

LAB: EXTI SysTick

Date: 2022-10-19

Author/Partner: DongMin Kim / SeongJun Park

Github: https://github.com/DongminKim21800064/EC_dmkim-064/tree/main/lab/EC LAB EXTI SysTick 21800064 %EA%B9%80%EB%8F%99%EB%AF%BC

Demo Video: <https://youtu.be/HSpYlvjLOf0>

Introduction

In this lab, you are required to create two simple programs: toggling multiple LEDs with a push-button input and displaying the number counting from 0 to 9 at 1 second rate on a 7-segment display.

Requirement

Hardware

- MCU
 - NUCLEO-F411RE
- Display
 - 4 LEDs and load resistance
 - 7-segment display(5101ASR)
- Others
 - Array resistor (330 ohm)
 - Breadboard

Software

- Keil uVision, CMSIS, EC_HAL library

Problem 1: LED Toggle with EXTI Button

A program that toggles multiple LEDs with a push-button input using external interrupt.

Create HAL library

Declare and Define the following functions in your library. You must update your header files located in the directory `EC\lib\`.

ecEXTI.h

```
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);
```

Procedure

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

- The project name is “**LAB_EXTI**”.
- Create a new source file named as “**LAB_EXTI.c**”

You MUST write your name on the source file inside the comment section.

2. Include your updated library in `\repos\EC\lib\` to your project.

- **ecGPIO.h, ecGPIO.c**
- **ecRCC.h, ecRCC.c**
- **ecEXTI.h, ecEXTI.c**

1. Connect 4 LEDs externally on a breadboard.

2. Toggle LEDs, turning on one LED at a time by pressing the push button.

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

3. You must use your library function of EXTI.

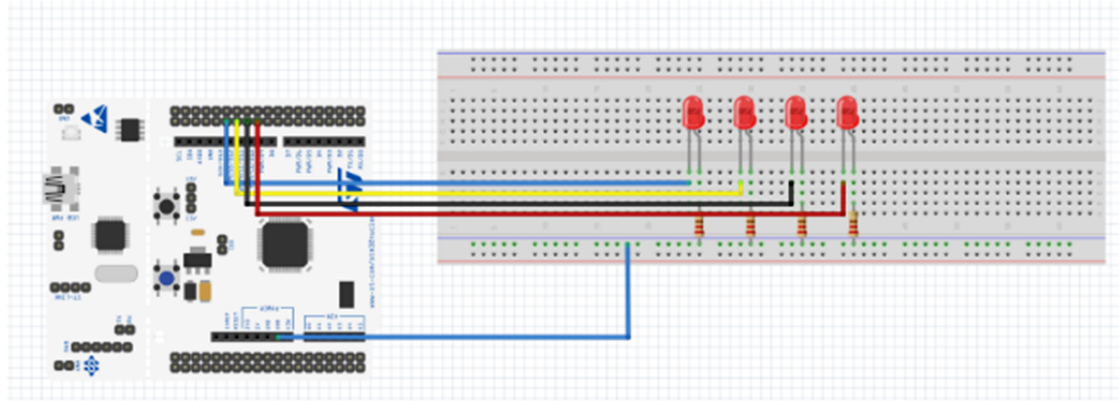
4. Refer to the [sample code](#)

Configuration

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

Circuit Diagram

You need to include the circuit diagram



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?

Interrupts and polling are the ways to handle events.

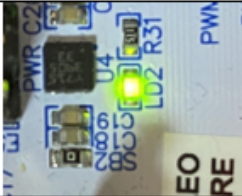




Polling is a method of running an event when it is received by periodically checking whether it has been received or not. The advantage of polling is that the process of implementing it with code is also simple. There is a disadvantage in that the reaction time is slow and takes up a lot of resources.

Interrupt is a method of performing an event directly through the handler when you receive a signal to perform the event. Interrupts have the advantages of accurate timing, short reaction time, and the disadvantages of implementation because they have to be through handlers.

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

Once the LED is turn on, LED looks like never turns off before reset. In reality, because the pending value was not cleared, the function in the interrupt handler was repeatedly running at a very high speed, which seemed to keep the LED turning on state because it could not be recognized by the human eye.

To check this phenomenon, i tried to these process. As you can see, the code was written to repeat the function of turning off the LED while the handler function was running once, and it was confirmed that the brightness of the LED decreased.

