LAB REPORT

SysTick and External Interrupt

Control a 7-segment LEDs with Interrupt

Student ID	21700791	**Subject**	Embedded Controller
Name	Hong Se Hyun	**Professor**	Kim Young Keun
Partner Name	Park Jeong Woo	**Date**	2021.10.21.

Table of Contents

I. Introduction

In this lab, I was required to create a simple program that toggle multiple LEDs with a push-button input. I created HAL drivers for GPIO digital in and out control and used these APIs for the lab. I controlled the number by 1 second and displayed it on 7-segment led. Also, I learned how to configure external interrupt (EXTI) to reset the number with push-button

Hardware

NUCLEO -F411RE

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

Software

Keil uVision IDE, CMSIS, EC_HAL

II. Procedure

A. Create HAL, API driver

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

Include File	Function

ecSysTick.h, c	void SysTick_init(uint32_t msec);
	void delay_ms(uint32_t msec);
	uint32_t SysTick_val(void);
	void SysTick_reset (void);
	void SysTick_enable (void);
	void SysTick_disable (void);
	--- 	
	void EXTIx_IRQHandler (void); // in main()
	void SysTick_Handler (void); // in main() or SysTick.h

Include File	Function

ecEXTI.h, c	void EXTI_init(GPIO_TypeDef *port, int pin, int trig_type, int priority);
	void EXTI_enable(uint32_t pin); // mask in IMR
	void EXTI_disable(uint32_t pin); // unmask in IMR
	uint32_t is_pending_EXTI(uint32_t pin);
	void clear_pending_EXTI(uint32_t pin);

Documenation of Library

LED_toggle()

Definition

Code of Toggling the LED by using XOR Logic.

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

Parameters

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

Example code

```
LED_toggle(GPIOA, LED_PIN);
```

Figure1.1 Documentation of LED_toggle

SysTick_init()

Definition

Initializes Systick Interrupt.

In this function, we choose clock for processor clock and down counting method.

```
void SysTick_init(uint32_t msec);
```

Parameters

- **msec:** Value of interval
  └ msec for '1000', means we choose 1ms for interval.

Example code

```
SysTick_init(1000);
```

Figure1.2 Documentation of SysTick_init

delay_ms()

Definition

This Function makes delay by using Systick Interrupt.

```
void delay_ms(uint32_t msec);
```

Parameters

- **msec:** Enter an integer value corresponding to the number want to delay.

Example code

```
delay_ms(1000);
```

Figure1.3 Documentation of delay_ms

SysTick_val()

Definition

Get the Current SysTick Value.

```
uint32_t SysTick_val(void);
```

Example code

```
uint_32t cur_int = SysTick_val();
```

Figure1.4 Documentation of SysTick_val

SysTick_reset()

Definition

Reset the SysTick Value.

```
void SysTick_reset(void);
```

Example code

```
SysTick_reset();
```

Figure1.5 Documentation of SysTick_reset

SysTick_enable()

Definition

Enable SysTick Interrupt.

```
void SysTick_enable(void);
```

Example code

```
SysTick_enable();
```

Figure1.6 Documentation of SysTick_enable

SysTick_disable()

Definition

disable SysTick Interrupt.

```
void SysTick_disable(void);
```

Example code

```
SysTick_disable();
```

Figure1.7 Documentation of SysTick_disable

EXTI15_10_IRQHandler()

Definition

Function that runs when 'button interrupt' occurs.

```
void EXTI15_10_IRQHandler(void);
```

Example code

```
void EXTI15_10_IRQHandler(void) {   }
```

Figure1.8 Documentation of IRQHandler

SysTick_Handler()

Definition

Function of SysTick Interrupt. This function runs periodically.

```
void SysTick_Handler(void);
```

Example code

```
void SysTick_Handler(void) {   }
```

Figure1.9 Documentation of SysTick_Handler

EXTI_init()

Definition

Initializing and set EXTI interrupt.

```
void EXTI_init(GPIO_TypeDef *port, unsigned int pin, unsigned int
trig_type, unsigned int priority);
```

Parameters

- **Port:** Port Number, GPIOA~GPIOH
- **pin**: pin number (int) 0~15
- **trig_type**: RISING, FALLING, BOTH
- **priority**: Set priority. '0' is the first priority implementation.

Example code

```
EXTI_init(GPIOC, BUTTON_PIN, FALLING, 0);
```

Figure1.10 Documentation of EXTI_init

EXTI_enable()

Definition

Set the EXTI Interrupt mask register.

```
void EXTI_enable(uint32_t pin); // mask in IMR
```

Parameters

- **Pin:** pin number (int) 0~15

Example code

```
EXTI_enable(BUTTON_PIN);
```

Figure1.11 Documentation of EXTI_enable

EXTI_disable()

Definition

Unmask(clear) the EXTI Interrupt mask register.

```
void EXTI_disable(uint32_t pin); // unmask in IMR
```

Parameters

- **Pin:** pin number (int) 0~15

Example code

```
EXTI_disable(BUTTON_PIN);
```

Figure1.12 Documentation of EXTI_disable

is_pending_EXTI()

Definition

Determining and checking whether Pending or not.

```
uint32_t is_pending_EXTI(uint32_t pin);
```

Parameters

- **Pin:** pin number (int) 0~15

Example code

```
is_pending_EXTI(BUTTON_PIN);
```

Figure1.13 Documentation of is_pending_EXTI

clear_pending_EXTI()

Definition

Clear Pending Status.

```
void clear_pending_EXTI(uint32_t pin);
```

Parameters

- **Pin:** pin number (int) 0~15

Example code

```
clear_pending_EXTI(BUTTON_PIN);
```

Figure1.14 Documentation of clear_pending

B. LED Toggle with EXTI Button

Use HAL library to toggle LED2 with User button.

