Embedded Controller

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

Date: 2021.10.07 Thur.

Name(ID): Hong Se Hyun(21700791)

Partner Name: Yoo Jin Su

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.

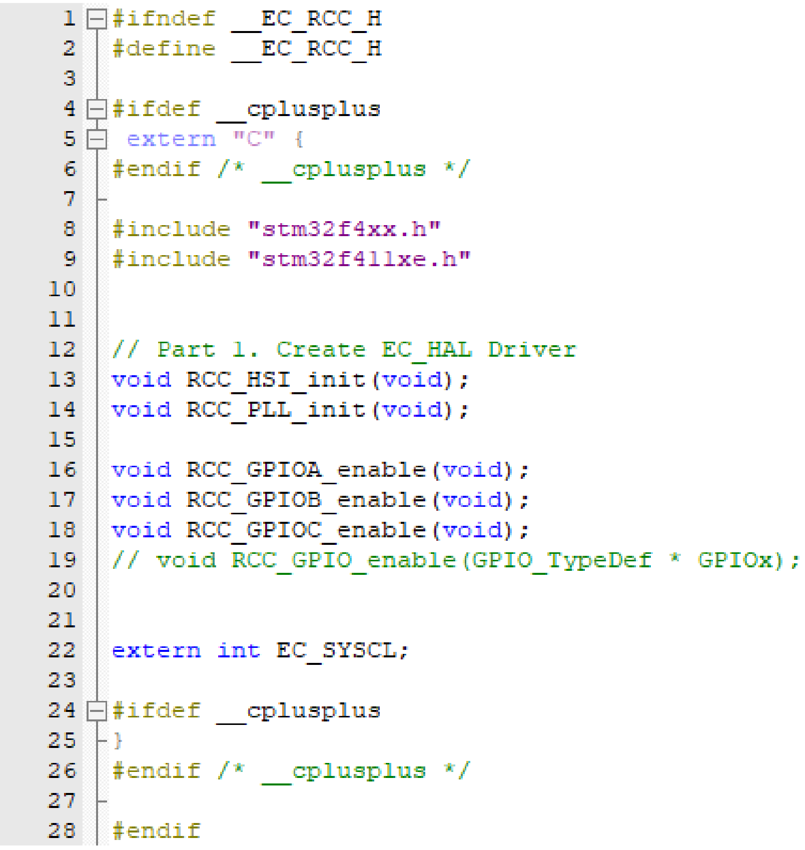
Include File	Function

ecRCC.h, c	void RCC_HSI_init(void);
	void RCC_GPIOA_enable(void); // This can go inside GPIO_init()
	void RCC_GPIOB_enable(void);
	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);

Source code

ecRCC.h

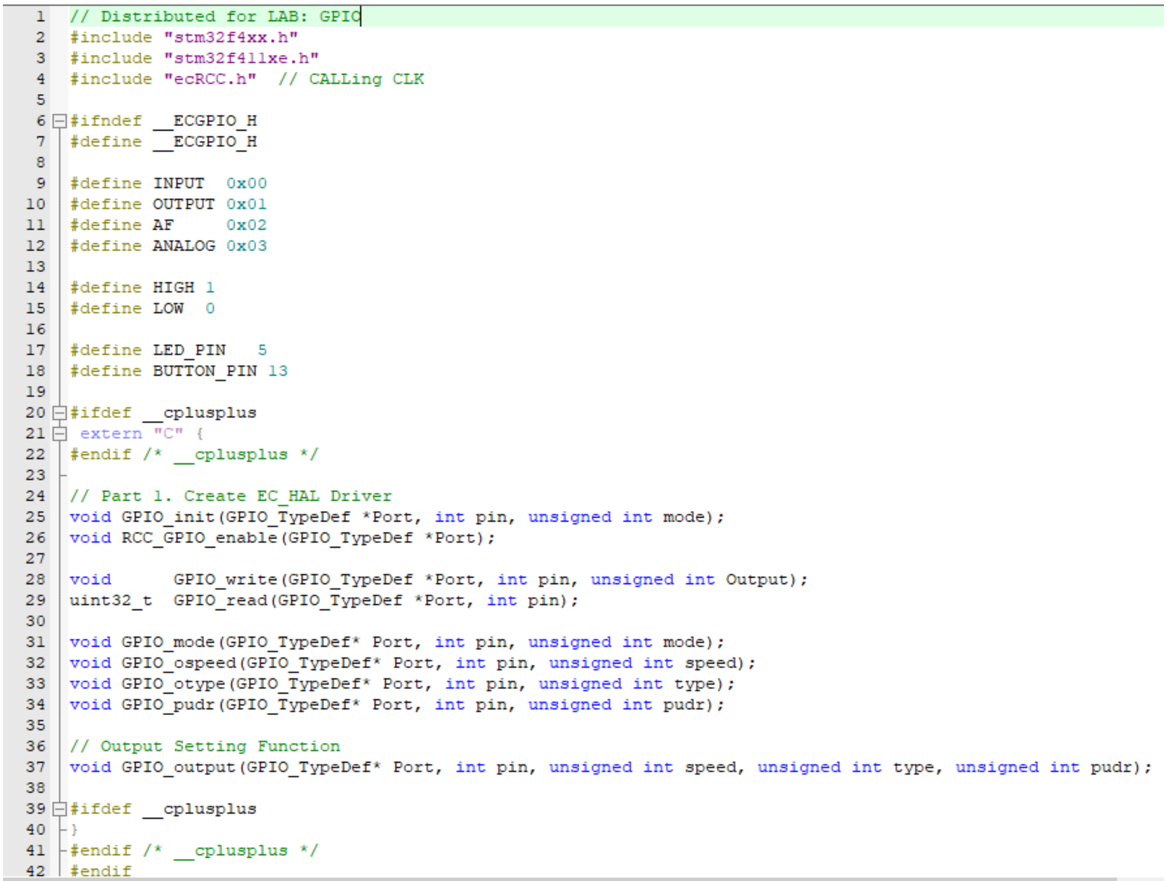


```c
#ifndef __EC_RCC_H
#define __EC_RCC_H

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

#include "stm32f4xx.h"
#include "stm32f411xe.h"


// Part 1. Create EC_HAL Driver
void RCC_HSI_init(void);
void RCC_PLL_init(void);

void RCC_GPIOA_enable(void);
void RCC_GPIOB_enable(void);
void RCC_GPIOC_enable(void);
// void RCC_GPIO_enable(GPIO_TypeDef * GPIOx);


extern int EC_SYSCL;

#ifdef __cplusplus
}
#endif /* __cplusplus */

#endif
```

Figure 1.1 Header Code of ecRCC.h

ecRCC.c : See Appendix

ecGPIO.h

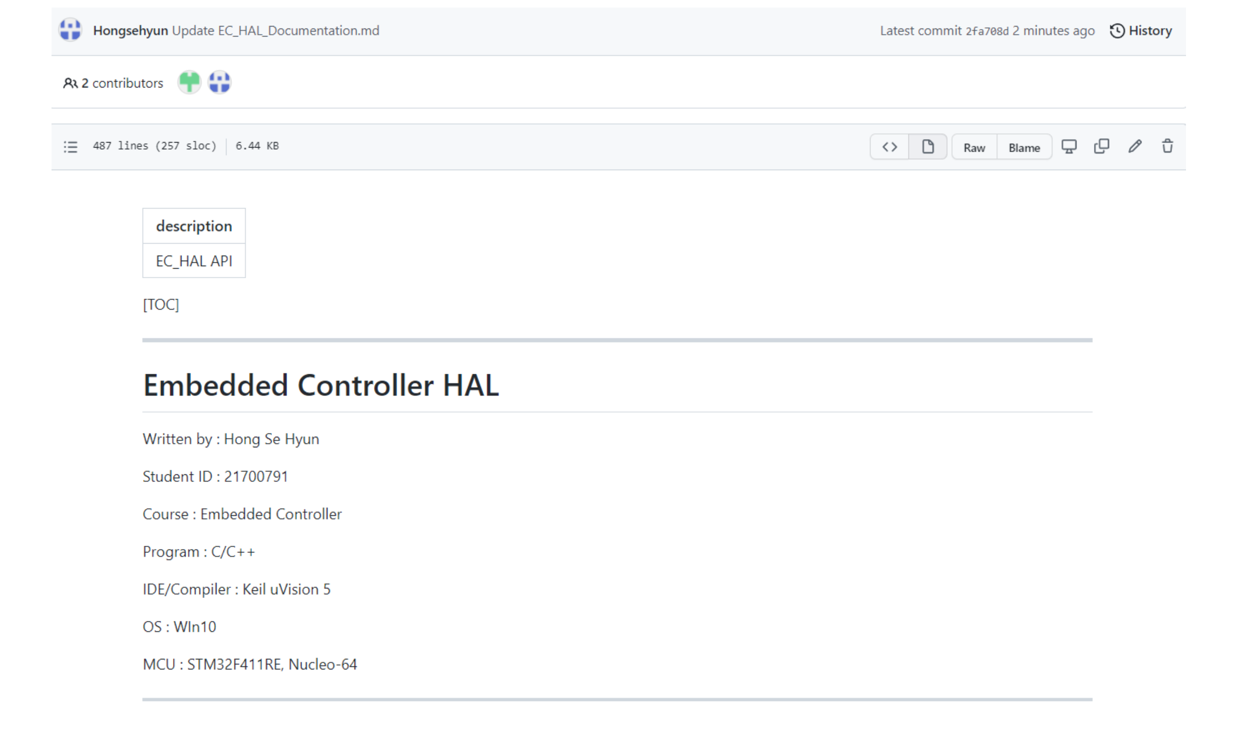


```c
// Distributed for LAB: GPIO
#include "stm32f4xx.h"
#include "stm32f411xe.h"
#include "ecRCC.h"  // CALLing CLK

#ifndef __ECGPIO_H
#define __ECGPIO_H

#define INPUT  0x00
#define OUTPUT 0x01
#define AF     0x02
#define ANALOG 0x03

#define HIGH 1
#define LOW  0

#define LED_PIN    5
#define BUTTON_PIN 13

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

// Part 1. Create EC_HAL Driver
void GPIO_init(GPIO_TypeDef *Port, int pin, unsigned int mode);
void RCC_GPIO_enable(GPIO_TypeDef *Port);

void       GPIO_write(GPIO_TypeDef *Port, int pin, unsigned int Output);
uint32_t  GPIO_read(GPIO_TypeDef *Port, int pin);

void GPIO_mode(GPIO_TypeDef* Port, int pin, unsigned int mode);
void GPIO_ospeed(GPIO_TypeDef* Port, int pin, unsigned int speed);
void GPIO_otype(GPIO_TypeDef* Port, int pin, unsigned int type);
void GPIO_pudr(GPIO_TypeDef* Port, int pin, unsigned int pudr);

// Output Setting Function
void GPIO_output(GPIO_TypeDef* Port, int pin, unsigned int speed, unsigned int type, unsigned int pudr);

#ifdef __cplusplus
}
#endif /* __cplusplus */
#endif
```

Figure 1.2 Header Code of ecGPIO.h

ecGPIO.c :   See Appendix

Documenation of Library



Hongsehyun Update EC_HAL_Documentation.md
Latest commit 2fa708d 2 minutes ago
History
2 contributors
487 lines (257 sloc) | 6.44 KB
Raw
Blame

description
EC_HAL API

[TOC]

Embedded Controller HAL

Written by : Hong Se Hyun

Student ID : 21700791

Course : Embedded Controller

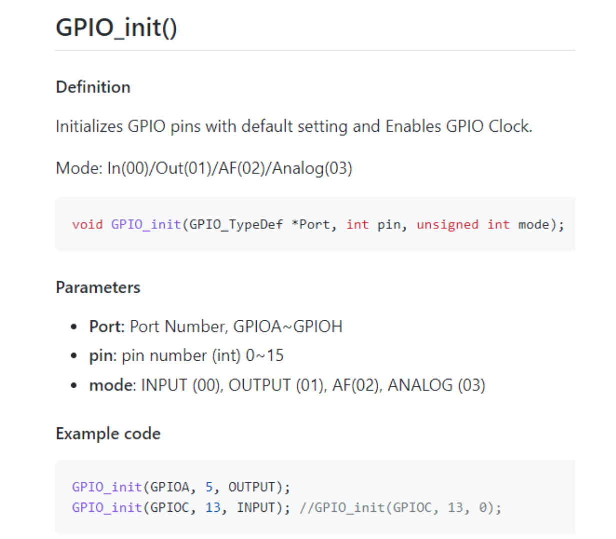
Program : C/C++

IDE/Compiler : Keil uVision 5

OS : WIn10

MCU : STM32F411RE, Nucleo-64

Figure 2.1 Documentation in GitHub



GPIO_init()

Definition

Initializes GPIO pins with default setting and Enables GPIO Clock.

