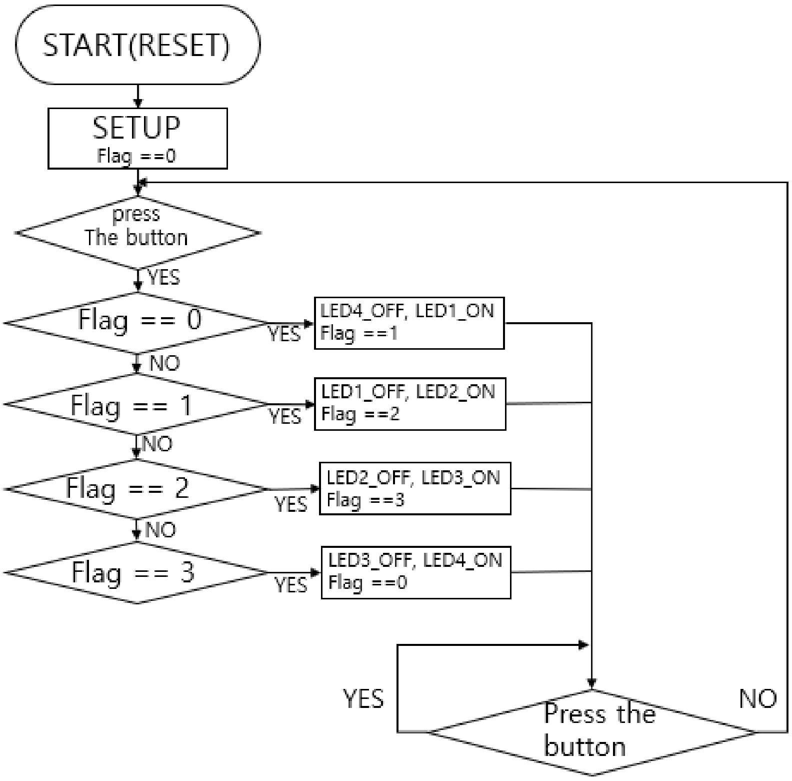
Example:  LED0--> LED1--> …LED3-->  …LED0….

## Observation of the output

If pressing and releasing the button after compilation, the LED connected to PA5 turns on, and when I press and release the button again, the LED connected to PA5 turns off and the LED connected to PA6 turns on. When pressed and released again, the LED connected to

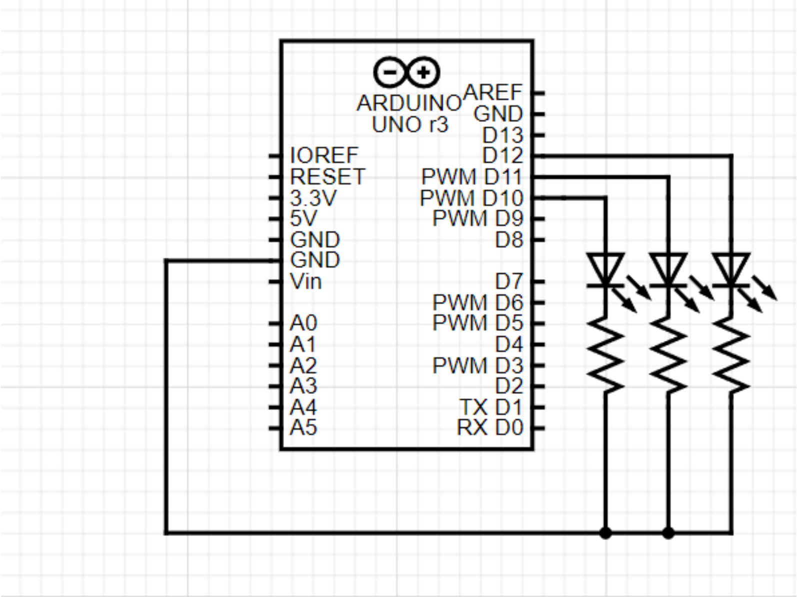
PA6 is turned off and the LED connected to PA7 is turned on. When pressed and released again, the LED connected to PA7 is turned off and the LED connected to PB6 is turned on. When pressed and released again, the LED connected to PB6 is turned off and the LED connected to PA5 is turned on. and if I press the button, repeat the above operation.