Main Code of Part B

```c
/*
******************************************************************************
* @author  SSSLAB
* @Mod     2021-10-15 by SHHnog
* @brief   Embedded Controller:  Tutorial ___SysTick and EXTI
*
******************************************************************************
*/

//**************************************************************************//
//****************                                        ****************//
//****************   PART B.  LED Toggle with EXTI Button   ****************//
//****************                                        ****************//
//**************************************************************************//

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

#include "ecRCC.h"
#include "ecGPIO.h"
#include "ecEXTI.h"
#include "ecSysTick.h"

//#define BUTTON_PIN 13
#define LED_PIN    5

void setup(void);
void EXTI15_10_IRQHandler(void);

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

  // Infinite Loop -------------------------------------------------------------
  while(1){}
}


void EXTI15_10_IRQHandler(void) {
   if (is_pending_EXTI(BUTTON_PIN)) {
     LED_toggle(GPIOA, LED_PIN);
     clear_pending_EXTI(BUTTON_PIN);
  }
}


void setup(void)
{
  // Program setting
  RCC_PLL_init();                          // System Clock = 84MHz
  EXTI_init(GPIOC, BUTTON_PIN, FALLING, 0);
  // Input Setting
  GPIO_init(GPIOC, BUTTON_PIN, INPUT );    // calls RCC_GPIOA_enable
  GPIO_pudr(GPIOC, BUTTON_PIN, PD);
  // Output Setting
  GPIO_init(GPIOA, LED_PIN, OUTPUT);
  GPIO_output(GPIOA, LED_PIN, Fast, PushPull , N_PUPD);
}
```

Figure2.1 Main Code of Part B.

FlowChart

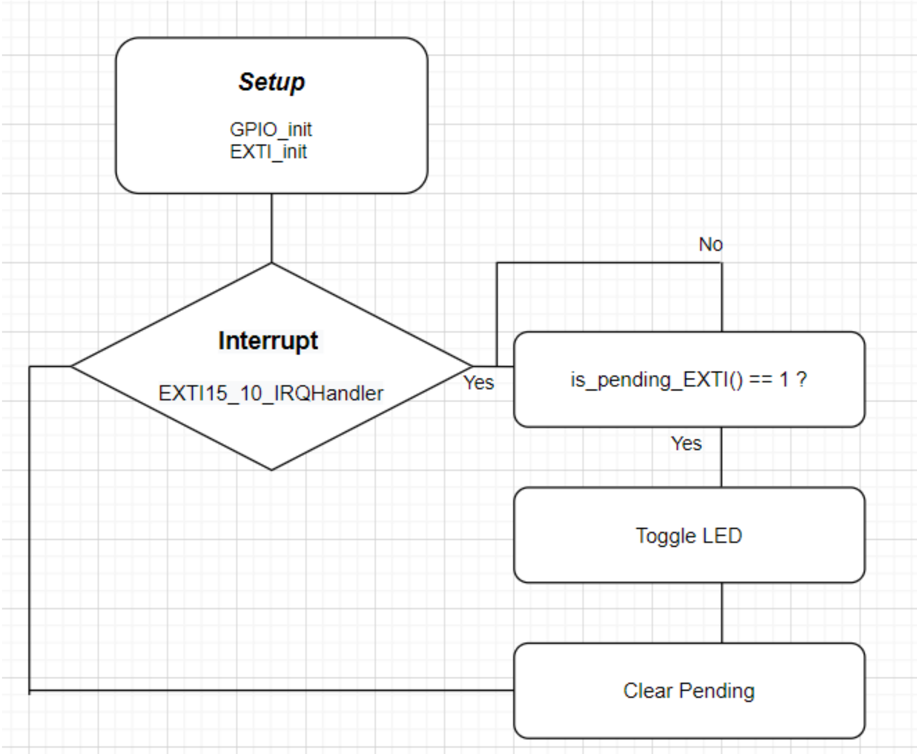

Figure2.2 Flow Chart of Part B.

C. 7-Segment Display with EXTI Button

I Created a new project named as **"LAB_EXTI_SysTick".**

Configuration Input and Output pins

Digital In: Button	Digital Out: LED
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

Display a number in sequence with timer

Displaying the number 0 to 9 on the 7-segment LED at the rate of 1 sec.

After displaying up to 9, then the segment displays ‘0’ and continue counting. When the button is pressed it goes to reset ‘0’ and start counting.