Mode: In(00)/Out(01)/AF(02)/Analog(03)

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

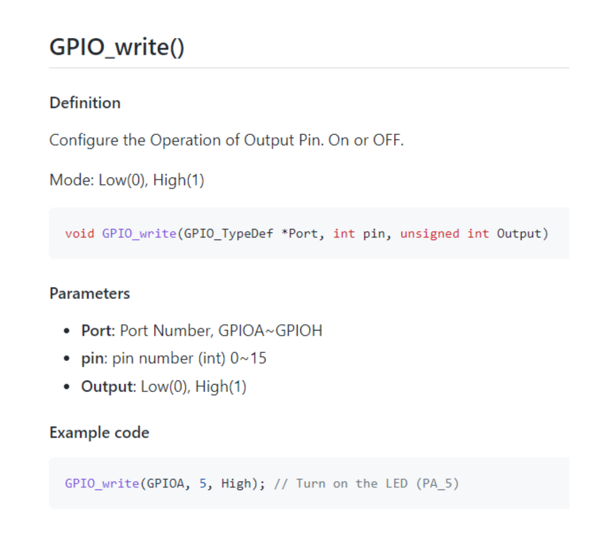
Parameters

- **Port**: Port Number, GPIOA~GPIOH
- **pin**: pin number (int) 0~15
- **mode**: INPUT (00), OUTPUT (01), AF(02), ANALOG (03)

Example code

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

Figure 2.2 Documentation of GPIO_init



GPIO_write()

Definition

Configure the Operation of Output Pin. On or OFF.

Mode: Low(0), High(1)

```
void GPIO_write(GPIO_TypeDef *Port, int pin, unsigned int Output)
```

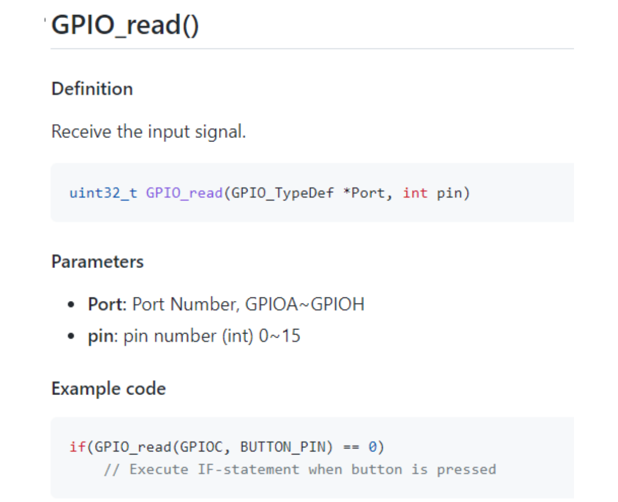
Parameters

- **Port**: Port Number, GPIOA~GPIOH
- **pin**: pin number (int) 0~15
- **Output**: Low(0), High(1)

Example code

```
GPIO_write(GPIOA, 5, High); // Turn on the LED (PA_5)
```

Figure 2.3 Documentation of GPIO_write



GPIO_read()

Definition

Receive the input signal.

```
uint32_t GPIO_read(GPIO_TypeDef *Port, int pin)
```

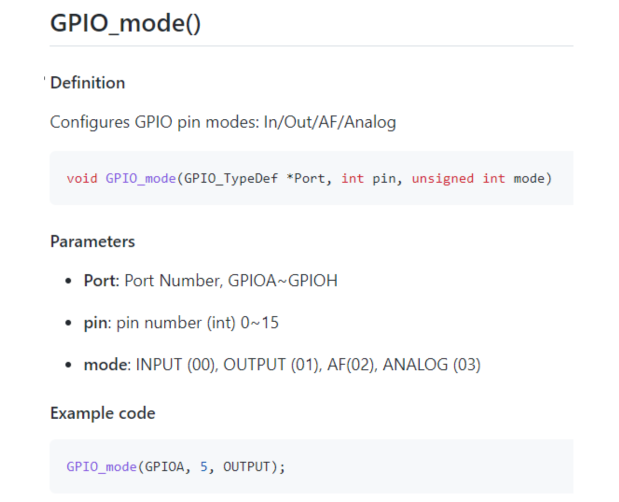
Parameters

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

Example code

```
if(GPIO_read(GPIOC, BUTTON_PIN) == 0)
    // Execute IF-statement when button is pressed
```

Figure 2.4 Documentation of GPIO_read



GPIO_mode()

Definition

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

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

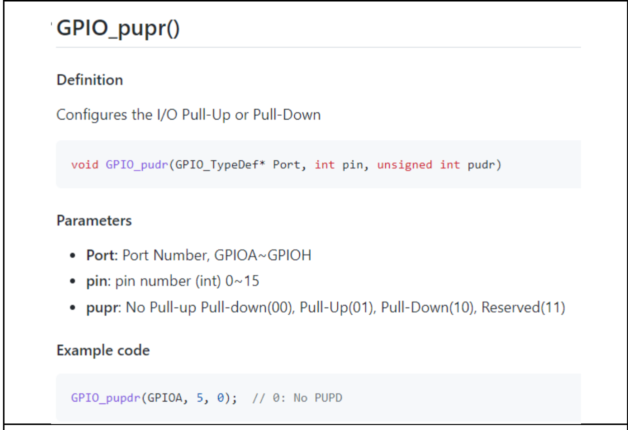
Parameters

- **Port**: Port Number, GPIOA~GPIOH
- **pin**: pin number (int) 0~15
- **mode**: INPUT (00), OUTPUT (01), AF(02), ANALOG (03)

Example code

```
GPIO_mode(GPIOA, 5, OUTPUT);
```

Figure 2.5 Documentation of GPIO_mode



GPIO_pupr()

Definition

Configures the I/O Pull-Up or Pull-Down

```
void GPIO_pudr(GPIO_TypeDef* Port, int pin, unsigned int pudr)
```

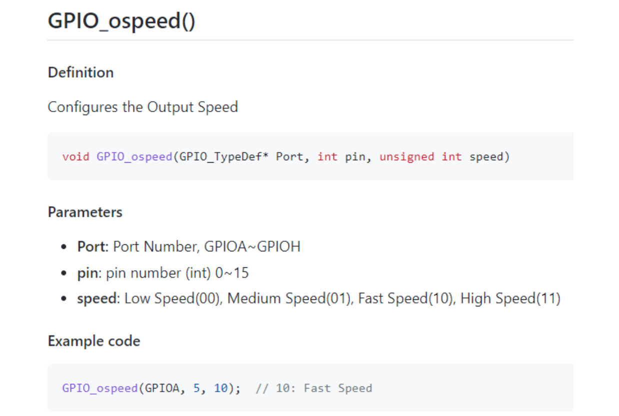
Parameters

- **Port**: Port Number, GPIOA~GPIOH
- **pin**: pin number (int) 0~15
- **pupr**: No Pull-up Pull-down(00), Pull-Up(01), Pull-Down(10), Reserved(11)

Example code

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

Figure 2.6 Documentation of GPIO_pupr



GPIO_ospeed()

Definition

Configures the Output Speed

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

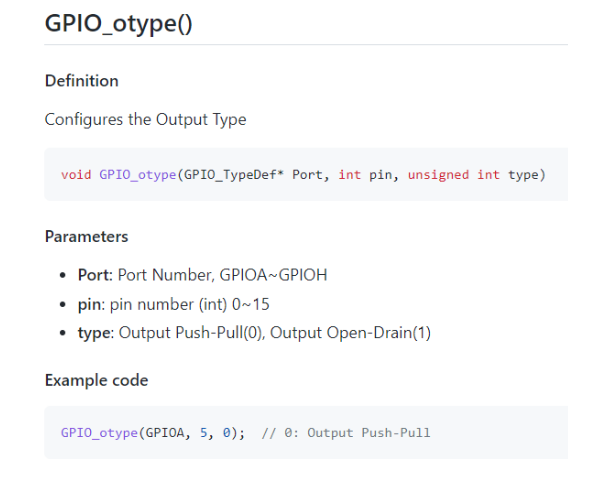
Parameters

- **Port**: Port Number, GPIOA~GPIOH
- **pin**: pin number (int) 0~15
- **speed**: Low Speed(00), Medium Speed(01), Fast Speed(10), High Speed(11)

Example code

```
GPIO_ospeed(GPIOA, 5, 10);  // 10: Fast Speed
```

Figure 2.7 Documentation of GPIO_ospeed



GPIO_otype()

Definition

Configures the Output Type

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

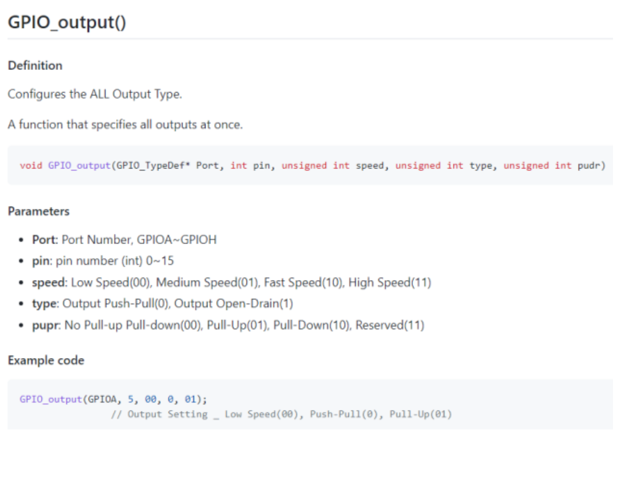
Parameters

- **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: Output Push-Pull
```

Figure 2.8 Documentation of GPIO_otype



GPIO_output()

Definition

Configures the ALL Output Type.

A function that specifies all outputs at once.