## Flow Chart



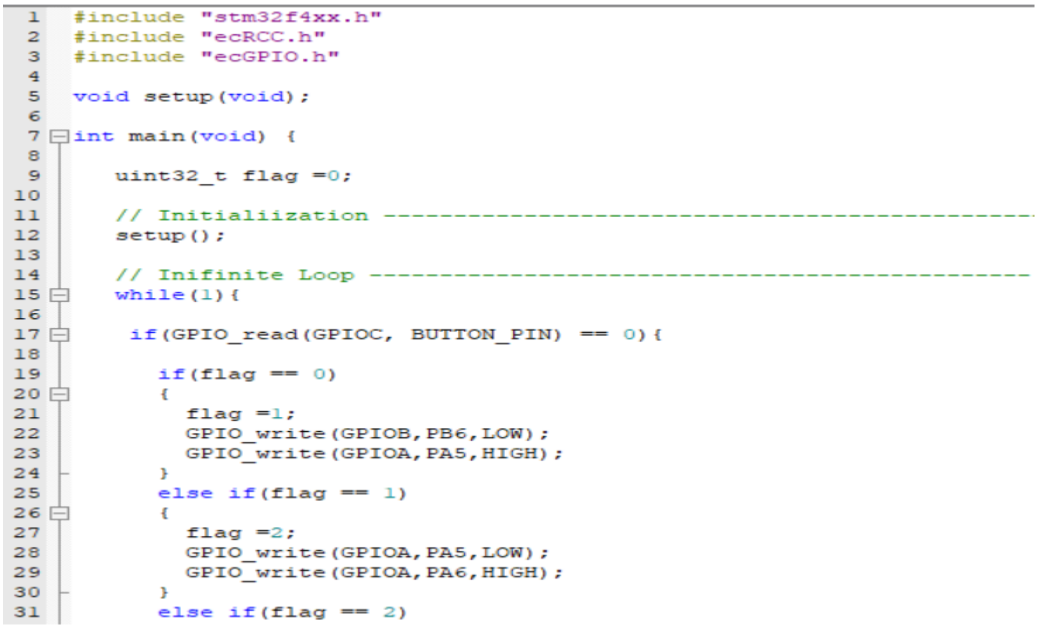
| Digital In: Button | Digital Out: LEDs |
|---|---|
| GPIOC, Pin 13<br>Digital Input<br>Set PULL-UP | PA 5, PA6, PA7,  PB6<br>Digital Output<br>Push-Pull<br>Pull-up<br>Medium Speed |