	
<pre> 28 void EXTI15_10_IRQHandler(void) { 29 30 if (is_pending_EXTI (BUTTON_PIN)) { 31 GPIO_write(GPIOA,LED_PIN,HIGH); 32 //clear_pending_EXTI(BUTTON_PIN); 33 } 34 } </pre>	<pre> 28 void EXTI15_10_IRQHandler(void) { 29 30 if (is_pending_EXTI (BUTTON_PIN)) { 31 GPIO_write(GPIOA,LED_PIN,HIGH); 32 GPIO_write(GPIOA,LED_PIN,LOW); 33 GPIO_write(GPIOA,LED_PIN,LOW); 34 GPIO_write(GPIOA,LED_PIN,LOW); 35 GPIO_write(GPIOA,LED_PIN,LOW); 36 //clear_pending_EXTI(BUTTON_PIN); 37 } 38 } </pre>
Test1 (100% turn on)	Test2(20% turn on)
	
<pre> 28 void EXTI15_10_IRQHandler(void) { 29 30 if (is_pending_EXTI (BUTTON_PIN)) { 31 GPIO_write(GPIOA,LED_PIN,HIGH); 32 GPIO_write(GPIOA,LED_PIN,LOW); 33 GPIO_write(GPIOA,LED_PIN,LOW); 34 GPIO_write(GPIOA,LED_PIN,LOW); 35 GPIO_write(GPIOA,LED_PIN,LOW); 36 GPIO_write(GPIOA,LED_PIN,LOW); 37 GPIO_write(GPIOA,LED_PIN,LOW); 38 GPIO_write(GPIOA,LED_PIN,LOW); 39 GPIO_write(GPIOA,LED_PIN,LOW); 40 GPIO_write(GPIOA,LED_PIN,LOW); 41 GPIO_write(GPIOA,LED_PIN,LOW); 42 GPIO_write(GPIOA,LED_PIN,LOW); 43 GPIO_write(GPIOA,LED_PIN,LOW); 44 GPIO_write(GPIOA,LED_PIN,LOW); 45 GPIO_write(GPIOA,LED_PIN,LOW); 46 GPIO_write(GPIOA,LED_PIN,LOW); 47 GPIO_write(GPIOA,LED_PIN,LOW); 48 GPIO_write(GPIOA,LED_PIN,LOW); 49 GPIO_write(GPIOA,LED_PIN,LOW); 50 GPIO_write(GPIOA,LED_PIN,LOW); 51 //clear_pending_EXTI(BUTTON_PIN); 52 } 53 } </pre>	
Test3 (5% turn on)	Test4 (1% turn on)

Code

Your code goes here : https://github.com/DongminKim21800064/EC_dmkim-064/tree/main/lab/EC LAB EXTI SysTick 21800064 %EA%B9%80%EB%8F%99%EB%AF%BC

LAB_EXIT.c

```

#include "eCRCC.h"
#include "ecGPIO.h"
#include "ecEXTI.h"
#include "stm32f411xe.h"

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

int main(void) {

    // System CLOCK, GPIO Initialiization -----
    -
    setup();

    while (1){}
}

void EXTI15_10_IRQHandler(void) {
    uint32_t count=0;

```

```

while(1){
    if (is_pending_EXTI (BUTTON_PIN)) {
        if(count==4) count = 0;
        multipleLED(count);
        count++;
        clear_pending_EXTI(BUTTON_PIN); // cleared by writing '1'
    }
}

// Initialiization
void setup(void)
{
    RCC_PLL_init();
    // Initialize multipleLED_init() for Output
    multipleLED_init();
    EXTI_init(GPIOC, BUTTON_PIN, FALL, 0);
}

```

code: EXTI15_10_IRQHandler

The BUTTON_PIN is pin 13 so, it operates in External Line[15:10]. When the BUTTON_PIN interrupt, multipleLED start and cleared by "clear_pending_EXTI(BUTTON_PIN)".

In this lab, I use 4 LED pins and toggled them. In the iteration, 'count' is increased until 4 and reset for turning on 4 LED pins sequentially.

code: multipleLED_init()