```
void GPIO_output(GPIO_TypeDef* Port, int pin, unsigned int speed, unsigned int type, unsigned int pudr)
```

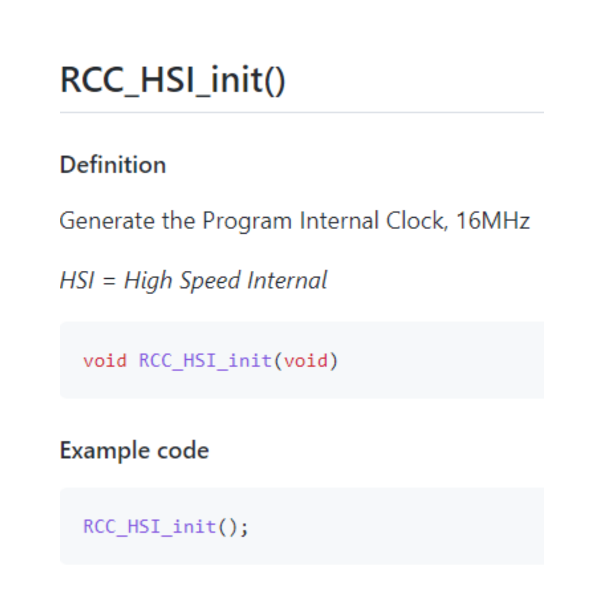
Parameters

- **Port**: Port Number, GPIOA~GPIOH
- **pin**: pin number (int) 0~15
- **speed**: Low Speed(00), Medium Speed(01), Fast Speed(10), High Speed(11)
- **type**: Output Push-Pull(0), Output Open-Drain(1)
- **pupr**: No Pull-up Pull-down(00), Pull-Up(01), Pull-Down(10), Reserved(11)

Example code

```
GPIO_output(GPIOA, 5, 00, 0, 01);
            // Output Setting _ Low Speed(00), Push-Pull(0), Pull-Up(01)
```

Figure 2.9 Documentation of GPIO_output



RCC_HSI_init()

Definition

Generate the Program Internal Clock, 16MHz

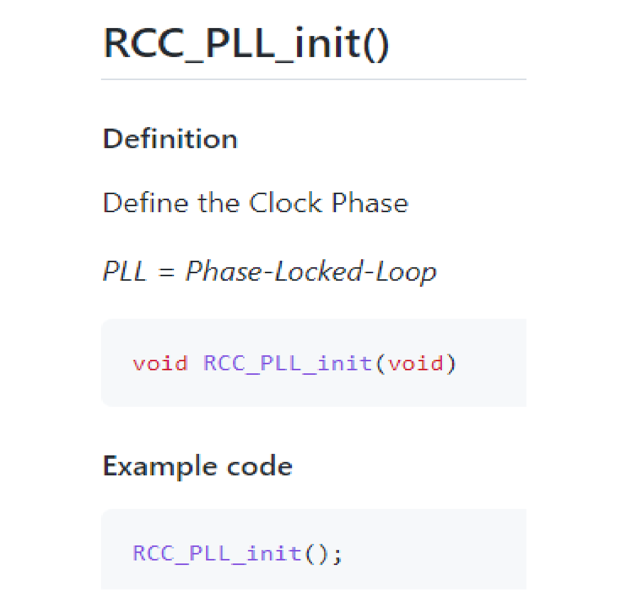
HSI = High Speed Internal

```
void RCC_HSI_init(void)
```

Example code

```
RCC_HSI_init();
```

Figure 2.10 Documentation of RCC_HSI_init



RCC_PLL_init()

Definition

Define the Clock Phase

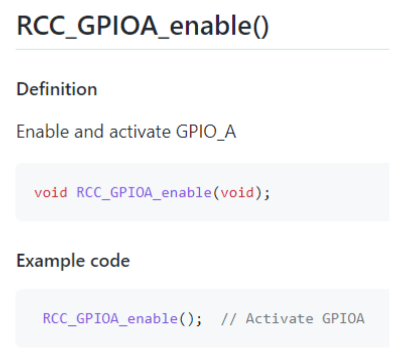
PLL = Phase-Locked-Loop

```
void RCC_PLL_init(void)
```

Example code

```
RCC_PLL_init();
```

Figure 2.11 Documentation of RCC_PLL_init



RCC_GPIOA_enable()

Definition

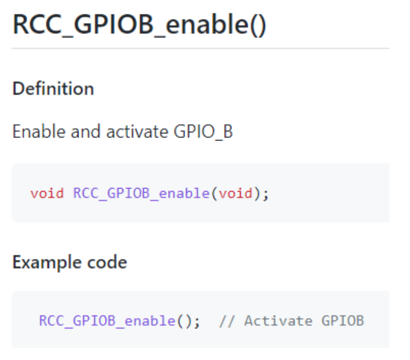
Enable and activate GPIO_A

```
void RCC_GPIOA_enable(void);
```

Example code

```
RCC_GPIOA_enable();  // Activate GPIOA
```

Figure 2.12 GPIOA_enable



RCC_GPIOB_enable()

Definition

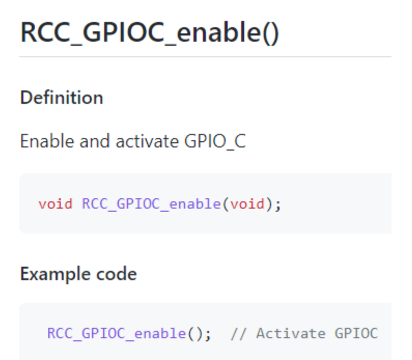
Enable and activate GPIO_B

```
void RCC_GPIOB_enable(void);
```

Example code

```
RCC_GPIOB_enable();  // Activate GPIOB
```

Figure GPIOB_enable



RCC_GPIOC_enable()

Definition

Enable and activate GPIO_C