## Circuit Diagram

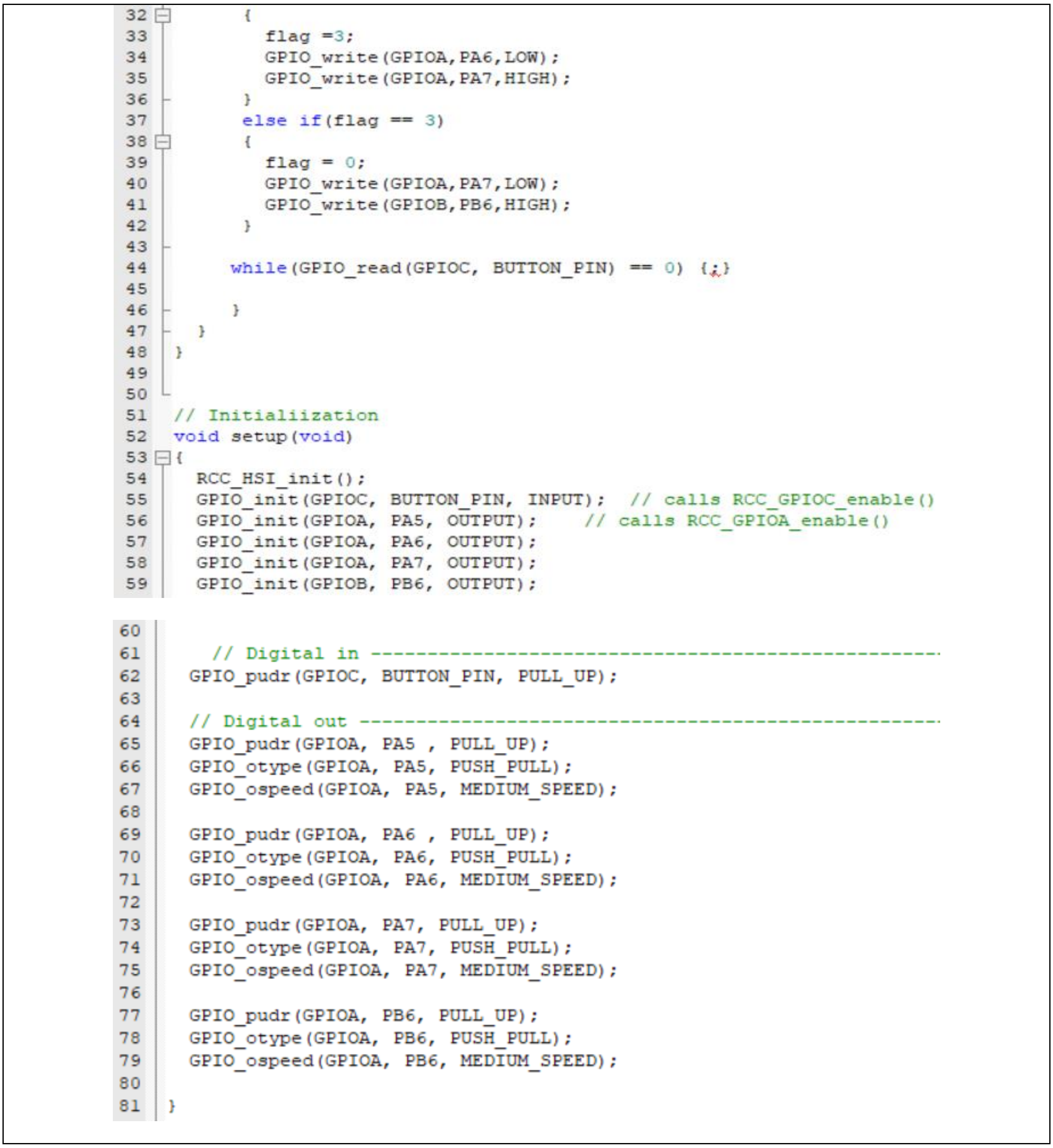


## Souce code

**LAB_GPIO_DigitialInOut_multipleLED.c**



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

void setup(void);

int main(void) {

   uint32_t flag =0;

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

   // Inifinite Loop ----------------------------------------------
   while(1){

    if(GPIO_read(GPIOC, BUTTON_PIN) == 0){

      if(flag == 0)
      {
        flag =1;
        GPIO_write(GPIOB,PB6,LOW);
        GPIO_write(GPIOA,PA5,HIGH);
      }
      else if(flag == 1)
      {
        flag =2;
        GPIO_write(GPIOA,PA5,LOW);
        GPIO_write(GPIOA,PA6,HIGH);
      }
      else if(flag == 2)
```



```
      {
        flag =3;
        GPIO_write(GPIOA,PA6,LOW);
        GPIO_write(GPIOA,PA7,HIGH);
      }
      else if(flag == 3)
      {
        flag = 0;
        GPIO_write(GPIOA,PA7,LOW);
        GPIO_write(GPIOB,PB6,HIGH);
      }

     while(GPIO_read(GPIOC, BUTTON_PIN) == 0) {;}

     }
  }
}


// Initialiization
void setup(void)
{
  RCC_HSI_init();
  GPIO_init(GPIOC, BUTTON_PIN, INPUT);  // calls RCC_GPIOC_enable()
  GPIO_init(GPIOA, PA5, OUTPUT);    // calls RCC_GPIOA_enable()
  GPIO_init(GPIOA, PA6, OUTPUT);
  GPIO_init(GPIOA, PA7, OUTPUT);
  GPIO_init(GPIOB, PB6, OUTPUT);

    // Digital in ---------------------------------------------------
  GPIO_pudr(GPIOC, BUTTON_PIN, PULL_UP);

