LAB REPORT

Digital In/Out - 7Segment Display

Student ID	21700791	**Subject**	Embedded Controller
Name	Hong Se Hyun	**Professor**	Kim Young Keun
Partner Name	Kim Sang Hyun	**Date**	2021.10.13.

Table of Contents

I. Introduction

In this lab, I was required to create a simple program to control a 7-segment display to show a decimal number (0~9).

Hardware

NUCLEO -F411RE

One 7-segment display(5101ASR), array resistor (330 ohm), breadboard

Software

Keil uVision IDE, CMSIS, EC_HAL

II. Procedure

A. 7-Segment

Review 7-segment Decoder and Display from Digital Logic lecture. Also, can refer this.

Popular BCD 7-segment decoder chip are 74LS47, CD4511. Here, I am going to make the 7-segment decoder by the MCU programming.

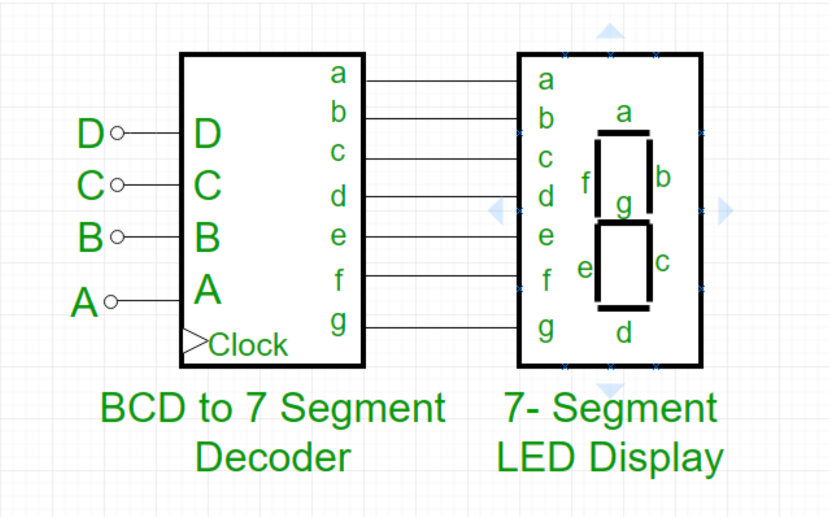


Figure1.1 7-Segment LED Display

Discussion

1) Draw the truth table for the BCD 7-segment decoder.

$Input$				$Output$								$Display$
x_4	x_3	x_2	x_1	a	b	c	d	e	f	g	dp	
0	0	0	0	1	1	1	1	1	1	0	0	0
0	0	0	1	0	1	1	0	0	0	0	0	1
0	0	1	0	1	1	0	1	1	0	1	0	2
0	0	1	1	1	1	1	1	0	0	1	0	3
0	1	0	0	0	1	1	0	0	1	1	0	4
0	1	0	1	1	0	1	1	0	1	1	0	5
0	1	1	0	1	0	1	1	1	1	1	0	6
0	1	1	1	1	1	1	0	0	1	0	0	7
1	0	0	0	1	1	1	1	1	1	1	0	8
1	0	0	1	1	1	1	1	0	1	1	0	9
1	0	1	0	X	X	X	X	X	X	X	X	$InValid$
1	0	1	1	X	X	X	X	X	X	X	X	$InValid$
1	1	0	0	X	X	X	X	X	X	X	X	$InValid$
1	1	0	1	X	X	X	X	X	X	X	X	$InValid$
1	1	1	0	X	X	X	X	X	X	X	X	$InValid$
1	1	1	1	X	X	X	X	X	X	X	X	$InValid$

Table1.1 Truth Table of BCD 7-segment decoder

The Truth Table above explains the 7-Segment of the Cathode Type.

In the case of 7-Segment in the Anode Type used in this lab, we have to change ‘High’ to ‘Low’ and ’Low’ to ‘High’ in the ‘Truth Table’ above.

2) What are the common cathode and common anode of 7-segment?

Common Cathode is a method of connecting ground to Common Pin. In addition, in order to light the LED in the Cathode type, High must be given as an input. The reason is that when Low is applied, there is no potential difference in the circuit, so that current cannot flow. Therefore, by applying High to each pin, a potential difference is created and a current suitable for the diode direction can flow.