```
void RCC_GPIOC_enable(void);
```

Example code

```
RCC_GPIOC_enable();  // Activate GPIOC
```

Figure 2.14 GPIOC_enable

Part 2. Toggle LED with push – button input

Create a new project named as “LAB_GPIO_DigitialInOut_LED”.

Name the source file as “LAB_GPIO_DigitialInOut_LED.c”,

Output Observation

- As soon as I pressed the button and removed my hand from the button, I could observe that ‘LD2’ was turned on and off. In other words, If the button was pressed and removed with ‘LD2’ on, the LED turned off, and if the button was pressed and removed with ‘LD2’ turned off, the LED turned on.

  In particular, because the Output Type was set to OpenDrain, the Output was floated and the light was dimly lit in LD2.

Configuration Input and Output pins

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

Flow Chart

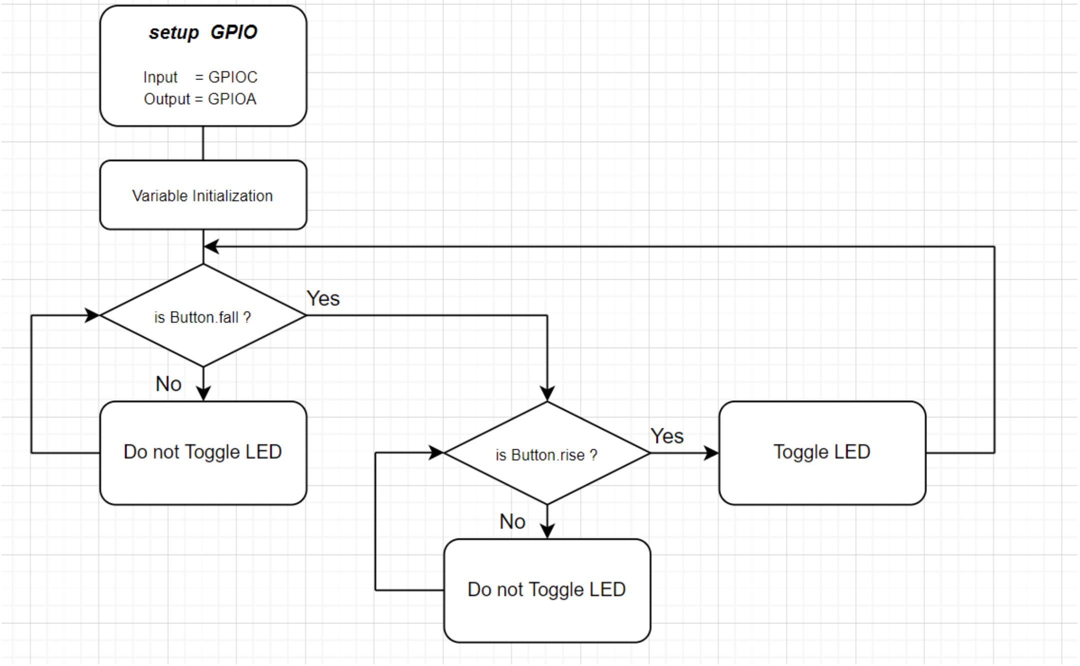
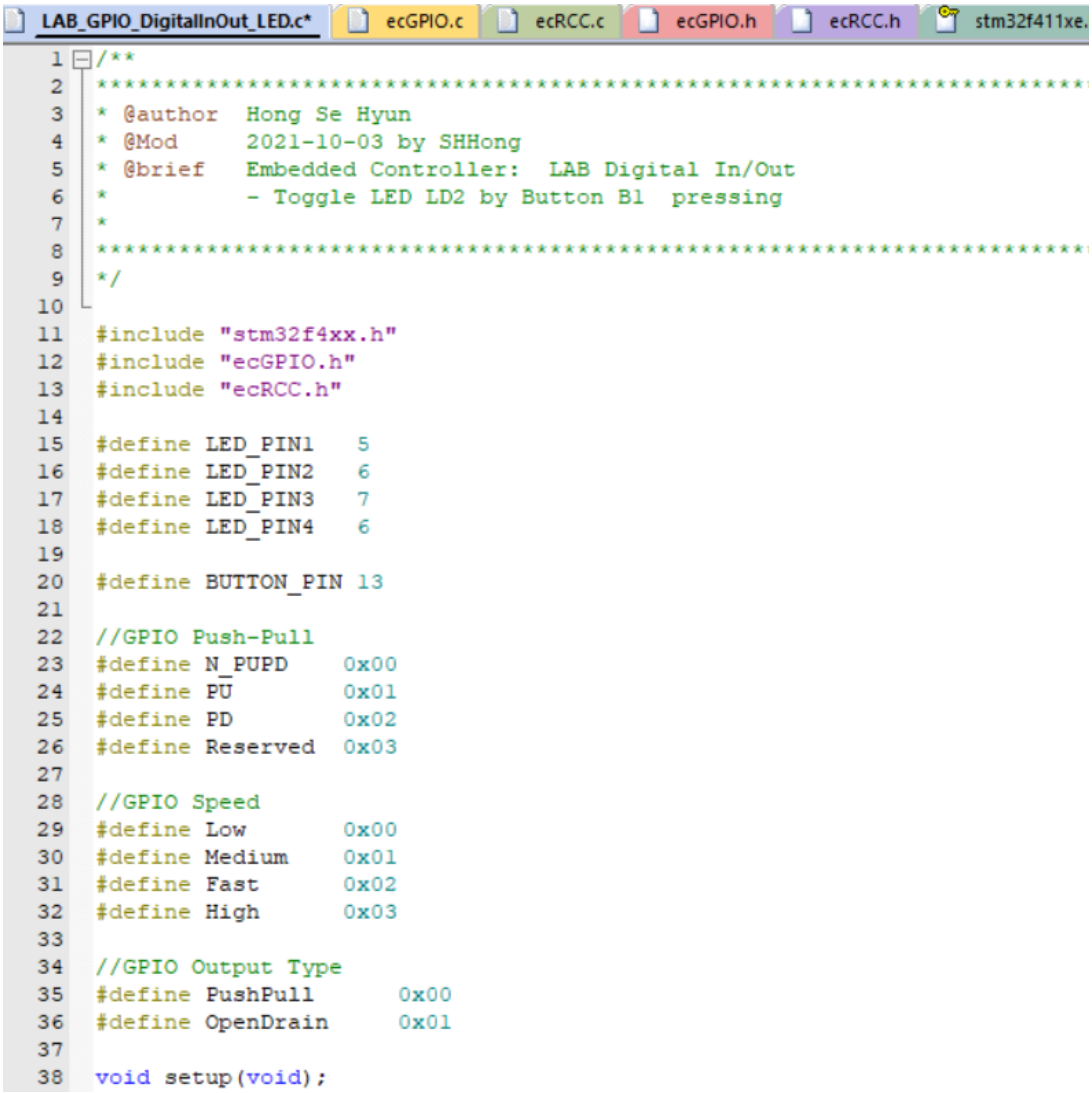


Figure 3.1 Flow-Chart of Part(2)

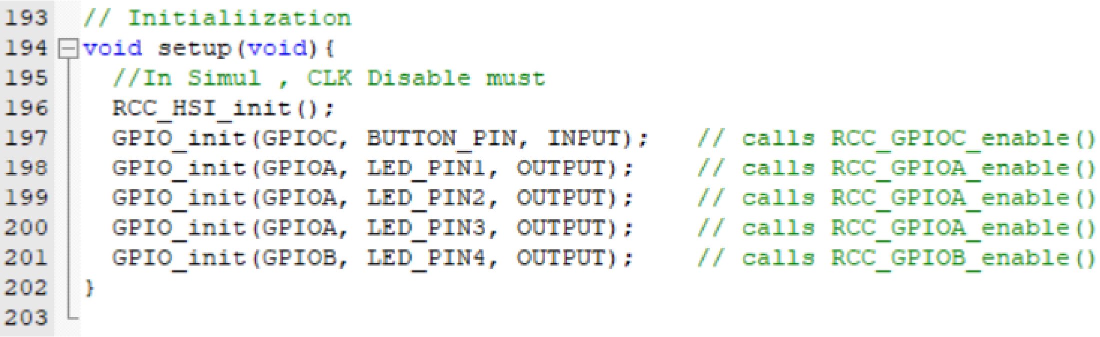
Source code

LAB_GPIO_DigitialInOut_LED.c



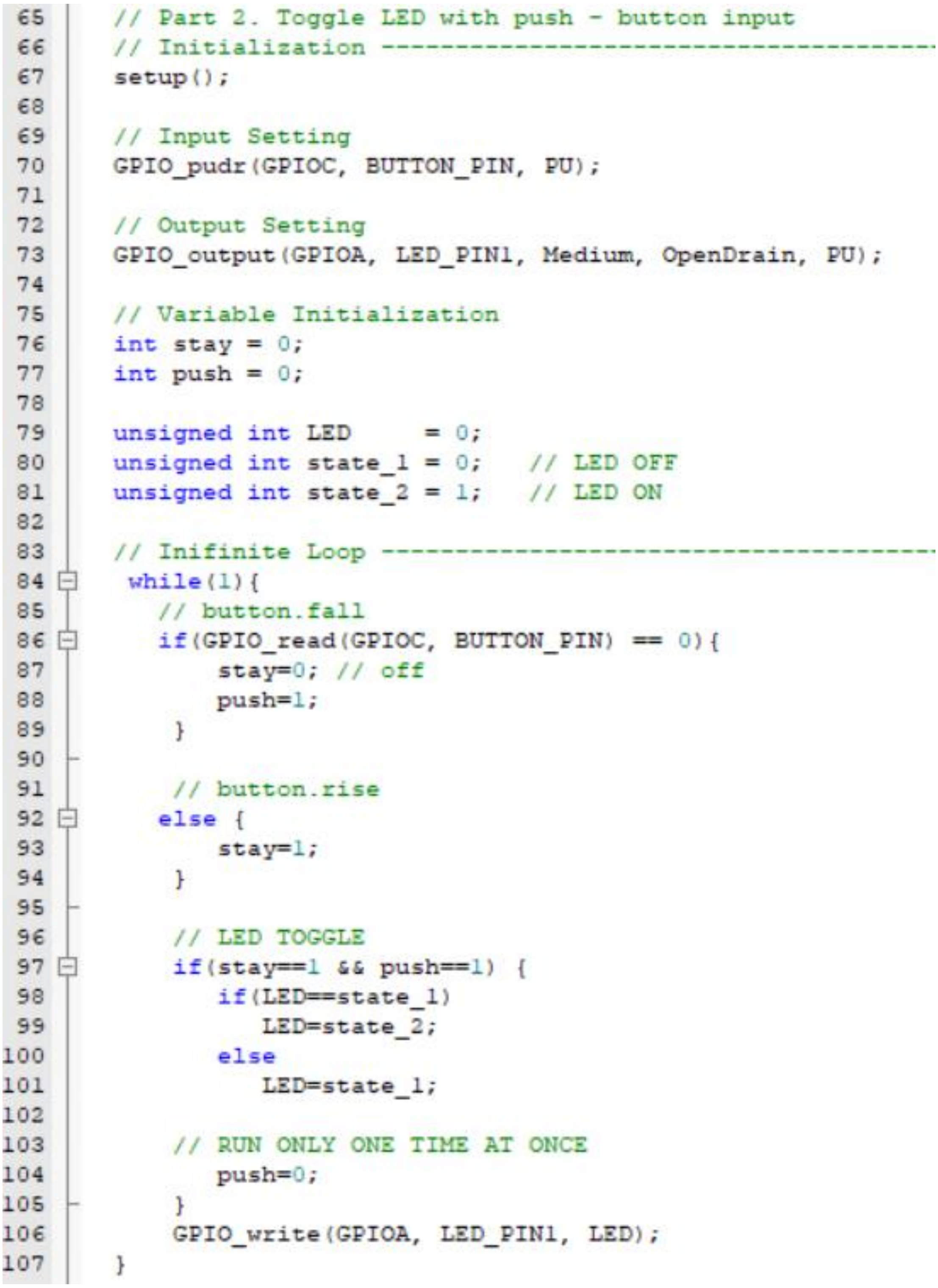
```c
/**
**************************************************************************
* @author  Hong Se Hyun
* @Mod     2021-10-03 by SHHong
* @brief   Embedded Controller:  LAB Digital In/Out
*           - Toggle LED LD2 by Button B1  pressing
*
**************************************************************************
*/

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

#define LED_PIN1    5
#define LED_PIN2    6
#define LED_PIN3    7
#define LED_PIN4    6

#define BUTTON_PIN 13

//GPIO Push-Pull
#define N_PUPD    0x00
#define PU        0x01
#define PD        0x02
#define Reserved  0x03

//GPIO Speed
#define Low       0x00
#define Medium    0x01
#define Fast      0x02
#define High      0x03

//GPIO Output Type
#define PushPull      0x00
#define OpenDrain     0x01

void setup(void);
```

Figure 3.2 Define Code of Main.c



```c
// Initialiization
void setup(void){
  //In Simul , CLK Disable must
  RCC_HSI_init();
  GPIO_init(GPIOC, BUTTON_PIN, INPUT);    // calls RCC_GPIOC_enable()
  GPIO_init(GPIOA, LED_PIN1, OUTPUT);     // calls RCC_GPIOA_enable()
  GPIO_init(GPIOA, LED_PIN2, OUTPUT);     // calls RCC_GPIOA_enable()
  GPIO_init(GPIOA, LED_PIN3, OUTPUT);     // calls RCC_GPIOA_enable()
  GPIO_init(GPIOB, LED_PIN4, OUTPUT);     // calls RCC_GPIOB_enable()
}
```

Figure 3.3 Setup Function in Main.c



```
// Part 2. Toggle LED with push - button input
// Initialization -------------------------------------
setup();

// Input Setting
GPIO_pudr(GPIOC, BUTTON_PIN, PU);

// Output Setting
GPIO_output(GPIOA, LED_PIN1, Medium, OpenDrain, PU);

// Variable Initialization
int stay = 0;
int push = 0;

unsigned int LED     = 0;
unsigned int state_1 = 0;   // LED OFF
unsigned int state_2 = 1;   // LED ON

// Inifinite Loop -------------------------------------
 while(1){
   // button.fall
   if(GPIO_read(GPIOC, BUTTON_PIN) == 0){
       stay=0; // off
       push=1;
    }

    // button.rise
   else {
       stay=1;
    }

    // LED TOGGLE
    if(stay==1 && push==1) {
       if(LED==state_1)
          LED=state_2;
       else
          LED=state_1;