  // Digital out ---------------------------------------------------
  GPIO_pudr(GPIOA, PA5 , PULL_UP);
  GPIO_otype(GPIOA, PA5, PUSH_PULL);
  GPIO_ospeed(GPIOA, PA5, MEDIUM_SPEED);

  GPIO_pudr(GPIOA, PA6 , PULL_UP);
  GPIO_otype(GPIOA, PA6, PUSH_PULL);
  GPIO_ospeed(GPIOA, PA6, MEDIUM_SPEED);

  GPIO_pudr(GPIOA, PA7, PULL_UP);
  GPIO_otype(GPIOA, PA7, PUSH_PULL);
  GPIO_ospeed(GPIOA, PA7, MEDIUM_SPEED);

  GPIO_pudr(GPIOA, PB6, PULL_UP);
  GPIO_otype(GPIOA, PB6, PUSH_PULL);
  GPIO_ospeed(GPIOA, PB6, MEDIUM_SPEED);

}
```

# III. Conclusion & Trouble Shooting

## Conclusion

This experiment confirmed that the output was changed by changing the values of registers in charge of mode, speed, type, etc. of GPIO. Through this, I learned how each register works. It was difficult to solve bouncing because I don't use function of time in the experiment. The problem was solved by dividing the state of button pressed and released.

## TroubleShooting

Write technical problems during the lab and solution for it. (you have any)

Q. After binary is exported to MUC LED does not blink even the button is toggled.

A. Press Reset Button