Circuit Diagram

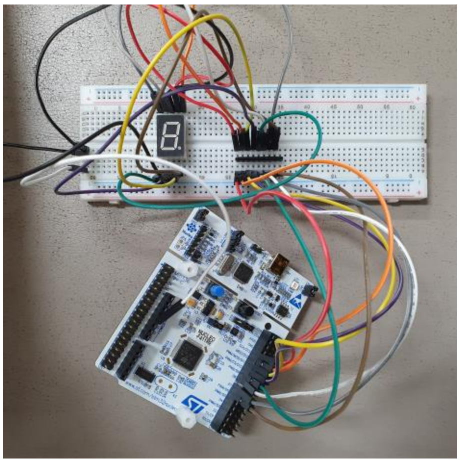

Figure3.1 Circuit Composition

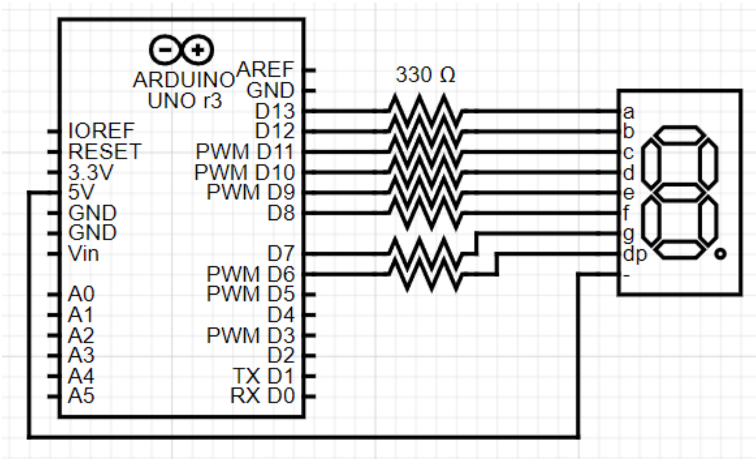

Figure3.2 Circuit Diagram of Lab4

FlowChart

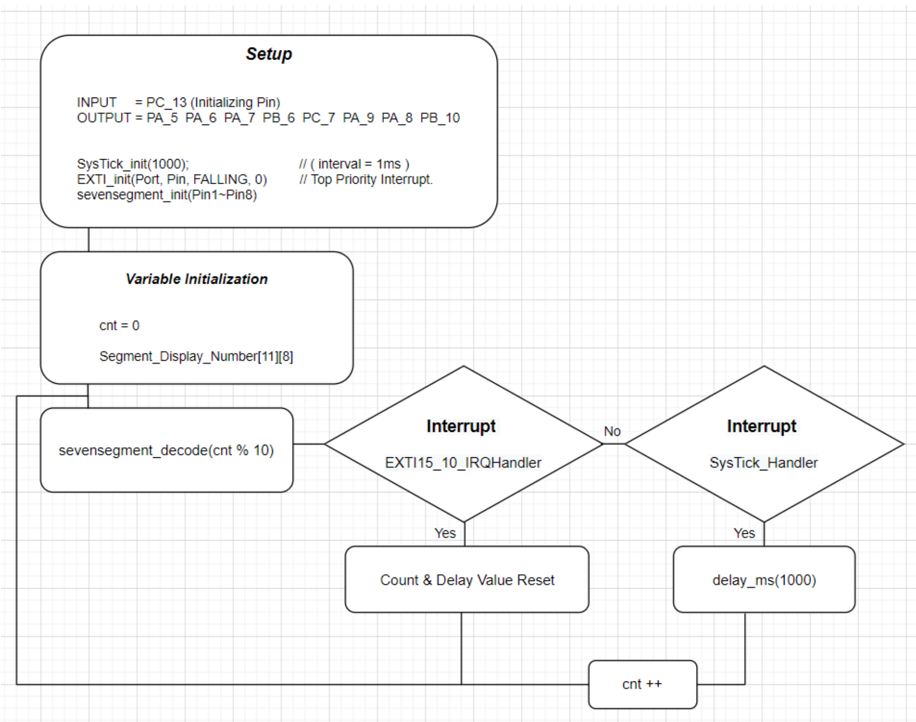

Figure3.3 Flow Chart of Part C.

Main Code of Part C

```c
/*
*********************************************************************************
* @author  SSSLAB
* @Mod     2021-10-19 by SHHnog
* @brief   Embedded Controller:  Tutorial ___SysTick and EXTI
*
*********************************************************************************
*/
//*****************************************************************************//
//*************                                                ***************//
//*************   PART C. 7-Segment Display with EXTI Button   ***************//
//*************                                                ***************//
//*****************************************************************************//

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

#include "ecRCC.h"
#include "ecGPIO.h"
#include "ecEXTI.h"
#include "ecSysTick.h"

void setup(void);
void EXTI15_10_IRQHandler(void);

static volatile uint32_t sysTick_Interval = 1;    // Define Interval as 1msec
static volatile uint32_t delay            = 1000; // 1000cycle = 1msec*1000 = 1sec.

static volatile uint32_t Flag_Reset = 0 ;
static volatile uint32_t cnt = 0;

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

  // Infinite Loop ---------------------------------------------------------------
  while(1) {
      SysTick_reset();

      sevensegment_decode(cnt % 10);
      delay_ms(delay);

      if(Flag_Reset==0) {                  // Delay 1000ms = 1sec
        cnt++;
        if(cnt > 9)    cnt = 0;
      }
      if(Flag_Reset==1) {
        TimeDelay = 1;
        cnt = 0;
        Flag_Reset=0;
      }
   }
}



void EXTI15_10_IRQHandler(void) {
   if (is_pending_EXTI(BUTTON_PIN)) {
      clear_pending_EXTI(BUTTON_PIN);
      Flag_Reset = 1;
      delay_ms(0);
  }
}
void setup(void)
{
  //In Simul , CLK Disable must
  RCC_PLL_init();                        // CLK Freq. 84Mhz
  //Interrupt Setting  :: Systick Interrupt
  SysTick_init(sysTick_Interval);                    // Systick Interrupt
  //Interrupt Setting  :: Button Interrupt
  EXTI_init(GPIOC, BUTTON_PIN, FALLING, 0);

  //Initialization Output_Segment
  sevensegment_init(LED_PIN1, LED_PIN2, LED_PIN3, LED_PIN4, LED_PIN5, LED_PIN6, LED_PIN7, LED_PIN8);
  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);
}
```

Figure3.4 Main Code of Part C.