    // RUN ONLY ONE TIME AT ONCE
       push=0;
    }
    GPIO_write(GPIOA, LED_PIN1, LED);
}
```

Figure 3.4 Source Code of DigitalInOut_LED

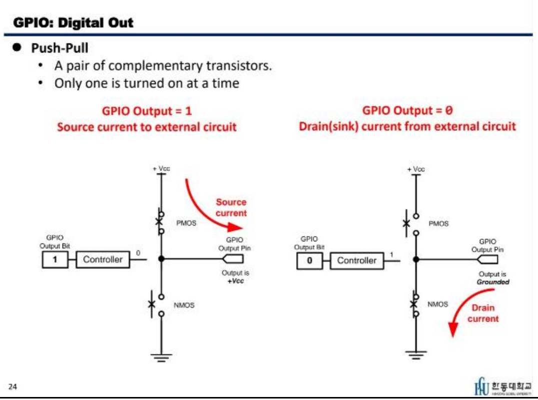
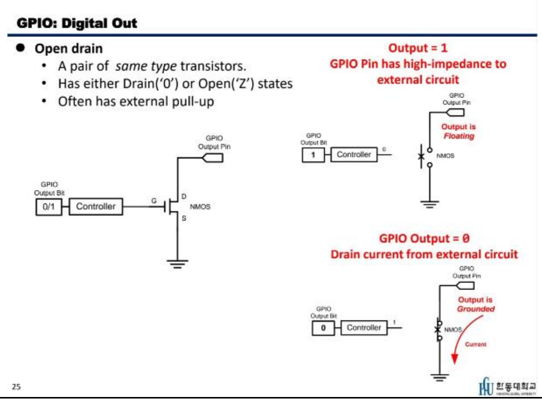
Discussion

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

The Output Port of Push-Pull consists of two transistors. In addition, Output Port of Push-Pull can operate on its own without any additional circuits. In other words, it may be implemented independently even if nothing is connected to the Output Port. This allows the applied power V_{cc} to be transmitted to the 'GPIO Output Pin' as it is.

In contrast, Output Port of Open-Drain consists of one transistor and has a structure that cannot operate independently. Therefore, an additional circuit is required outside, and it is characterized by being used with a Pull-Up Resistor. In this case, in the process of transmitting the input power V_{cc} to the 'GPIO Output Pin', the current passes through the resistor.

In summary, considering Push-Pull and Open-Drain from the perspective of Output Pin, LED outputs relatively bright light in Push-Pull, which does not pass through resistance, and relatively faint light in Open-Drain, which passes through resistance.

 GPIO: Digital Out ● Push-Pull • A pair of complementary transistors. • Only one is turned on at a time GPIO Output = 1 Source current to external circuit GPIO Output = 0 Drain(sink) current from external circuit + Vcc Source current PMOS GPIO Output Bit 1 Controller 0 GPIO Output Pin Output is +Vcc NMOS + Vcc PMOS GPIO Output Bit 0 Controller 1 GPIO Output Pin Output is Grounded NMOS Drain current 24	 GPIO: Digital Out ● Open drain • A pair of *same type* transistors. • Has either Drain('0') or Open('Z') states • Often has external pull-up Output = 1 GPIO Pin has high-impedance to external circuit GPIO Output = 0 Drain current from external circuit GPIO Output Pin GPIO Output Bit 0/1 Controller G D S NMOS GPIO Output Bit 1 Controller 0 GPIO Output Pin Output is Floating NMOS GPIO Output Pin Output is Grounded GPIO Output Bit 0 Controller 1 NMOS Current 25
Figure 4.1 Lecture Note of Push-Pull	Figure 4.2 Lecture Note of Open-Drain

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 common way to solve 'bouncing' is to give 'waiting' and wait until the end of 'bouncing' of the switch. However, in this lab, I tried to solve the 'bouncing' of the switch using 'if-else Statement' without using the 'waiting' function. For this, I began to look at the button being pressed and removed as a one flow. The case when the button is maintained pressed was treated by 'if Statement' so that 'waiting' is implemented until the button is removed. Eventually, through this, 'Switch Bouncing' was solved in software.

Meanwhile, let's think about a general hardware solution that can solve 'switch bouncing'. One common solution is to connect resistor and capacitor to bread-board. That is, 'switch bouncing' can be solved by time constant, τ, which is a multiplication of a resistor and a capacitor. In addition, there is a method through the 'Schmitt Trigger Circuit' among the solutions of 'bouncing' using resistor and capacitor.