Common Anode is a method of connecting V_{cc} to Common Pin. In addition, in order to light the LED in the Anode type, 'Low' must be given as an input. The reason is that when High is applied, there is no potential difference in the circuit, so that current cannot flow. Therefore, by applying Low to each pin, a potential difference is created and a current suitable for the diode direction can flow.

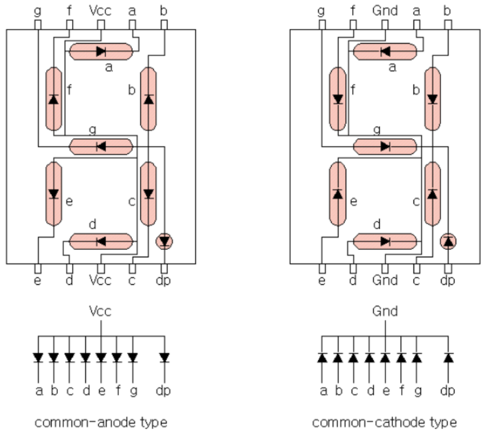


Figure1.2 Comparison Anode Cathode[1]

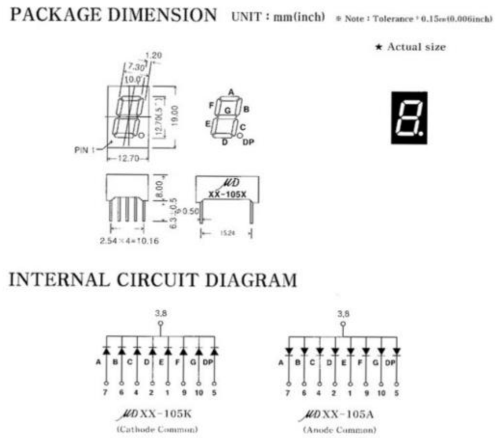


Figure1.3 Type of 7-Segment[2]

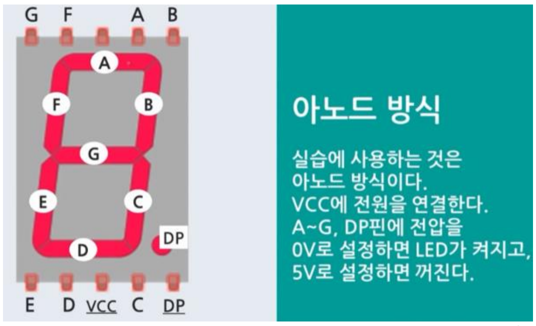


Figure1.4 Anode Type of 7-Segment[3]

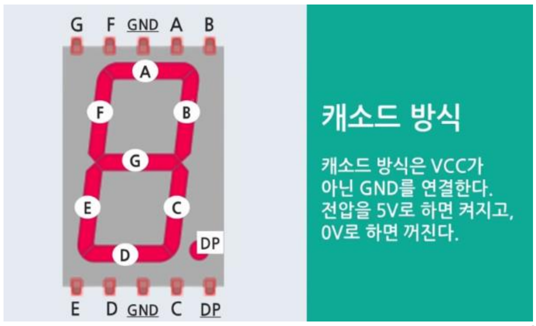


Figure1.5 Cathode Type of 7-Segment[3]

3) This is common anode 7-segment. Does the LED turn on when output pin from MCU is 'High'?

The 7-Segment used in this Lab is Anode Type, so when High was given as Input, it did not turn on. The reason is that Vcc is applied to Common Pin, so if High is given to each pin, there will be no potential difference and no current will flow.

---

1 Figure1.2 From here : https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=jamduino&logNo=220932416728

2 Figure1.3 From here : https://www.devicemart.co.kr/goods/view?no=11551

3 Figure1.4 and Figure1.5 From here : https://www.youtube.com/watch?v=SRtYpBnmPfA

4) Find out how to connect a 7-segment to MCU pins with current limiting resistors.

Resistance is not connected to the two common pins to which V_{cc} is applied. However, If the other eight pins are connected to the MCU pins without resistance, there is a risk of overcurrent flowing, so resistance must be used. Therefore, the resistance was connected to each pin more conveniently and neatly using the array resistor.