Demo Video for Part C, you can find it from ‘reference’.

Discussion

1) To detect an external signal we can use two different methods: polling and interrupt. What are the advantages and disadvantages of each approach?

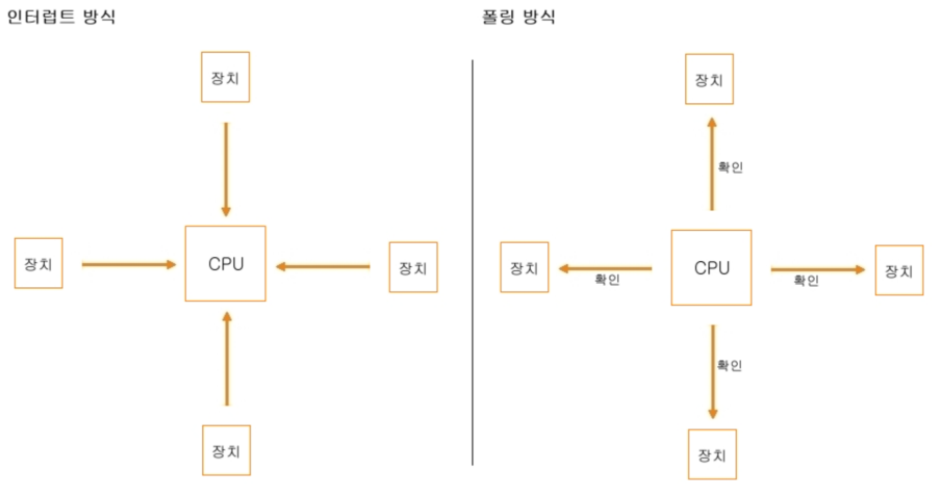

Figure4.1 Comparison between Polling and Interrupt[1]

'Polling' is a method, in which the 'CPU' periodically and continuously checks the input/output devices connected to it.

In contrast, 'Interrupt' is a method in which the 'CPU' gets a signal from an input/output device connected to it so that it can be checked at any time.

	Poiing	**Interrupt**
Advantage	1) The process of implementing the actual code is simple. 2) Since peripheral devices are continuously monitored line by line from CPU, there is an advantage that signals from each device do not overlap in cpu.	1) There is an advantage in that it is possible to accurately know when the interrupt enters, so that more precise control can be made. 2) Since priority can be specified, desired tasks can be selected and selectively executed.

[1] Figure4.1 From here : https://blog.naver.com/makeflood/222421800809

	Poiing	Interrupt
Disadvantage	1) As the number of connected input/output devices increases, the time to occupy the CPU increases, and thus performance decreases. 2) There is a disadvantage in that even if a signal is generated in a connected input/output device, it must wait for the CPU to check it.	1) It is more difficult than a method implemented by polling. 2) Execution of a function that requires a long operation in the 'Interrupt Handler' is limited. Such as ‘printf’.

2) What would happen if the EXTI interrupt handler does not clear the interrupt pending flags? Check with your codeB. Configuration

If pending is not clear, the 'is_pending_EXTI' function is always True. That is, the ‘EXTI15_10_IRQHandler' function is executed indefinitely according to the input frequency, so that the LED becomes toggling very quickly.

However, it seems like that the LED is continuously turned on in our eyes because the toggling speed is too fast. To confirm that the LED state is infinitely toggling, I appropriately modified the codes and observed the results as follows. The light remained on when it was executed with "High-Low," but when it was executed with "High-Low-Low-Low-..." it was confirmed that the light going to slowly faded. The more low is added, the lower the frequency of the voltage applied to the LED, and the larger the period. For this reason, it was thought that as 'Low' was added, the duty ratio of the voltage gradually decreased, and as a result, the light gradually appeared dimly. In other words, it could be observed through experiments that if pending is not cleared, eventually the 'EXTI15_10_IRQHandler' function is executed indefinitely.

```c
void EXTI15_10_IRQHandler(void) {
    if (is_pending_EXTI(BUTTON_PIN)) {
        LED_toggle(GPIOA, LED_PIN);
        //clear_pending_EXTI(BUTTON_PIN);
    }
}
```

Figure4.2 LED Experiment_1

```c
void EXTI15_10_IRQHandler(void) {
    if (is_pending_EXTI(BUTTON_PIN)) {
        //LED_toggle(GPIOA, LED_PIN);
        GPIO_write(GPIOA, LED_PIN, HIGH);
        GPIO_write(GPIOA, LED_PIN, LOW);
        //clear_pending_EXTI(BUTTON_PIN);
    }
}
```

Figure4.3 LED Experiment_2

```c
void EXTI15_10_IRQHandler(void) {
    if (is_pending_EXTI(BUTTON_PIN)) {
        //LED_toggle(GPIOA, LED_PIN);
        GPIO_write(GPIOA, LED_PIN, HIGH);
        GPIO_write(GPIOA, LED_PIN, LOW);
        GPIO_write(GPIOA, LED_PIN, LOW);
        GPIO_write(GPIOA, LED_PIN, LOW);
        GPIO_write(GPIOA, LED_PIN, LOW);
        //clear_pending_EXTI(BUTTON_PIN);
    }
}
```