3) Check the output pin with oscilloscope and observe how the signals change with input button

   Due to the corona situation, laboratory use is restricted, and so the check of the output value through the oscilloscope is skipped.

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….

Output Observation

- Since there were no four 'LED' pins, the code could not be physically implemented. Accordingly, it was confirmed that four LED pins were sequentially turned on according to the pressing of the button through code debugging. The debugging results are summarized as follows.

  Default -> button pressed -> only 'GPIOA_ODR5' on -> button pressed -> only 'GPIOA_ODR6' on -> button pressed -> only 'GPIOA_ODR7' on -> button pressed -> only 'GPIOB_ODR6' on -> button pressed -> only 'GPIOA_ODR5' on -> …

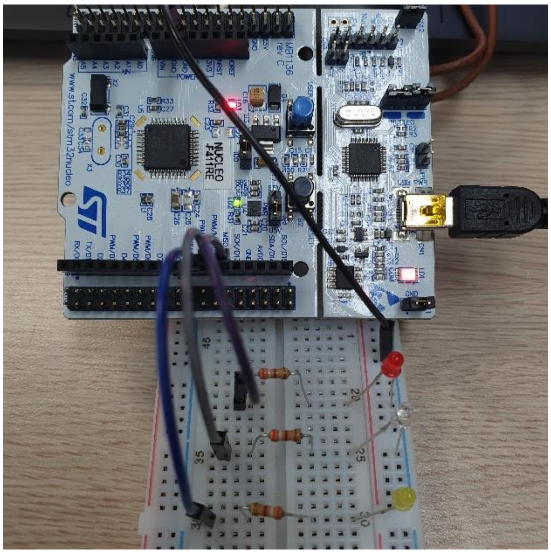
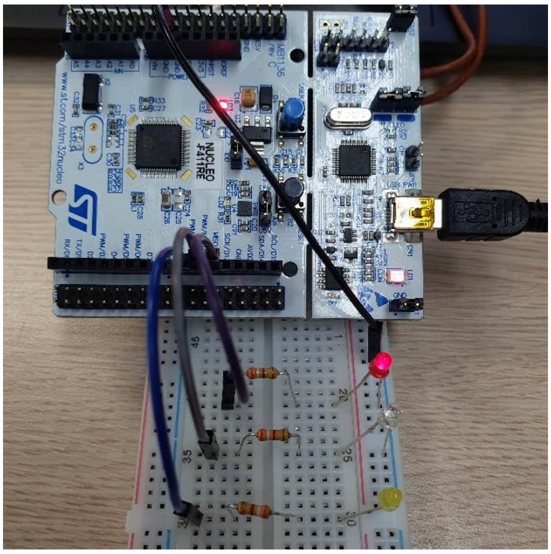
	

Figure 5.1 Multiple LED Sequence(1)	Figure 5.2 Multiple LED Sequence(2)

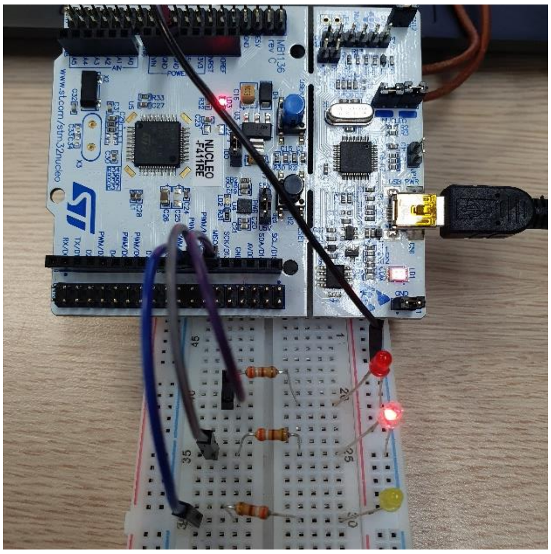


Figure 5.3 Multiple LED Sequence(3)

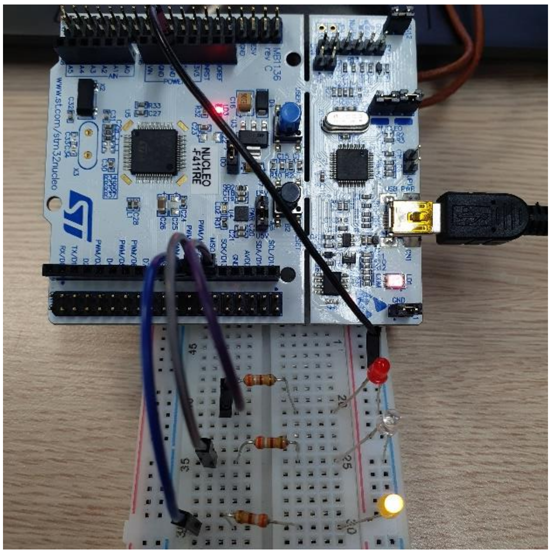
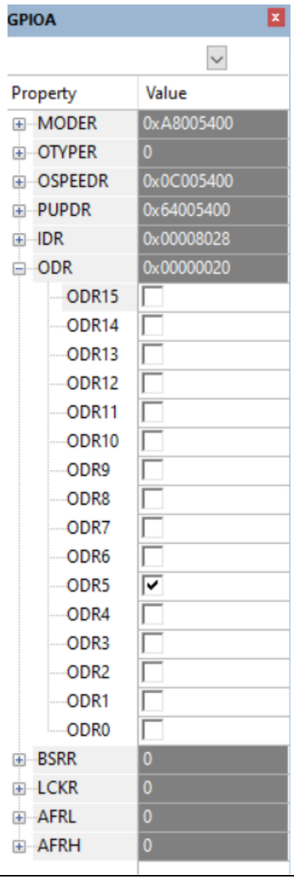


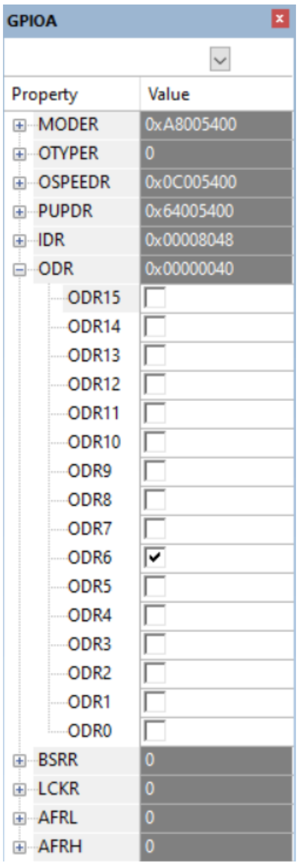
Figure 5.4 Multiple LED Sequence(4)



GPIOA

Property	Value
MODER	0xA8005400
OTYPER	0
OSPEEDR	0x0C005400
PUPDR	0x64005400
IDR	0x00008028
ODR	0x00000020
ODR15	□
ODR14	□
ODR13	□
ODR12	□
ODR11	□
ODR10	□
ODR9	□
ODR8	□
ODR7	□
ODR6	□
ODR5	☑
ODR4	□
ODR3	□
ODR2	□
ODR1	□
ODR0	□
BSRR	0
LCKR	0
AFRL	0
AFRH	0

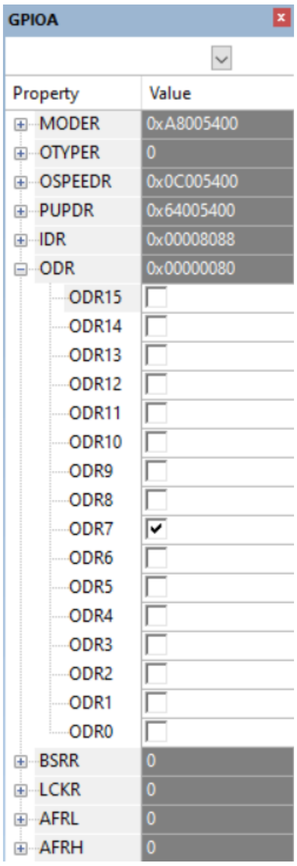
Figure 5.5 Debug_1



GPIOA

Property	Value
MODER	0xA8005400
OTYPER	0
OSPEEDR	0x0C005400
PUPDR	0x64005400
IDR	0x00008048
ODR	0x00000040
ODR15	□
ODR14	□
ODR13	□
ODR12	□
ODR11	□
ODR10	□
ODR9	□
ODR8	□
ODR7	□
ODR6	☑
ODR5	□
ODR4	□
ODR3	□
ODR2	□
ODR1	□
ODR0	□
BSRR	0
LCKR	0
AFRL	0
AFRH	0

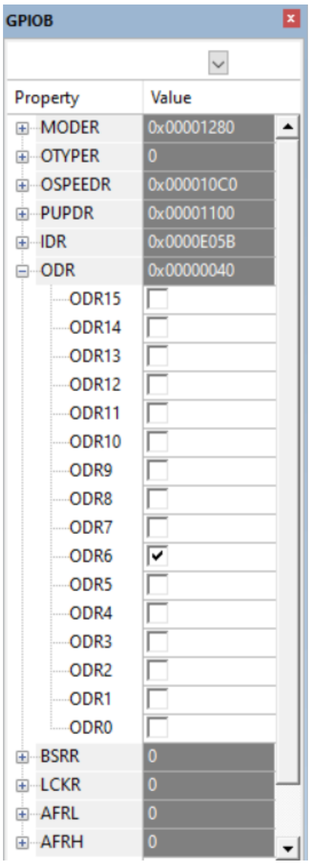
Figure 5.6 Debug_2



GPIOA

Property	Value
MODER	0xA8005400
OTYPER	0
OSPEEDR	0x0C005400
PUPDR	0x64005400
IDR	0x00008088
ODR	0x00000080
ODR15	□
ODR14	□
ODR13	□
ODR12	□
ODR11	□
ODR10	□
ODR9	□
ODR8	□
ODR7	☑
ODR6	□
ODR5	□
ODR4	□
ODR3	□
ODR2	□
ODR1	□
ODR0	□
BSRR	0
LCKR	0
AFRL	0
AFRH	0

Figure 5.7 Debug_3



GPIOB

Property	Value
MODER	0x00001280
OTYPER	0
OSPEEDR	0x000010C0
PUPDR	0x00001100
IDR	0x0000E05B
ODR	0x00000040
ODR15	□
ODR14	□
ODR13	□
ODR12	□
ODR11	□
ODR10	□
ODR9	□
ODR8	□
ODR7	□
ODR6	☑
ODR5	□
ODR4	□
ODR3	□
ODR2	□
ODR1	□
ODR0	□
BSRR	0
LCKR	0
AFRL	0
AFRH	0

Figure 5.8 Debug_4

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

Circuit Diagram

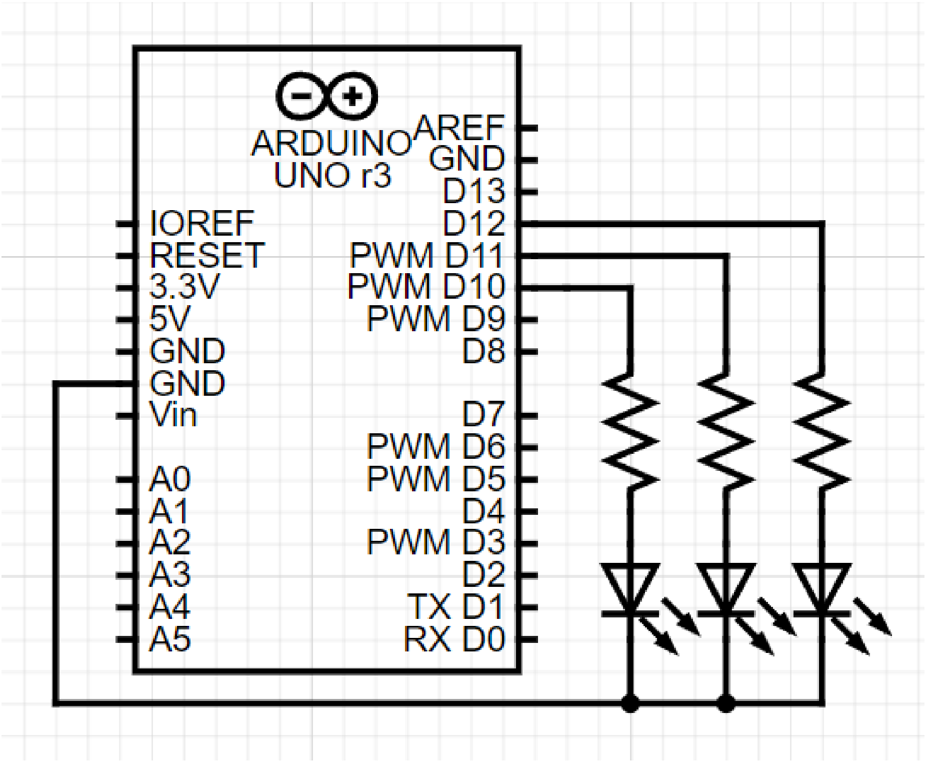


Figure 5.9 Circuit Diagram of Part(3)

Flow Chart

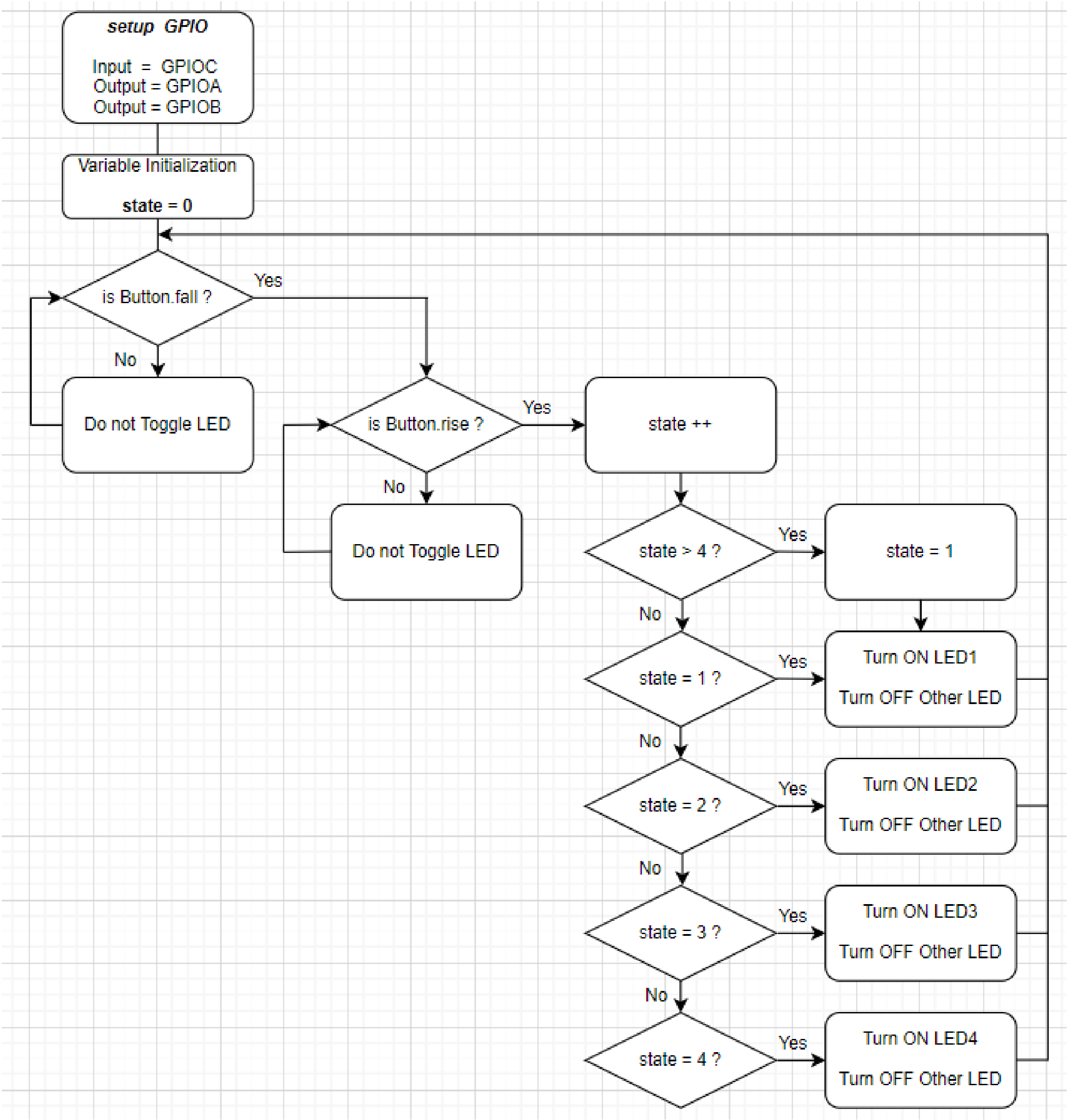
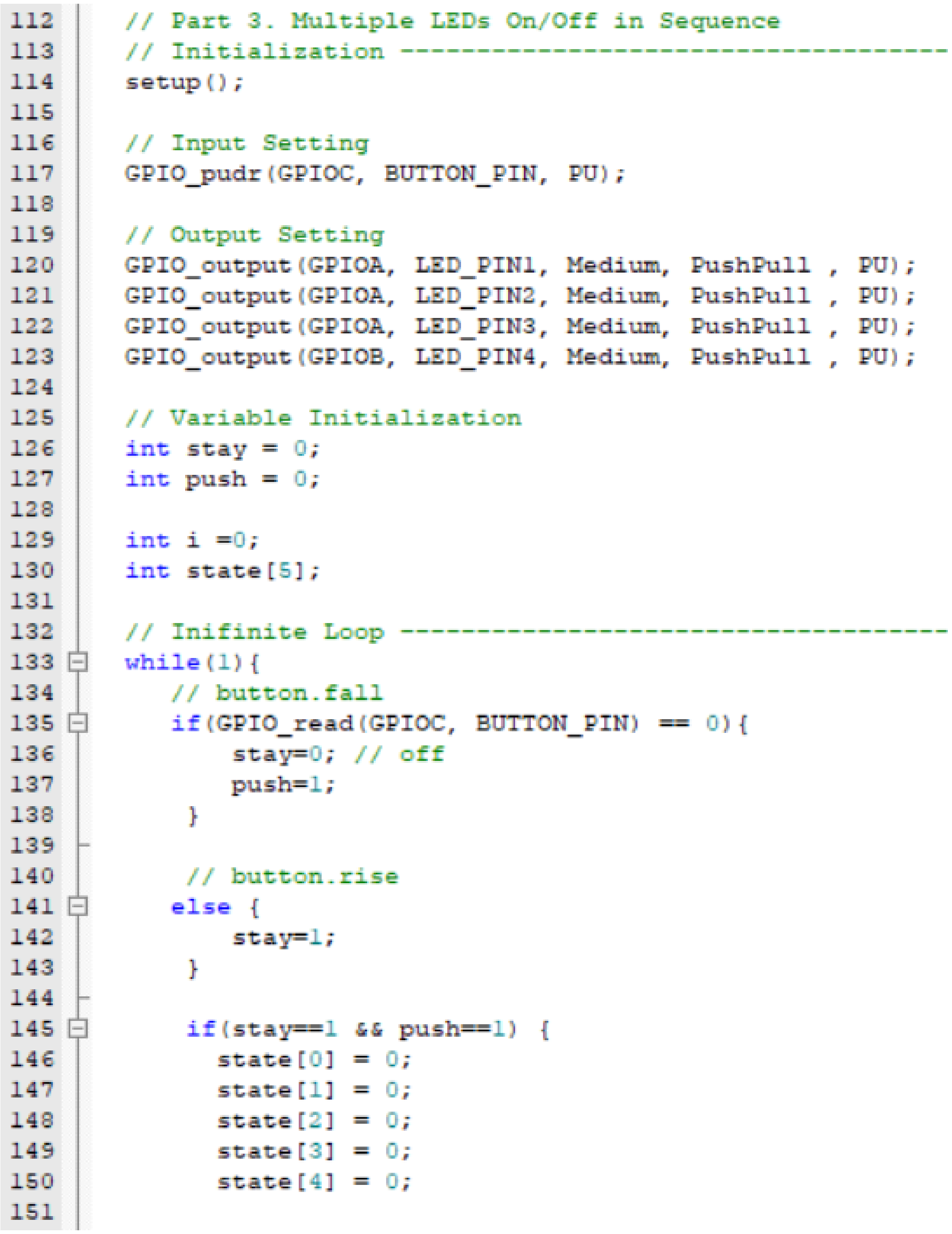


Figure 5.10 Flow-Chart of Part(3)

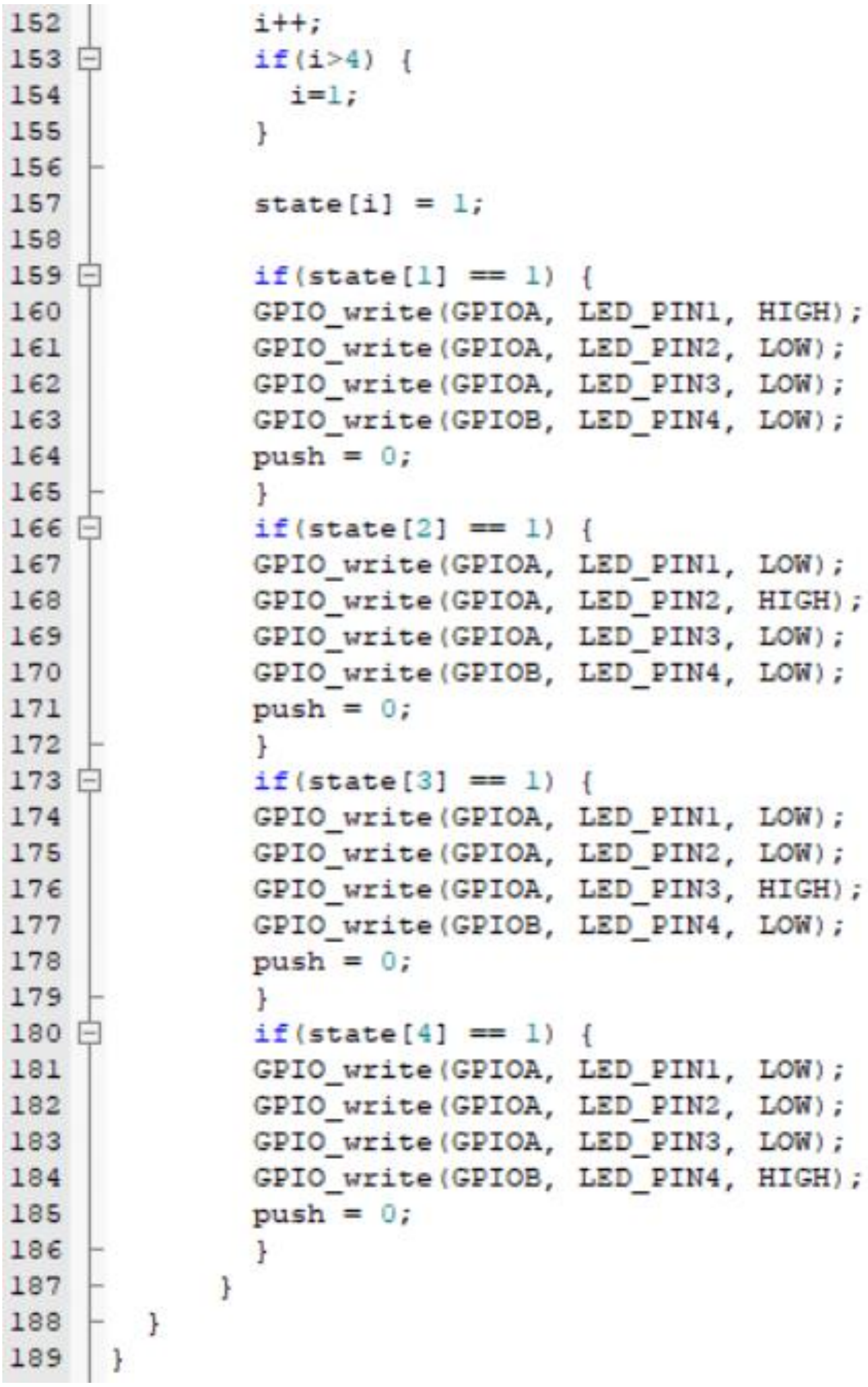
Source code

LAB_GPIO_DigitialInOut_multipleLED.c