Q. It is difficult to solve the problem without using "time delay". Because I don’t learn yet in c/c++

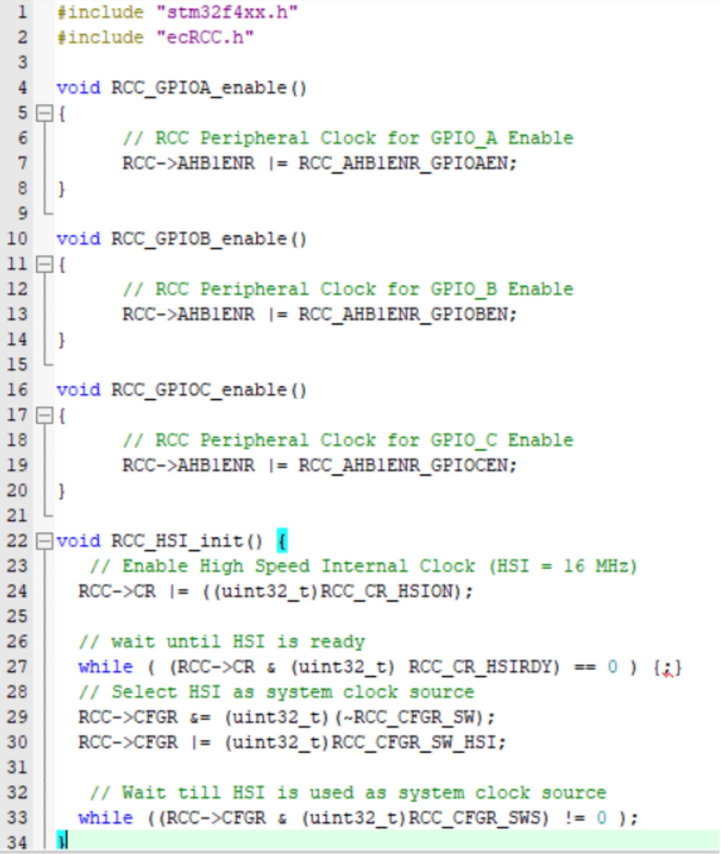
A. Use ‘While-Statemnet’ and ‘If-Statement’ logically.

Q.Check the code’s compile  without LED

A. Use the software compiler

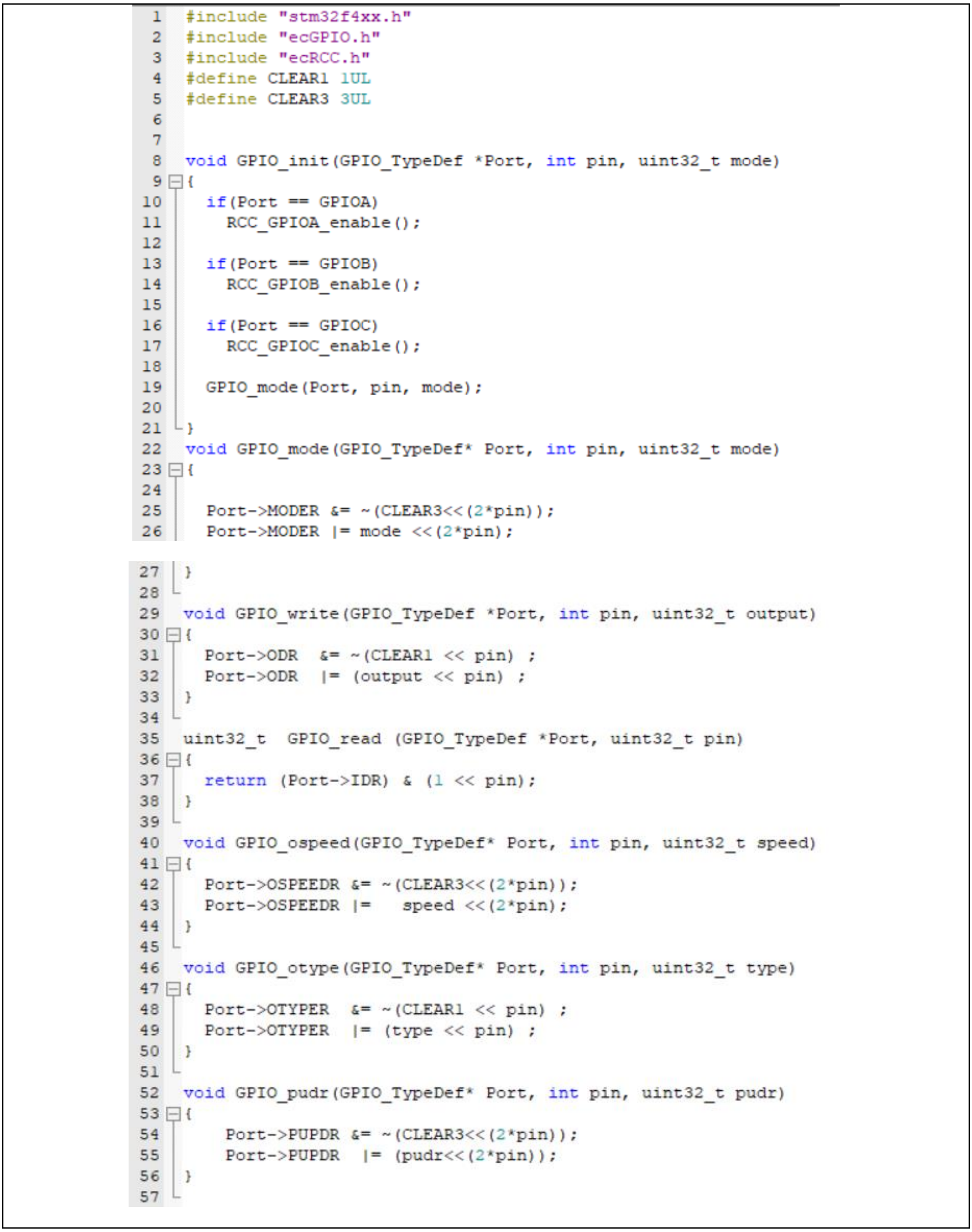
# Appendix

### Source file:  ecRCC.c



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

void RCC_GPIOA_enable()
{
      // RCC Peripheral Clock for GPIO_A Enable
      RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;
}

void RCC_GPIOB_enable()
{
      // RCC Peripheral Clock for GPIO_B Enable
      RCC->AHB1ENR |= RCC_AHB1ENR_GPIOBEN;
}

void RCC_GPIOC_enable()
{
      // RCC Peripheral Clock for GPIO_C Enable
      RCC->AHB1ENR |= RCC_AHB1ENR_GPIOCEN;
}

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

  // wait until HSI is ready
  while ( (RCC->CR & (uint32_t) RCC_CR_HSIRDY) == 0 ) {;}
  // Select HSI as system clock source
  RCC->CFGR &= (uint32_t)(~RCC_CFGR_SW);
  RCC->CFGR |= (uint32_t)RCC_CFGR_SW_HSI;

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

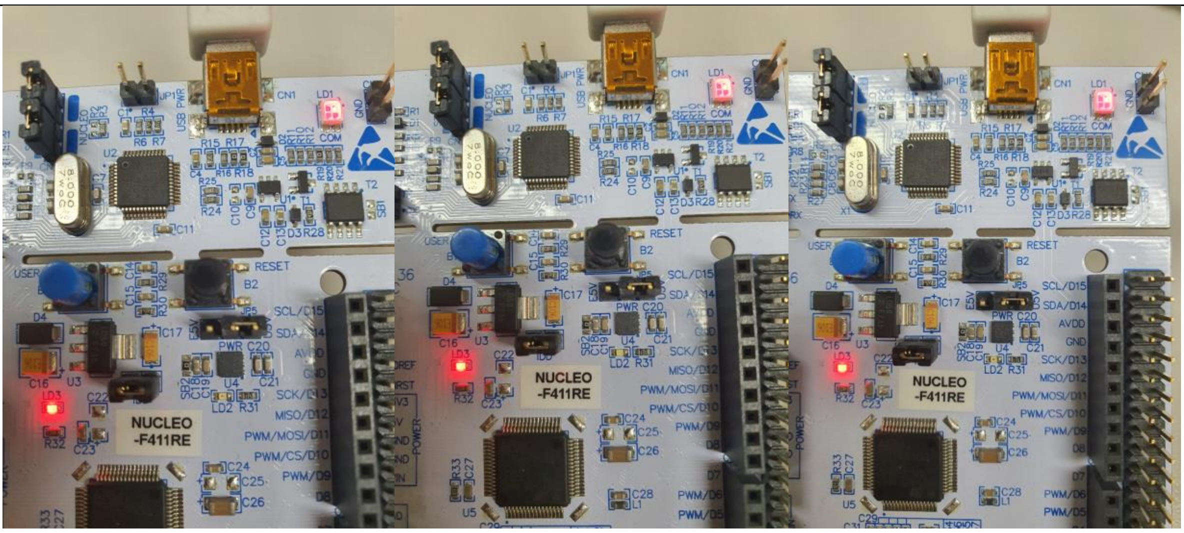
### Source file:  ecGPIO.c