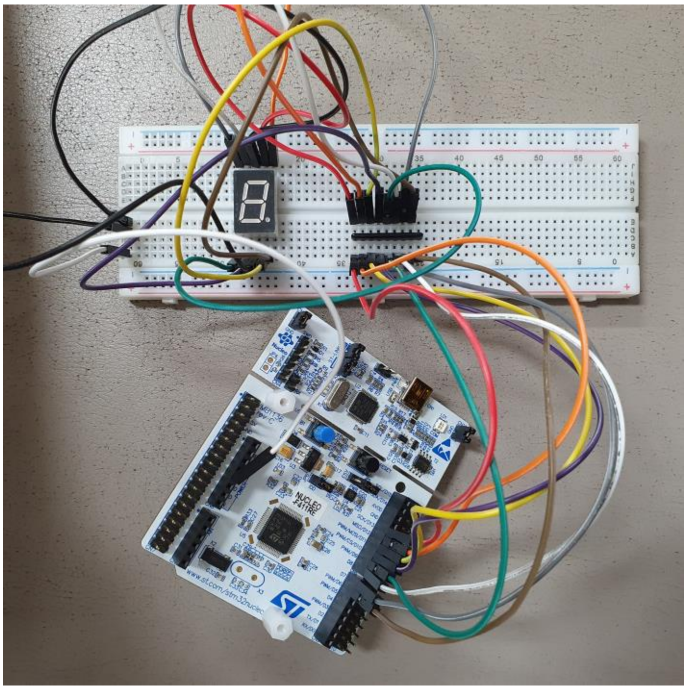


Figure1.6 Circuit Composition

B. Configuration

Create a new project named as **"LAB_GPIO_7segment".**

Name the source file as **"LAB_GPIO_7segment.cpp"**

Configuration Input and Output pins

Digital In: Button	Digital Out:
GPIOC, Pin 13 Digital Input Set PULL-UP	PA5, PA6, PA7, PB6, PC7, PA9, PA8, PB10 Digital Output Push-Pull No Pull-up Pull-down Fast

Fill in the table

Port/Pin	Description	Register setting
Port A Pin 5	Clear Pin5 mode	GPIOA→MODER &= ~(3<<(5*2))
Port A Pin 5	Set Pin5 mode = Output	GPIOA→MODER |= (1<<(5*2))
Port A Pin 6	Clear Pin6 mode	GPIOA→MODER &= ~(3<<(6*2))
Port A Pin 6	Set Pin6 mode = Output	GPIOA→MODER |= (1<<(6*2))
Port A Pin Y	Clear PinY mode	GPIOA→MODER &= ~(3<<(Y*2))
Port A Pin Y	Set PinY mode = Output	GPIOA→MODER |= (1<<(Y*2))
Port A Pin 5~9	Clear Pin5~9 mode	GPIOA→MODER &= ~ ($(0011\ 1111\ 1111_2)$ <<(5*2))
	Set Pin5~9 mode = Output	GPIOA→MODER |= ($(0001\ 0101\ 0101_2)$ <<(5*2))
Port X Pin Y	Clear PinY mode	GPIOX→MODER &= ~(3<<(Y*2))
	Set PinY mode = Output	GPIOX→MODER |= (1<<(Y*2))
Port A Pin 5	Set Pin5 otype = push-pull	GPIOA→OTYPER &= ~(1<<5)
Port A Pin Y	Set PinY otype = push-pull	GPIOA→OTYPER &= ~(1<<Y)
Port A Pin 5	Set Pin5 ospeed = Fast	GPIOA→OSPEEDR &= ~(3<<(5*2)) GPIOA→OSPEEDR |= (2<<(5*2))
Port A Pin Y	Set PinY ospeed = Fast	GPIOA→OSPEEDR &= ~(3<<(Y*2)) GPIOA→OSPEEDR |= (2<<(Y*2))
Port A Pin 5	Set Pin5 PUPD = no Pullup/down	GPIOA→PUPDR &= ~(3<<(5*2))
Port A Pin Y	Set PinY PUPD = no Pullup/down	GPIOA→PUPDR &= ~(3<<(Y*2))

For 'register setting', refer to 'reference'.