```c
// Part 3. Multiple LEDs On/Off in Sequence
// Initialization ------------------------------------
setup();

// Input Setting
GPIO_pudr(GPIOC, BUTTON_PIN, PU);

// Output Setting
GPIO_output(GPIOA, LED_PIN1, Medium, PushPull , PU);
GPIO_output(GPIOA, LED_PIN2, Medium, PushPull , PU);
GPIO_output(GPIOA, LED_PIN3, Medium, PushPull , PU);
GPIO_output(GPIOB, LED_PIN4, Medium, PushPull , PU);

// Variable Initialization
int stay = 0;
int push = 0;

int i =0;
int state[5];

// Inifinite Loop ------------------------------------
while(1){
   // button.fall
   if(GPIO_read(GPIOC, BUTTON_PIN) == 0){
       stay=0; // off
       push=1;
    }

    // button.rise
   else {
       stay=1;
    }

    if(stay==1 && push==1) {
      state[0] = 0;
      state[1] = 0;
      state[2] = 0;
      state[3] = 0;
      state[4] = 0;

```

Figure 5.1 Source Code of DigitialInOut_multipleLED



```
        i++;
        if(i>4) {
          i=1;
        }

        state[i] = 1;

        if(state[1] == 1) {
        GPIO_write(GPIOA, LED_PIN1, HIGH);
        GPIO_write(GPIOA, LED_PIN2, LOW);
        GPIO_write(GPIOA, LED_PIN3, LOW);
        GPIO_write(GPIOB, LED_PIN4, LOW);
        push = 0;
        }
        if(state[2] == 1) {
        GPIO_write(GPIOA, LED_PIN1, LOW);
        GPIO_write(GPIOA, LED_PIN2, HIGH);
        GPIO_write(GPIOA, LED_PIN3, LOW);
        GPIO_write(GPIOB, LED_PIN4, LOW);
        push = 0;
        }
        if(state[3] == 1) {
        GPIO_write(GPIOA, LED_PIN1, LOW);
        GPIO_write(GPIOA, LED_PIN2, LOW);
        GPIO_write(GPIOA, LED_PIN3, HIGH);
        GPIO_write(GPIOB, LED_PIN4, LOW);
        push = 0;
        }
        if(state[4] == 1) {
        GPIO_write(GPIOA, LED_PIN1, LOW);
        GPIO_write(GPIOA, LED_PIN2, LOW);
        GPIO_write(GPIOA, LED_PIN3, LOW);
        GPIO_write(GPIOB, LED_PIN4, HIGH);
        push = 0;
        }
      }
  }
}
```

Figure 5.2 Source Code of DigitialInOut_multipleLED

III. Conclusion & TroubleShooting

Conclusion

In this lab, I implemented and studied about ‘HAL Drivers’ that can control the GPIO digital in and out by using NUCLEO-F411RE and Keil uVision IDE. In this process, I was able to think about the basic structure of the register and the bouncing of the switch.

Theoretically, I was able to solve the bounding through 'wait' or 'delay', but because I haven't learned it yet, I implemented it through 'if-statement'. If there are cases where the bounding of the switch needs to be solved next time I think it is worth considering implementing it using 'delay'.

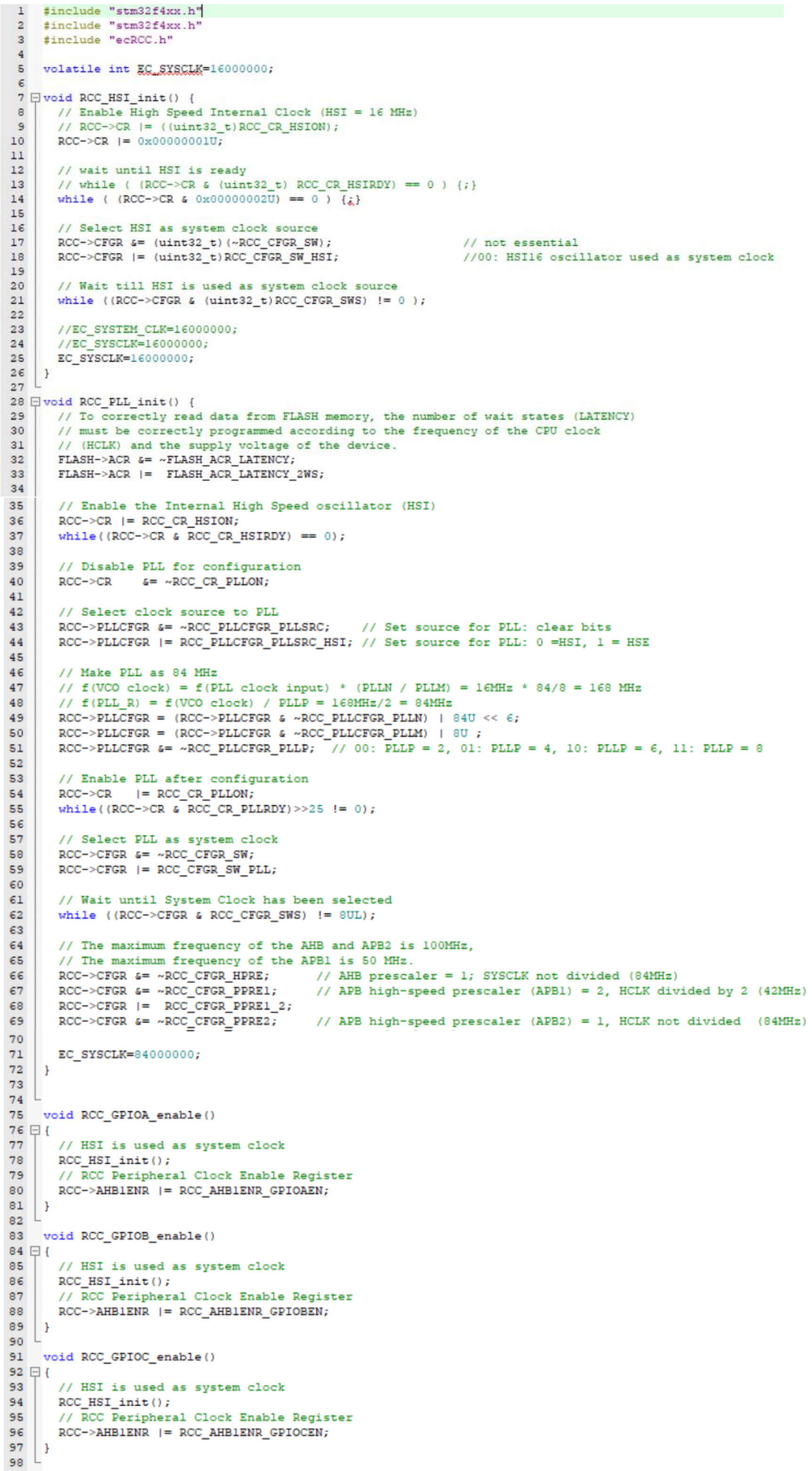
TroubleShooting

It was difficult to recognize only once that the button was pressed in Infinite loop, while (1). At first, I tried to turn on and off 'LD2' as soon as the button was pressed. However, in this case, 'if-statement' that recognizes that the button was pressed and 'if-statement' that lights 'LD2' collided, and 'LD2' was rarely toggled properly.

I felt the need to allow the conditional sentence that recognizes that the button is pressed and the conditional sentence that toggle the light of 'LD2' to be executed independently. Accordingly, I began to look at the button being pressed and removed as a one flow. That is, if the user stays while pressing the button, only the 'if-statement' that recognizes that the button has been pressed is executed. And, the moment when the user releases his or her hand from the button was treated as the phrase 'else'. The value of variable, In this 'else' phrase, was set to satisfy the 'if-statement' that toggle 'LD2'. Through this, as soon as the user releases his or her hand from the button, the toggle condition statement was immediately executed to toggle 'LD2'.

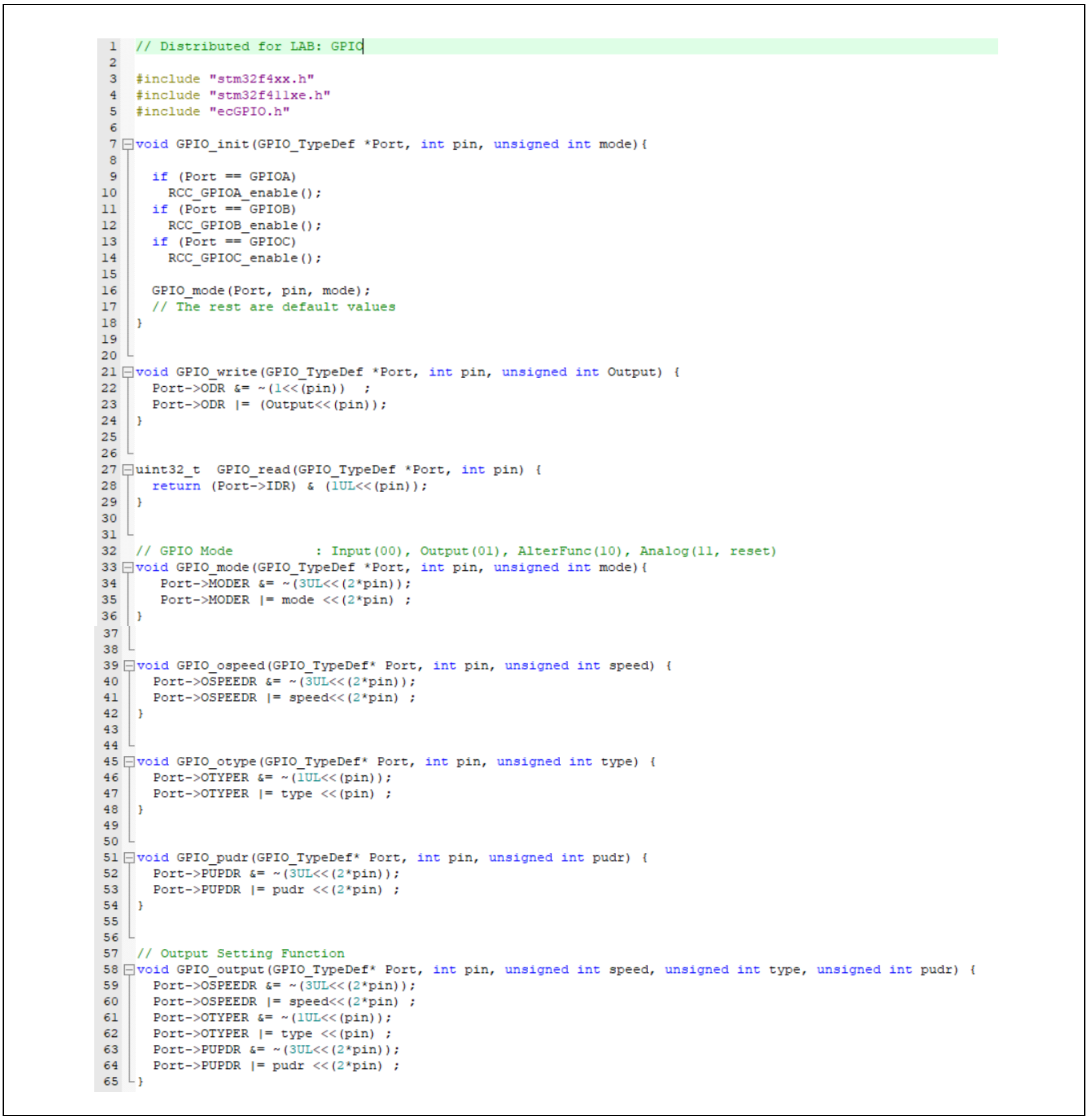
Appendix

Source file: ecRCC.c