Figure4.4 LED Experiment_3

```c
void EXTI15_10_IRQHandler(void) {
    if (is_pending_EXTI(BUTTON_PIN)) {
        //LED_toggle(GPIOA, LED_PIN);
        GPIO_write(GPIOA, LED_PIN, HIGH);
        GPIO_write(GPIOA, LED_PIN, LOW);
        GPIO_write(GPIOA, LED_PIN, LOW);
        GPIO_write(GPIOA, LED_PIN, LOW);
        GPIO_write(GPIOA, LED_PIN, LOW);
        GPIO_write(GPIOA, LED_PIN, LOW);
        GPIO_write(GPIOA, LED_PIN, LOW);
        GPIO_write(GPIOA, LED_PIN, LOW);
        GPIO_write(GPIOA, LED_PIN, LOW);
        //clear_pending_EXTI(BUTTON_PIN);
    }
}
```

Figure4.5 LED Experiment_4

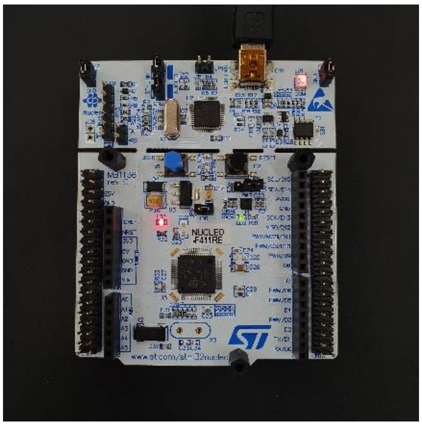

Figure4.6 LED Experiment_1

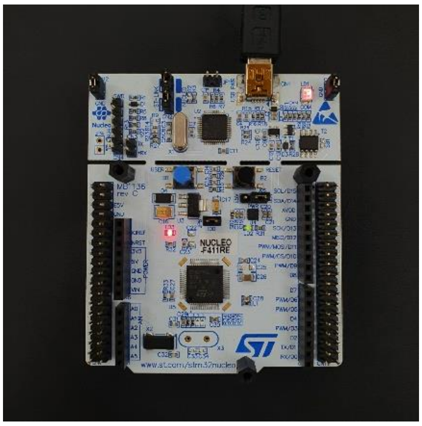

Figure4.7 LED Experiment_2

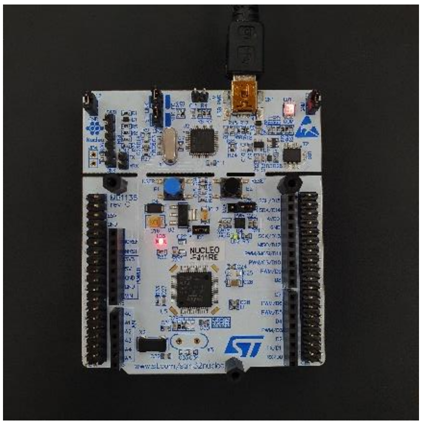

Figure4.8 LED Experiment_3

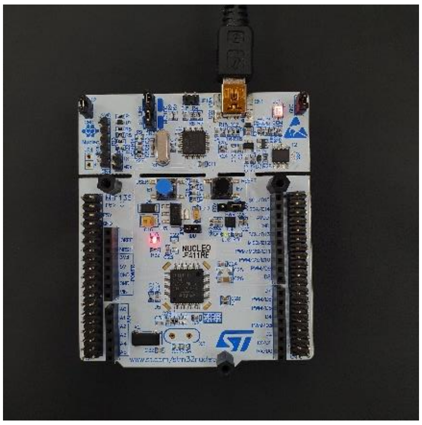

Figure4.9 LED Experiment_4

D. Create User API (Extra Credit)

Below are the examples of functions

Include File	Function	
	Class EC_Ticker // e.g.	EC_Ticker tick
EC_API.h, c	Reset()	// Clear Function
	read_ms()	// Get the Current Value of Timer
	Delay_ms(uint32_t _t)	// Delay Function

➢ Example code:

```
/**
******************************************************************************
* @author   SSSLAB
* @Mod      2021-8-30 by YKKIM
* @brief    Embedded Controller:  LAB Systick&EXTI with API
*
*
******************************************************************************
*/

#include "EC_API.h"

EC_Ticker tick(1);
int count = 0;

// Initialiization
void setup(void)
{
  RCC_PLL_init();
  sevensegment_init();
}

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

  // Inifinite Loop ---------------------------------------------------------
  while(1){
    sevensegment_decode(count);
    tick.Delay_ms(1000);
    count++;
    if (count ==10) count =0;
    tick.reset();
  }
}
```

Figure5.1 Example Code of Part D.