C. Create EC_HAL functions

Specific for given Output Pins

Include File	Function
ecGPIO.h.c	void sevensegment.init(void);
	void sevensegment.decoder(uint8 num);

Source code

ecGPIO.h *(Attached only LAB3 Code)*

```
// LAB3. 7 Segment
void sevensegment_init(int pin1, int pin2, int pin3, int pin4, int pin5, int pin6, int pin7, int pin8);
void sevensegment_decode(unsigned int cnt);
```

Figure2.1 Source Code of ecGPIO

ecGPIO.c

```
// LAB3. 7 Segment
void sevensegment_init(int pin1, int pin2, int pin3, int pin4, int pin5, int pin6, int pin7, int pin8) {
  GPIO_init(GPIOA, pin1, OUTPUT);    // calls RCC_GPIOA_enable()
  GPIO_init(GPIOA, pin2, OUTPUT);    // calls RCC_GPIOA_enable()
  GPIO_init(GPIOA, pin3, OUTPUT);    // calls RCC_GPIOA_enable()
  GPIO_init(GPIOB, pin4, OUTPUT);    // calls RCC_GPIOB_enable()
  GPIO_init(GPIOC, pin5, OUTPUT);    // calls RCC_GPIOC_enable()
  GPIO_init(GPIOA, pin6, OUTPUT);    // calls RCC_GPIOA_enable()
  GPIO_init(GPIOA, pin7, OUTPUT);    // calls RCC_GPIOA_enable()
  GPIO_init(GPIOB, pin8, OUTPUT);    // calls RCC_GPIOB_enable()
}


void sevensegment_decode(unsigned int cnt) {

      unsigned int SEGnum[11][8]={
                    {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
                    {0,0,0,1,1,0,1,1},          //seven
                    {0,0,0,0,0,0,0,1},          //eight
                    {0,0,0,0,1,0,0,1},          //nine
                    {1,1,1,1,1,1,1,0}           //dot
                    };

        GPIO_write(GPIOA, LED_PIN1, SEGnum[cnt][0]);
        GPIO_write(GPIOA, LED_PIN2, SEGnum[cnt][1]);
        GPIO_write(GPIOA, LED_PIN3, SEGnum[cnt][2]);
        GPIO_write(GPIOB, LED_PIN4, SEGnum[cnt][3]);
        GPIO_write(GPIOC, LED_PIN5, SEGnum[cnt][4]);
        GPIO_write(GPIOA, LED_PIN6, SEGnum[cnt][5]);
        GPIO_write(GPIOA, LED_PIN7, SEGnum[cnt][6]);
        GPIO_write(GPIOB, LED_PIN8, SEGnum[cnt][7]);
}
```

Figure2.2 Source Code of ecGPIO

ecRCC.c : See Appendix

ecRCC.h : See Appendix

Documenation of Library

sevensegment_init()

Definition

initializing each led of 7-segment

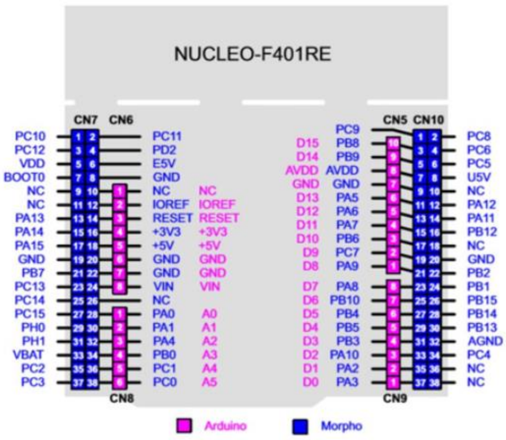
```
void sevensegment_init(int pin1, int pin2, int pin3, int pin4, int pin5, int pin6, int pin7, int pin8)
```

Parameters

- **pin1**: pin number (int) 0~15
- **pin2**: pin number (int) 0~15
- **pin3**: pin number (int) 0~15
- **pin4**: pin number (int) 0~15
- **pin5**: pin number (int) 0~15
- **pin6**: pin number (int) 0~15
- **pin7**: pin number (int) 0~15
- **pin8**: pin number (int) 0~15

  └ In general, pin1, pin2, pin3, and so on is referred to in the order of pins on the board.

  └ For Example, PA5 for pin1 PA6 for pin2 PA7 for pin3 PB6 for pin4, etc.



Example code

```
sevensegment_init(LED_PIN1, LED_PIN2, LED_PIN3, LED_PIN4, LED_PIN5, LED_PIN6, LED_PIN7, LED_PIN8);
                                                                  // Initializing Pin 1 to 8 at once
```

Figure2.3 Documentation of sevensegment_init

sevensegment_decode()

Definition

Controlling each LED in the 7-segment to on/off(High/Low).

```
void sevensegment_decode(unsigned int cnt)
```

Parameters

- **cnt:** unsigned integer value.