```c
#include "stm32f4xx.h"
#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;
}
```

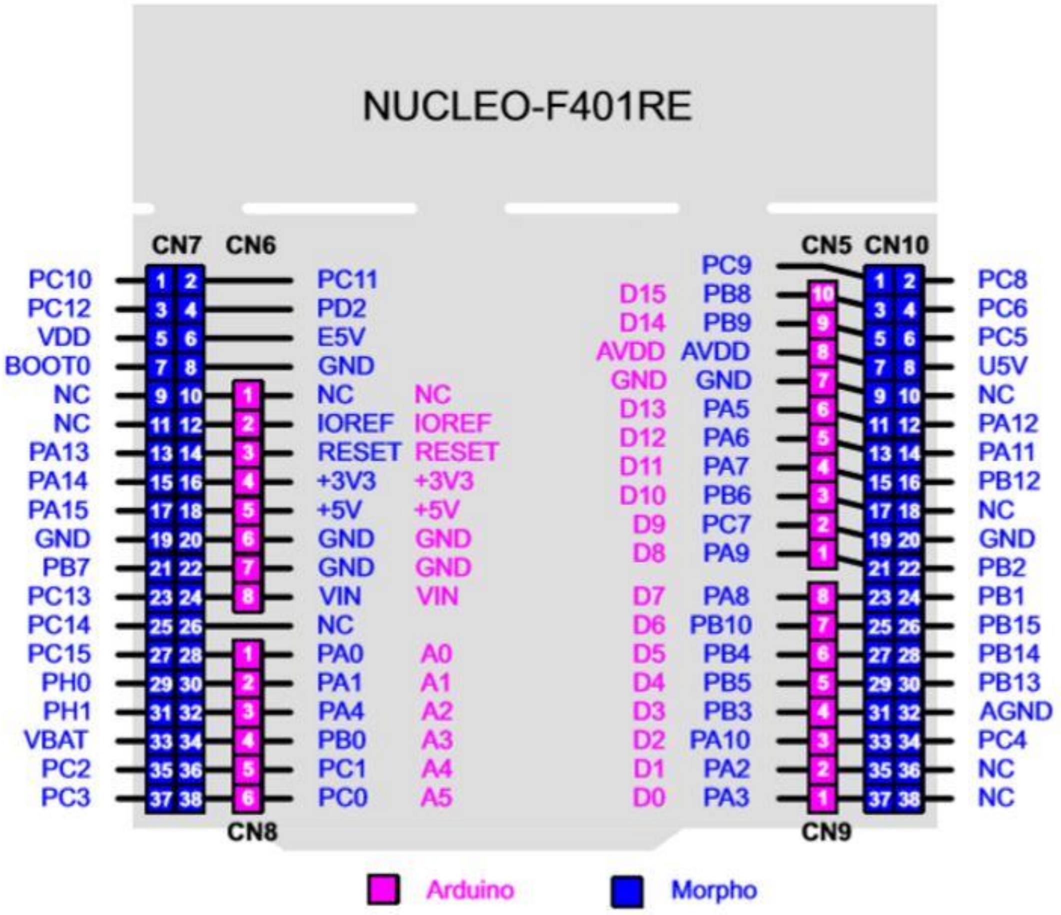
Source file: ecGPIO.c



```c
// Distributed for LAB: GPIO

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

void GPIO_init(GPIO_TypeDef *Port, int pin, unsigned int mode){

  if (Port == GPIOA)
    RCC_GPIOA_enable();
  if (Port == GPIOB)
    RCC_GPIOB_enable();
  if (Port == GPIOC)
    RCC_GPIOC_enable();

  GPIO_mode(Port, pin, mode);
  // The rest are default values
}


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


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


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


void GPIO_ospeed(GPIO_TypeDef* Port, int pin, unsigned int speed) {
  Port->OSPEEDR &= ~(3UL<<(2*pin));
  Port->OSPEEDR |= speed<<(2*pin) ;
}


void GPIO_otype(GPIO_TypeDef* Port, int pin, unsigned int type) {
  Port->OTYPER &= ~(1UL<<(pin));
  Port->OTYPER |= type <<(pin) ;
}


void GPIO_pudr(GPIO_TypeDef* Port, int pin, unsigned int pudr) {
  Port->PUPDR &= ~(3UL<<(2*pin));
  Port->PUPDR |= pudr <<(2*pin) ;
}


// Output Setting Function
void GPIO_output(GPIO_TypeDef* Port, int pin, unsigned int speed, unsigned int type, unsigned int pudr) {
  Port->OSPEEDR &= ~(3UL<<(2*pin));
  Port->OSPEEDR |= speed<<(2*pin) ;
  Port->OTYPER &= ~(1UL<<(pin));
  Port->OTYPER |= type <<(pin) ;
  Port->PUPDR &= ~(3UL<<(2*pin));
  Port->PUPDR |= pudr <<(2*pin) ;
}
```

Reference Table: NUCLEO-F401RE

Figure 18. NUCLEO-F401RE


NUCLEO-F401RE
CN7 CN6
CN5 CN10
PC10 1 2 PC11
PC12 3 4 PD2
VDD 5 6 E5V
BOOT0 7 8 GND
NC 9 10 1 NC NC
NC 11 12 2 IOREF IOREF
PA13 13 14 3 RESET RESET
PA14 15 16 4 +3V3 +3V3
PA15 17 18 5 +5V +5V
GND 19 20 6 GND GND
PB7 21 22 7 GND GND
PC13 23 24 8 VIN VIN
PC14 25 26 NC
PC15 27 28 1 PA0 A0
PH0 29 30 2 PA1 A1
PH1 31 32 3 PA4 A2
VBAT 33 34 4 PB0 A3
PC2 35 36 5 PC1 A4
PC3 37 38 6 PC0 A5
CN8
PC9 1 2 PC8
D15 PB8 10 3 4 PC6
D14 PB9 9 5 6 PC5
AVDD AVDD 8 7 8 U5V
GND GND 7 9 10 NC
D13 PA5 6 11 12 PA12
D12 PA6 5 13 14 PA11
D11 PA7 4 15 16 PB12
D10 PB6 3 17 18 NC
D9 PC7 2 19 20 GND
D8 PA9 1 21 22 PB2
D7 PA8 8 23 24 PB1
D6 PB10 7 25 26 PB15
D5 PB4 6 27 28 PB14
D4 PB5 5 29 30 PB13
D3 PB3 4 31 32 AGND
D2 PA10 3 33 34 PC4
D1 PA2 2 35 36 NC
D0 PA3 1 37 38 NC
CN9
Arduino
Morpho