Main Code of Part D

```c
/*
****************************************************************************
* @author   SSSLAB
* @Mod      2021-10-21 by SHHnog
* @brief    Embedded Controller:  Tutorial ___SysTick and EXTI
*
****************************************************************************
*/
//****************************************************************************//
//*************                                              **************//
//*************              PART D. Create User API          **************//
//*************                                              **************//
//****************************************************************************//

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

#include "ecRCC.h"
#include "ecGPIO.h"
#include "ecEXTI.h"
#include "ecSysTick.h"

#include "EC_API.h"

void setup(void);

static volatile uint32_t delay      = 1000; // 1000cycle = 1msec*1000 = 1sec.
static volatile uint32_t cnt        = 0;

// API Section
EC_Ticker tick(1);

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

  // Infinite Loop ----------------------------------------------------------------
  while(1) {
    sevensegment_decode(cnt % 10);
    tick.delay_msec(delay);     // Delay 1000ms = 1sec
    cnt++;

    if(cnt==10) cnt=0;
    tick.reset();
  }
}


void setup(void)
{
  //In Simul , CLK Disable must
  RCC_PLL_init();                          // CLK Freq. 84Mh
  //Interrupt Setting  :: Button Interrupt
  EXTI_init(GPIOC, BUTTON_PIN, FALLING, 0);

  //Initialization Output_Segment
  sevensegment_init(LED_PIN1, LED_PIN2, LED_PIN3, LED_PIN4, LED_PIN5, LED_PIN6, LED_PIN7, LED_PIN8);
  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);
}
```

Figure5.2 Main Code of Part D.

III. Conclusion

Conclusion

In this lab, I controlled the number by 1 second and displayed it on 7-segment led. To control the 7-Segment by using External Interrupt, I created ‘ecSysTick’ and ‘ecEXTI’ HAL drivers.

In the process of using External Interrupt and SysTick Interrupt, I was able to think about the difference between Polling and Interrupt. In addition, in the process of thinking about 'Clock', I was able to think about the overall execution order and structure of the code. Although counting perfectly every one second was not completed for many reasons, it would be good to continue to think about the 'time' in the upcoming 'Lab' and allow precise control. The reason why counting every second was not done perfectly, is specified in 'Trouble Shooting' below.

TroubleShooting

I thought a lot about the initialization point of '7-Segment'. I thought about whether to reset '7-Segment' to zero immediately when the the button is pressed or to zero after proceeding with the remaining delay time when the the button is pressed. After long consideration, I decided to implement both. Depending on the value of the global variable called 'TimeDelay', which is initialized in the function 'EXTI15_10_IRQHandler', initialization style of '7-segment' can be changed. If the code 'TimeDelay=0' of the function 'EXTI15_10_IRQHandler' is not annotated, the LED will be initialized immediately. On the other hand, if the code ‘TimeDelay=0’ of the function ‘EXTI15_10_IRQHandler’ is annotated, the LED is initialized after Delay is completed.

Also, I considered about the reason why we cannot display the number on '7-Segment' exactly every second. The reason why the output was not accurate, the 'HSI CLK' is less accurate. And Second, 'delay' was given a second, but I thought that we couldn't ignore the time when other codes were executed and set before and after the delay was executed. For a similar reason, I thought that it would be difficult to implement as much delay as we wanted because pressing the initialization button would bring in "Button Interrupt" in a situation where "SysTick Interrupt" is in progress. If there are any chances next time, I thought it would be good to think about consider the execution time of code and complete more perfect toggling.

Appendix

A. Demo Video

Part C. See here for detail.

B. Source Code

Source file: ecRCC.h

```
#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
```

Figure6.1 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;
}
```

Figure6.2 Code of ecRCC.c

Source file: ecGPIO.h

```
#ifndef __ECGPIO_H
#define __ECGPIO_H

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

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

//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

#define HIGH 1
#define LOW  0
#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


// The three lines below allow the 'C' language to be recognized in 'C++'.
#ifdef __cplusplus
 extern "C" {
#endif /* __cplusplus */
// The three lines above allow the 'C' language to be recognized in 'C++'.


// LAB2. 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);
// 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);


// LAB4.
void LED_toggle(GPIO_TypeDef *Port, unsigned int pin);


// The three lines below allow the 'C' language to be recognized in 'C++'.
#ifdef __cplusplus
}
#endif /* __cplusplus */
// The three lines above allow the 'C' language to be recognized in 'C++'.

#endif
```

Figure6.3 Code of ecGPIO.h

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) ;
}
// 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]);
}

// LAB4.
void LED_toggle(GPIO_TypeDef *Port, unsigned int pin) {
    Port->ODR ^= (1<<(pin))  ;
}
```

Figure6.4 Code of ecGPIO.c

Source file: ecSysTick.h

```c
#ifndef __EC_SYSTICK_H
#define __EC_SYSTICK_H

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

#define MCU_CLK_PLL 84000000
#define MCU_CLK_HSI 16000000

volatile static uint32_t TimeDelay;

// The three lines below allow the 'C' language to be recognized in 'C++'.
#ifdef __cplusplus
 extern "C" {
#endif /* __cplusplus */
// The three lines above allow the 'C' language to be recognized in 'C++'.

void SysTick_init(uint32_t msec);

uint32_t SysTick_val(void);

void SysTick_reset(void);

void SysTick_enable(void);

void SysTick_disable(void);

void SysTick_Handler(void);

void delay_ms(uint32_t msec);

// The three lines below allow the 'C' language to be recognized in 'C++'.
#ifdef __cplusplus
}
#endif /* __cplusplus */
// The three lines above allow the 'C' language to be recognized in 'C++'.
#endif
```