  └ In the case of one 7-Segment representing only 0 to 9, the range of cnt is also generally 0 to 9.

Example code

```
sevensegment_decode(cnt % 10);
```

Figure2.4 Documentation of sevensegment_decode

D. Display 0 to 9 with button input

Toggling LED by pushing button

check each number can be displayed, properly

Then, create the code such that whenever the button is pushed, the number should display 0 to 9 and back to 0 and so on.

Circuit Diagram

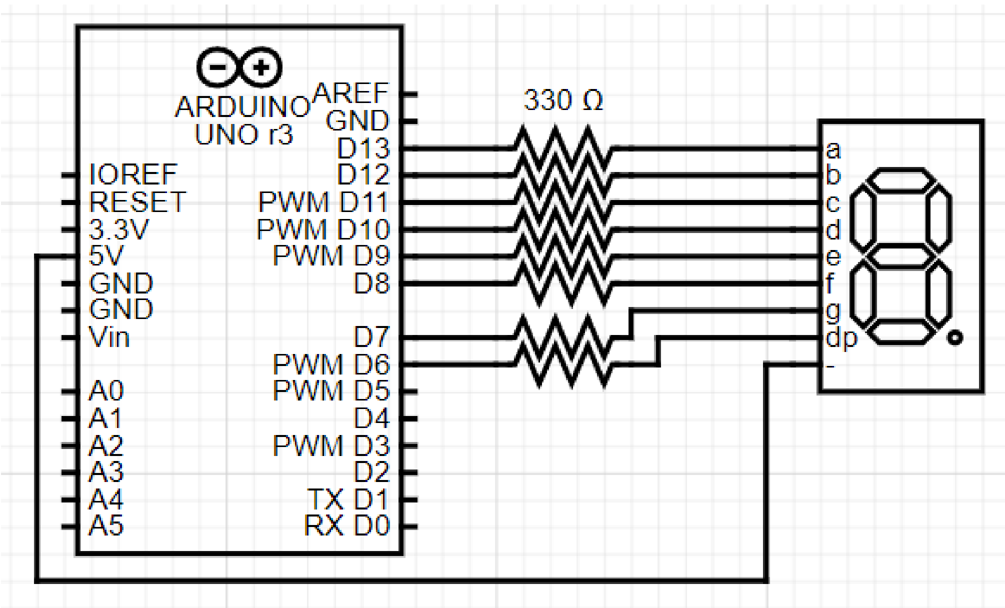


Figure2.5 Circuit Diagram of Lab3

Flow Chart

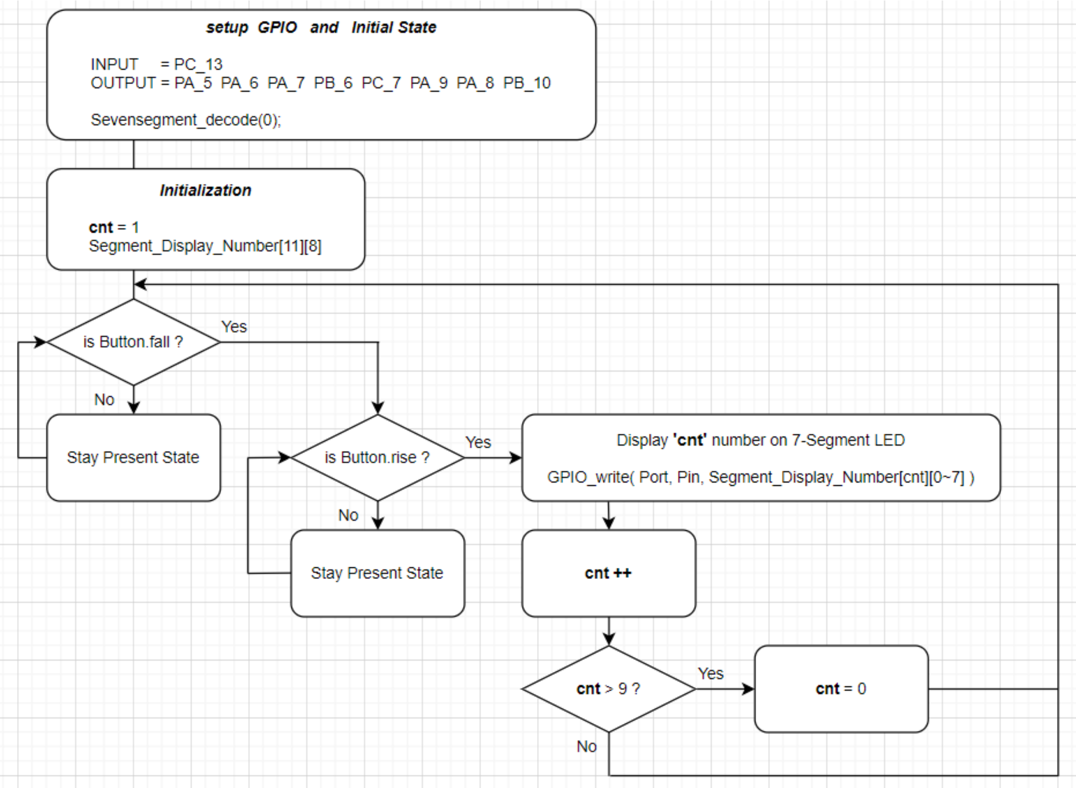


Figure2.6 Flow Chart of Lab3

Source Code (main.c)

```c
int main(void) {
  // Initialiization ---------------------------------------------------------
  setup();

  unsigned int cnt = 1;
  int stay = 0;
  int push = 0;

  // 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) {
        sevensegment_decode(cnt % 10);
        cnt++;
        push = 0;
        if(cnt > 9)    cnt = 0;
    }
  }
}
```

Figure2.7 Code of Toggling the 7-Segment

```c
/*
********************************************************************************
* @author Hong Se Hyun
* @Mod    2021-10-12
* @brief  Embedded Controller:  LAB 7-segment
*           - 7 segment decoder
*            -  PA5, PA6, PA7, PB6, PC7, PA9, PA8, PB10  for DOUT
********************************************************************************
*/

#include "stm32f4xx.h"

//#include "EC_GPIO_API.h"
#include "ecGPIO.h"
#include "ecRCC.h"

//PA5, PA6, PA7, PB6, PC7, PA9, PA8, PB10  for DOUT
#define LED_PIN1   5
#define LED_PIN2   6
#define LED_PIN3   7
#define LED_PIN4   6
#define LED_PIN5   7
#define LED_PIN6   9
#define LED_PIN7   8
#define LED_PIN8   10
#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);