```c
#include "stm32f4xx.h"
#include "ecGPIO.h"
#include "ecRCC.h"
#define CLEAR1 1UL
#define CLEAR3 3UL


void GPIO_init(GPIO_TypeDef *Port, int pin, uint32_t mode)
{
  if(Port == GPIOA)
    RCC_GPIOA_enable();

  if(Port == GPIOB)
    RCC_GPIOB_enable();

  if(Port == GPIOC)
    RCC_GPIOC_enable();

  GPIO_mode(Port, pin, mode);

}
void GPIO_mode(GPIO_TypeDef* Port, int pin, uint32_t mode)
{

  Port->MODER &= ~(CLEAR3<<(2*pin));
  Port->MODER |= mode <<(2*pin);
}

void GPIO_write(GPIO_TypeDef *Port, int pin, uint32_t output)
{
  Port->ODR  &= ~(CLEAR1 << pin) ;
  Port->ODR  |= (output << pin) ;
}

uint32_t  GPIO_read (GPIO_TypeDef *Port, uint32_t pin)
{
  return (Port->IDR) & (1 << pin);
}

void GPIO_ospeed(GPIO_TypeDef* Port, int pin, uint32_t speed)
{
  Port->OSPEEDR &= ~(CLEAR3<<(2*pin));
  Port->OSPEEDR |=   speed <<(2*pin);
}

void GPIO_otype(GPIO_TypeDef* Port, int pin, uint32_t type)
{
  Port->OTYPER  &= ~(CLEAR1 << pin) ;
  Port->OTYPER  |= (type << pin) ;
}