"multipleLED_init" is initialized digital in and out. To match the configuration, the button pin is pull-up mode, and the led pins are push-pull, pull-up and medium-speed mode like below.

```

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

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

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

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

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

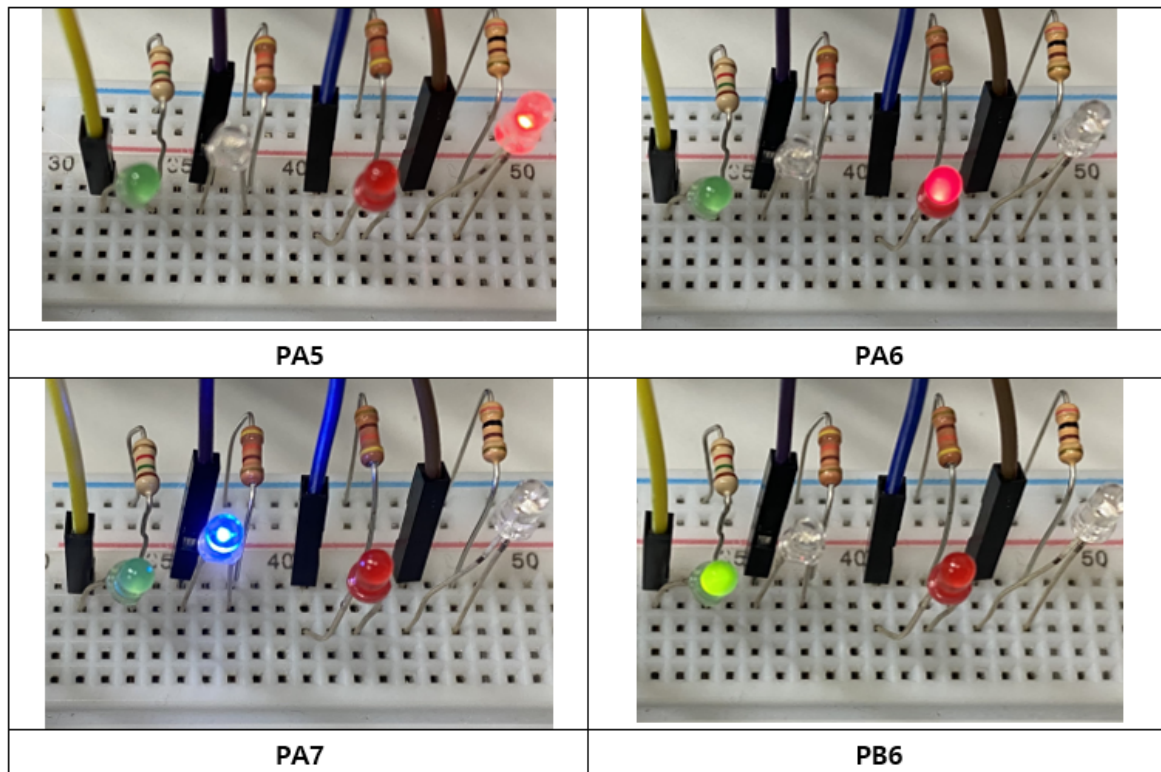
```

code: EXTI_init

Exti_init initialize port, pin number, trig type, and priority.

Results

Experiment images and results



Demo Video: <https://youtu.be/HSpYlvjLOF0>

Problem 2: Counting number on 7-Segment

Display the number 0 to 9 on the 7-segment LED at the rate of 1 sec. After displaying up to 9, then it should display '0' and continue counting.

When the button is pressed, the number should be reset '0' and start counting again.

Create HAL library

Declare and Define the following functions in your library. You must

update your header files located in the directory `EC\lib\`.

ecSysTick.h

```
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);
```

Procedure

1. Create a new project under the directory

`\repos\EC\LAB\LAB_EXTI_SysTick`

- The project name is "**LAB_EXTI_SysTick**".
- Create a new source file named as "**LAB_EXTI_SysTick.c**"

You MUST write your name on the source file inside the comment section.