```

Figure2.8 Define Section of MAIN

```c
// Initialiization
void setup(void)
{
  //In Simul , CLK Disable must
  RCC_HSI_init();
  //Initialization Input
  GPIO_init(GPIOC, BUTTON_PIN, INPUT);   // calls RCC_GPIOC_enable()
  //Initialization Output_Segment
  sevensegment_init(LED_PIN1, LED_PIN2, LED_PIN3, LED_PIN4, LED_PIN5, LED_PIN6, LED_PIN7, LED_PIN8);

  //Input Setting
  GPIO_pudr(GPIOC, BUTTON_PIN, PU);
  //Setting Input and Output Mode
  GPIO_output(GPIOA, LED_PIN1, Fast, PushPull , N_PUPD);
  GPIO_output(GPIOA, LED_PIN2, Fast, PushPull , N_PUPD);
  GPIO_output(GPIOA, LED_PIN3, Fast, PushPull , N_PUPD);
  GPIO_output(GPIOB, LED_PIN4, Fast, PushPull , N_PUPD);
  GPIO_output(GPIOC, LED_PIN5, Fast, PushPull , N_PUPD);
  GPIO_output(GPIOA, LED_PIN6, Fast, PushPull , N_PUPD);
  GPIO_output(GPIOA, LED_PIN7, Fast, PushPull , N_PUPD);
  GPIO_output(GPIOB, LED_PIN8, Fast, PushPull , N_PUPD);

  //Initial State == 0, When Reset
  sevensegment_decode(0);
}
```

Figure2.9 Setup Section of MAIN

III. Conclusion

Conclusion

In this lab, I was implemented a simple program to control a 7-segment display to show a decimal number, 0 to 9, by using NUCLEO-F411RE and Keil uVision IDE. In this process, I was able to think about the structure and type of 7-segment.

As in Lab 2, Button's bouncing problem was solved through if-statement because I had not yet learned about delay. However, in the subsequent lab, it would be good to make delay function and write Main more concisely.

TroubleShooting

As in Lab2, The point of this lab is to recognize only once that the button was pressed in infinite loop and resolve the problem of 'switch bouncing'. So, since there was no special trouble-shooting in this lab, I brought back what I wrote in the last lab2.