Figure6.5 Code of ecSysTick.h

Source file: ecSysTick.c

```c
#include "stm32f4xx.h"
#include "stm32f411xe.h"

#include "ecGPIO.h"
#include "ecRCC.h"
#include "ecSysTick.h"
#include "ecEXTI.h"


// LAB4
void SysTick_init(uint32_t msec) {
    //  SysTick Initialiization ------------------------------------------------------
    //  SysTick Control and Status Register
    SysTick->CTRL = 0;              // Disable SysTick IRQ and SysTick Counter

    // Select processor clock
    // 1 = processor clock;  0 = external clock   : we use processor CLK
    SysTick->CTRL |= SysTick_CTRL_CLKSOURCE_Msk;

    // uint32_t MCU_CLK=EC_SYSTEM_CLK
    // SysTick Reload Value Register
    SysTick->LOAD = ((MCU_CLK_PLL/1000)*msec)-1;             // 1ms
    // RELOAD = ( Clock Freq[Hz] * Interval[s] ) - 1
    // PLL = 84MHz = 8400,0000Hz
    // We have to choose What we gonna set the Interval.
    // Here, I chose 1ms for interval. ( 1ms = 1/1000sec )
    // If we want use 1sec for interval, then just put 1sec in Upper Formula.

    // Clear SysTick Current Value
    SysTick->VAL = 0;

    // Enables SysTick exception request
    // 1 = counting down to zero asserts the SysTick exception request
    SysTick->CTRL |= SysTick_CTRL_TICKINT_Msk;

    // Enable SysTick IRQ and SysTick Timer
    SysTick->CTRL |= SysTick_CTRL_ENABLE_Msk;

    NVIC_SetPriority(SysTick_IRQn, 16);      // Set Priority to 1
    SysTick_enable();                        // Enable interrupt in NVIC
}



uint32_t SysTick_val(void){
  return SysTick->VAL;
}


void SysTick_reset(void) {
  SysTick->VAL = 0;
}


void SysTick_enable(void) {
  NVIC_EnableIRQ(SysTick_IRQn);        // Enable interrupt in NVIC
}


void SysTick_disable(void) {
  NVIC_DisableIRQ(SysTick_IRQn);       // Disable interrupt in NVIC
}

void delay_ms(uint32_t msec) {
  TimeDelay = msec;
  while(TimeDelay != 0);
}


void SysTick_Handler(void) {
    if(TimeDelay>0)   TimeDelay--;
}
```

Figure6.6 Code of ecSysTick.c

Source file: ecEXTI.h

```c
#ifndef __EC_EXTI_H
#define __EC_EXTI_H

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

#define RISING  0
#define FALLING 1
#define BOTH    2

#define PA_x    0
#define PB_x    1
#define PC_x    2
#define PD_x    3
#define PE_x    4
#define PH_x    7

// The three lines below allow the 'C' language to be recognized in 'C++'.
#ifdef __cplusplus
 extern "C" {
#endif /* __cplusplus */
// The three lines above allow the 'C' language to be recognized in 'C++'.

 void EXTI_init(GPIO_TypeDef *port, unsigned int pin, unsigned int trig_type, unsigned int priority);

 void EXTI_enable(uint32_t pin); // mask in IMR

 void EXTI_disable(uint32_t pin); // unmask in IMR

 uint32_t is_pending_EXTI(uint32_t pin);
 void clear_pending_EXTI(uint32_t pin);

// The three lines below allow the 'C' language to be recognized in 'C++'.
#ifdef __cplusplus
}
#endif /* __cplusplus */
// The three lines above allow the 'C' language to be recognized in 'C++'.
#endif
```

Figure6.7 Code of ecEXTI.h

Source file: ecEXTI.c

```c
#include "stm32f4xx.h"
#include "stm32f411xe.h"

#include "ecGPIO.h"
#include "ecEXTI.h"


// LAB4
void EXTI_init(GPIO_TypeDef *port, unsigned int pin, unsigned int trig_type, unsigned int priority) {
  // Enable SYSCFG peripheral clock
  RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;

  // Connect the Corresponding External Line to the GPIO
  if(port == GPIOA) {
    SYSCFG->EXTICR[pin/4] &= ~(0xF << 4*(pin%4));
    SYSCFG->EXTICR[pin/4] |= (PA_x << 4*(pin%4));
  }
  if(port == GPIOB) {
    SYSCFG->EXTICR[pin/4] &= ~(0xF << 4*(pin%4));
    SYSCFG->EXTICR[pin/4] |= (PB_x << 4*(pin%4));
  }
  if(port == GPIOC) {
    SYSCFG->EXTICR[pin/4] &= ~(0xF << 4*(pin%4));
    SYSCFG->EXTICR[pin/4] |= (PC_x << 4*(pin%4));
  }
  if(port == GPIOD) {
    SYSCFG->EXTICR[pin/4] &= ~(0xF << 4*(pin%4));
    SYSCFG->EXTICR[pin/4] |= (PD_x << 4*(pin%4));
  }
  if(port == GPIOE) {
    SYSCFG->EXTICR[pin/4] &= ~(0xF << 4*(pin%4));
    SYSCFG->EXTICR[pin/4] |= (PE_x << 4*(pin%4));
  }
  if(port == GPIOH) {
    SYSCFG->EXTICR[pin/4] &= ~(0xF << 4*(pin%4));
    SYSCFG->EXTICR[pin/4] |= (PH_x << 4*(pin%4));
  }

  // Configure the trigger edge :: Falling trigger enable(Button: pull-up)
  if(trig_type == RISING)      EXTI->RTSR |= 1UL << pin;
  if(trig_type == FALLING)     EXTI->FTSR |= 1UL << pin;
  if(trig_type == BOTH) {
     EXTI->RTSR |= 1UL << pin;
     EXTI->FTSR |= 1UL << pin;
  }

  // Configure Interrupt Mask :: Unmask(Enable) EXT interrupt
  EXTI->IMR |= 1UL << BUTTON_PIN;

  // Interrupt IRQn, Priority
  // You Should Match Pin Number and EXTIx number
  if(pin==0) {  NVIC_SetPriority(EXTI0_IRQn, priority);
                NVIC_EnableIRQ(EXTI0_IRQn);                  }
  if(pin==1) {  NVIC_SetPriority(EXTI1_IRQn, priority);
                NVIC_EnableIRQ(EXTI1_IRQn);                  }
  if(pin==2) {  NVIC_SetPriority(EXTI2_IRQn, priority);
                NVIC_EnableIRQ(EXTI2_IRQn);                  }
  if(pin==3) {  NVIC_SetPriority(EXTI3_IRQn, priority);
                NVIC_EnableIRQ(EXTI3_IRQn);                  }
  if(pin==4) {  NVIC_SetPriority(EXTI4_IRQn, priority);
                NVIC_EnableIRQ(EXTI4_IRQn);                  }
  if(pin>=5 && pin<=9)     {  NVIC_SetPriority(EXTI9_5_IRQn, priority);
                              NVIC_EnableIRQ(EXTI9_5_IRQn);                  }
  if(pin>=10 && pin<=15)   {  NVIC_SetPriority(EXTI15_10_IRQn, priority);
                              NVIC_EnableIRQ(EXTI15_10_IRQn);                  }
}
// mask in IMR
void EXTI_enable(uint32_t pin){
  EXTI->IMR |= 1UL << pin;
}


// unmask in IMR
void EXTI_disable(uint32_t pin) {
  EXTI->IMR &= ~(1UL << pin);
}


// Determining whether Pending or not.
uint32_t is_pending_EXTI(uint32_t pin) {
  return ((EXTI->PR & 1UL << pin) == 1UL << pin);
}


// Clear Pending
void clear_pending_EXTI(uint32_t pin) {
  EXTI->PR |= 1<<pin ;
}