void GPIO_pudr(GPIO_TypeDef* Port, int pin, uint32_t pudr)
{
    Port->PUPDR &= ~(CLEAR3<<(2*pin));
    Port->PUPDR  |= (pudr<<(2*pin));
}

```

Embedded Controller

LED_operation_part2

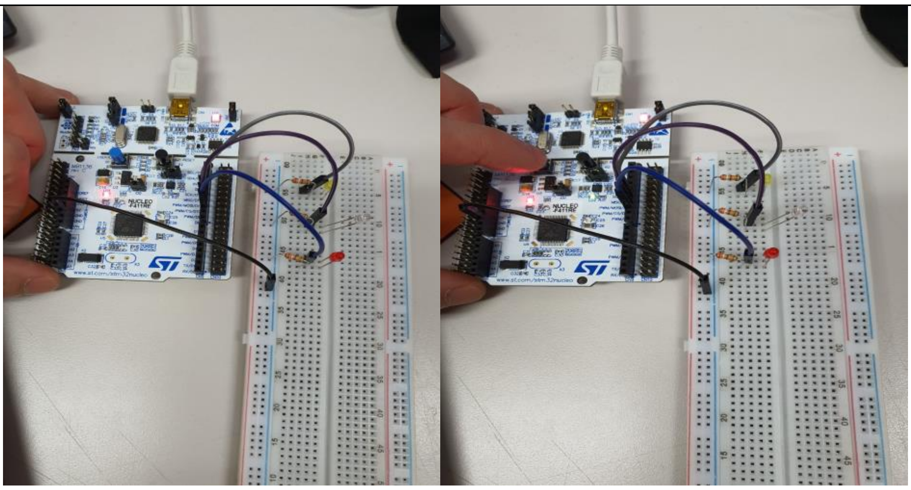


Left: default

Center: press one time

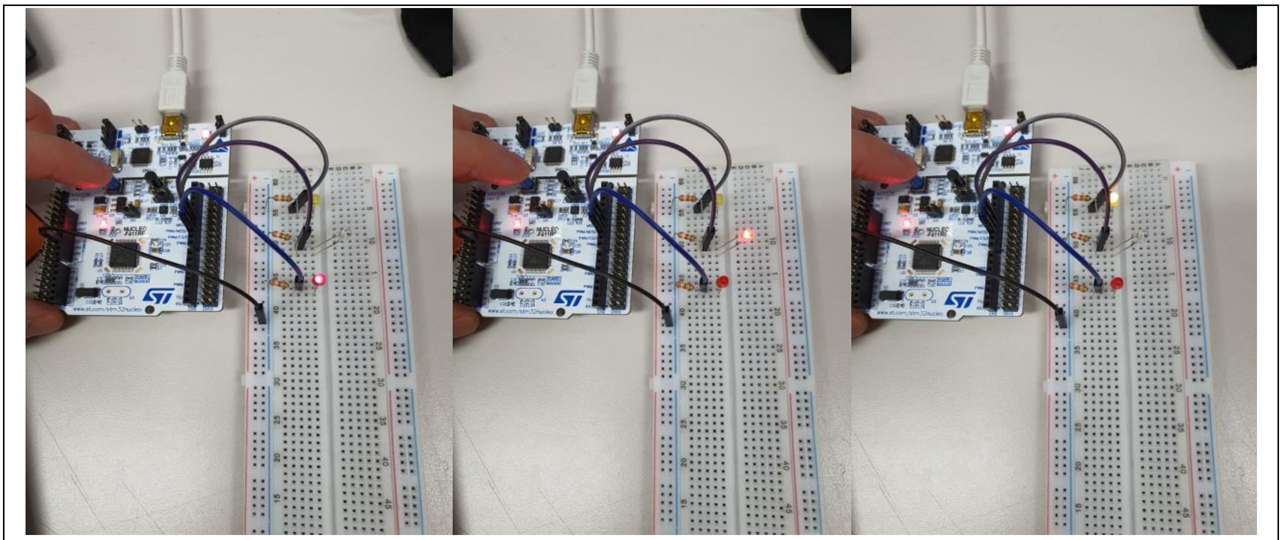
Right: press two times

LED_operation_part3



Left: default

Right: press one time



Left: press two times

Center: press three times

Right: press four times