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 the LED 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 LED collided, and the LED 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 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 the light. Through this, as soon as the user releases his or her hand from the button, the toggle condition statement was immediately executed to toggle the LED.

Appendix

A. Demo Video

See here for detail.

B. ecRCC

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
```

Figure3.1 Code of ecRCC.h

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;
}
```

Figure3.2 Code of ecRCC.c

C. Reference Book

Reference Manual - STM32F411xC/E advanced Arm®-based 32-bit MCUs

8.4.1 GPIO port mode register (GPIOx_MODER) (x = A..E and H)

Address offset: 0x00

Reset values:

- 0xA800 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **MODERy[1:0]:** Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O direction mode.
00: Input (reset state)
01: General purpose output mode
10: Alternate function mode
11: Analog mode

Figure3.3 Reference of GPIO_Moder

8.4.2 GPIO port output type register (GPIOx_OTYPER) (x = A..E and H)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **OTy**: Port x configuration bits (y = 0..15)
These bits are written by software to configure the output type of the I/O port.
0: Output push-pull (reset state)
1: Output open-drain

Figure3.4 Reference of GPIO_OTYPER

8.4.3 GPIO port output speed register (GPIOx_OSPEEDR) (x = A..E and H)

Address offset: 0x08

Reset values:

- 0x0C00 0000 for port A
- 0x0000 00C0 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OSPEEDR15 [1:0]		OSPEEDR14 [1:0]		OSPEEDR13 [1:0]		OSPEEDR12 [1:0]		OSPEEDR11 [1:0]		OSPEEDR10 [1:0]		OSPEEDR9 [1:0]		OSPEEDR8 [1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OSPEEDR7[1:0]		OSPEEDR6[1:0]		OSPEEDR5[1:0]		OSPEEDR4[1:0]		OSPEEDR3[1:0]		OSPEEDR2[1:0]		OSPEEDR1 [1:0]		OSPEEDR0 1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **OSPEEDRy[1:0]:** Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O output speed.
00: Low speed
01: Medium speed
10: Fast speed
11: High speed

Note: Refer to the product datasheets for the values of OSPEEDRy bits versus V_{DD} range and external load.

Figure3.5 Reference of GPIO_OSPPEDR

8.4.4 GPIO port pull-up/pull-down register (GPIOx_PUPDR) (x = A..E and H)

Address offset: 0x0C

Reset values:

- 0x6400 0000 for port A
- 0x0000 0100 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **PUPDRy[1:0]:** Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O pull-up or pull-down

00: No pull-up, pull-down

01: Pull-up

10: Pull-down

11: Reserved

Figure3.6 Reference of GPIO_PUPDR

D. Reference Information

NUCLEO-F401RE

Figure 18. NUCLEO-F401RE

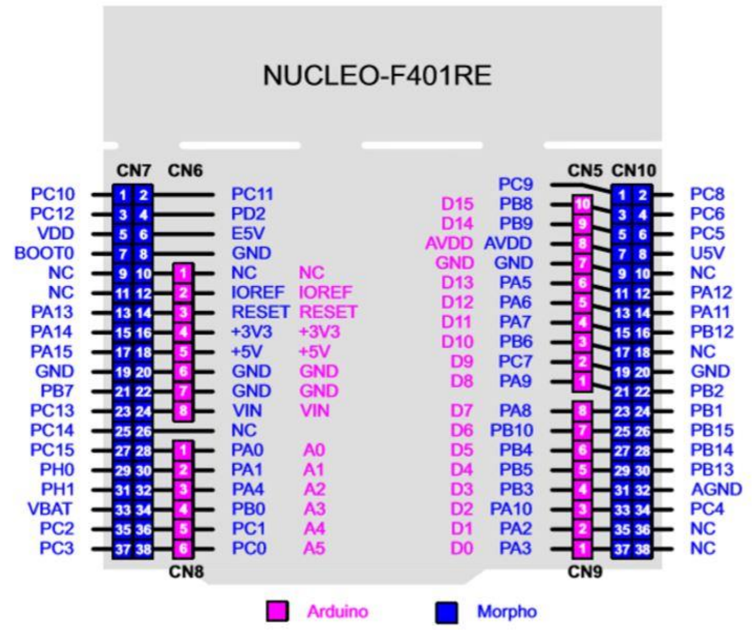


Figure3.7 Reference of NUCLEO Board