```

Figure6.8 Code of ecEXTI.c

Source file: ecAPI.h

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

#include "ecRCC.h"
#include "ecGPIO.h"
#include "ecEXTI.h"
#include "ecSysTick.h"
#include "EC_API.h"

/* System CLOCK is HSI by default */

// Define EC_Ticker
EC_Ticker::EC_Ticker(int reload)
{
        SysTick_init(reload);
}

void EC_Ticker::reset(void)
{
        SysTick_reset();
}

void EC_Ticker::read_ms(void)
{
        SysTick_val();
}

void EC_Ticker::delay_msec(uint32_t dalayValue)
{
        delay_ms(dalayValue);
}
```

Figure6.9 Code of ecAPI.h

Source file: ecAPI.c

```
#ifndef __EC_API_H
#define __EC_API_H
#include "stm32f4xx.h"
#include "stm32f411xe.h"
#include "ecRCC.h"
#include "ecGPIO.h"
#include "ecEXTI.h"
#include "ecSysTick.h"

// The three lines below allow the 'C' language to be recognized in 'C++'.
#ifdef __cplusplus
 extern "C" {
#endif /* __cplusplus */
// The three lines above allow the 'C' language to be recognized in 'C++'.

class EC_Ticker
{
public:
        EC_Ticker(int reload);

        void reset(void);
        void read_ms(void);
        void delay_msec(uint32_t dalayValue);

private:
            int reload;
            unsigned int dalayValue;
};

// The three lines below allow the 'C' language to be recognized in 'C++'.
#ifdef __cplusplus
}
#endif /* __cplusplus */
// The three lines above allow the 'C' language to be recognized in 'C++'.
#endif
```

Figure6.10 Code of ecAPI.c

C. Reference Information

NUCLEO-F401RE

Figure 18. NUCLEO-F401RE

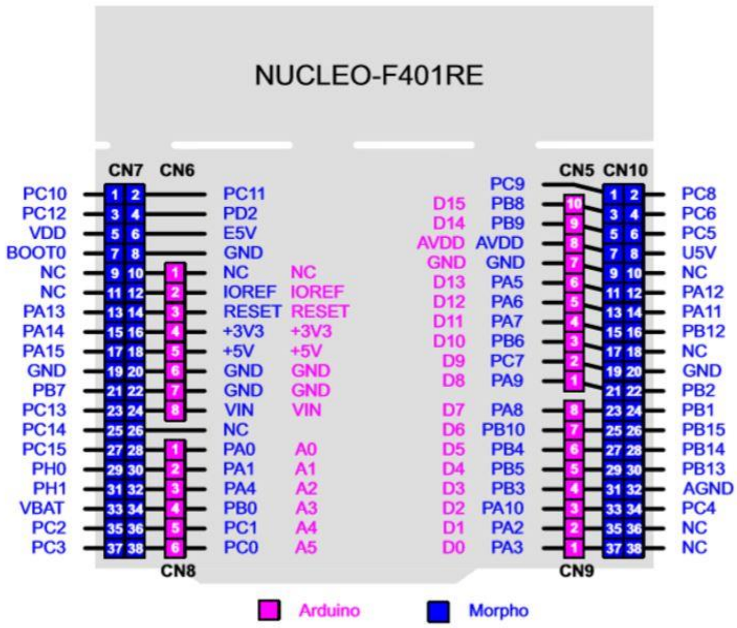

Figure3.7 Reference of NUCLEO Board