2. Include your updated library in `\repos\EC\lib\` to your project.

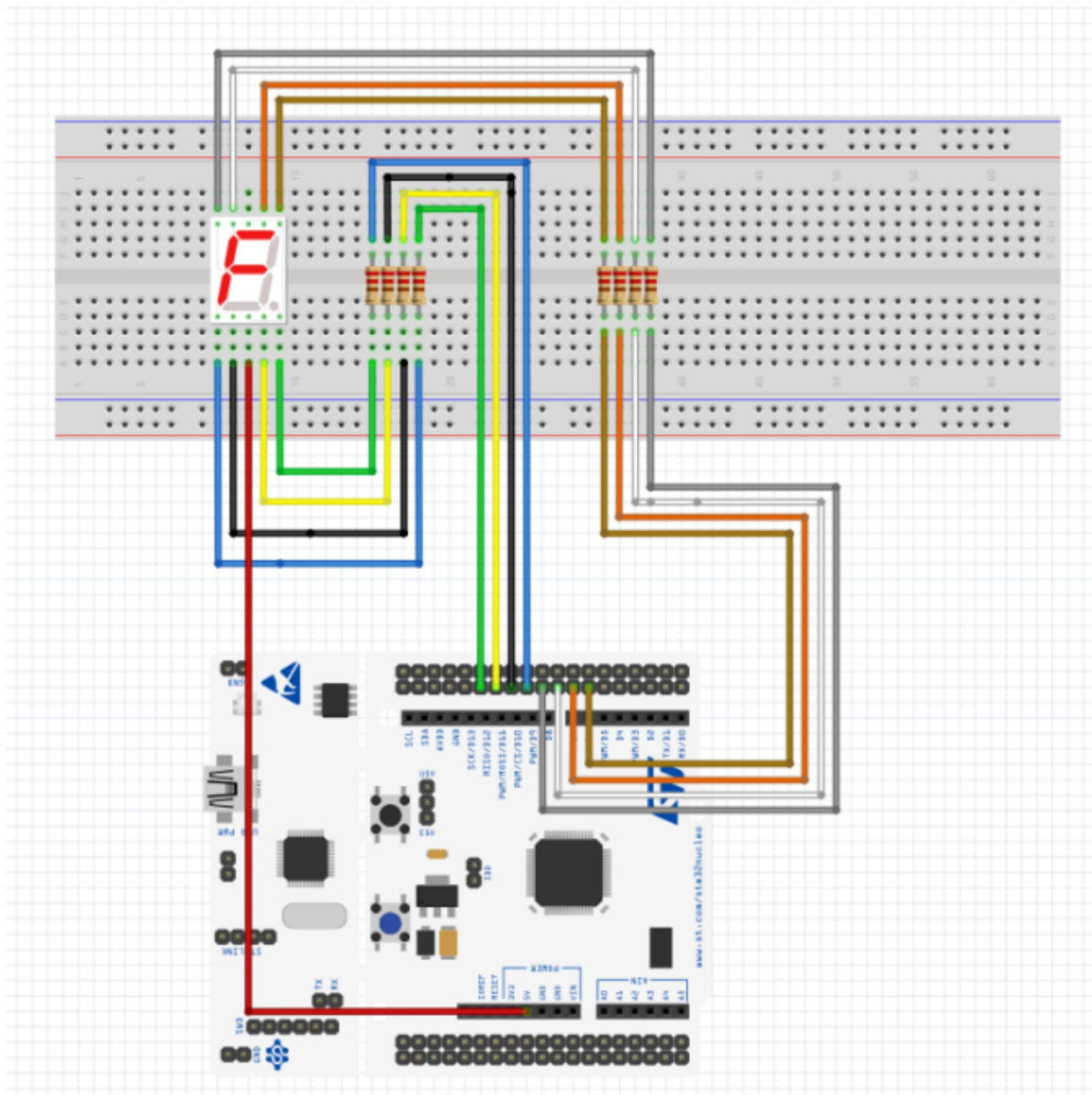
- **ecGPIO.h, ecGPIO.c**
- **ecRCC.h, ecRCC.c**
- **ecEXTI.h, ecEXTI.c**
- **ecSysTick.h, ecSysTick.c**

1. First, check if every number, 0 to 9, can be displayed properly on the 7-segment.
2. Then, create a code to display the number counting from 0 to 9 and repeat at the rate of 1 second.
3. When the button is pressed, it should start from '0' again. Use EXTI for this reset.
4. Refer to the [sample code](#)

Configuration

Digital In for Button (B1)	Digital Out for 7-Segment
Digital In	Digital Out
PC13	PA5, PA6, PA7, PB6, PC7, PA9, PA8, PB10 ('a'~'h', respectively)
PULL-UP	Push-Pull, No Pull-up-Pull-down, Medium Speed

Circuit Diagram



Code

Your code goes here : https://github.com/DongminKim21800064/EC_dmkim-064/tree/main/lab/E_C_LAB_EXTI_SysTick_21800064_%E4%B9%80%E5%8F%99%E5%AF%BC

LAB_EXIT_SysTick.c

```
#include "stm32f411xe.h"
#include "ecGPIO.h"
#include "ecRCC.h"
#include "ecEXTI.h"
#include "ecSysTick.h"

static volatile uint32_t count = 0;
static volatile uint32_t flag = 0;

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

int main(void) {
```



```

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

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

    if(flag==0){
        sevensegment_decode(count);
        delay_ms(1000);
        count++;
        if (count >9) count =0;

    }
    else if(flag==1){
        count = 0;
        sevensegment_decode(count);
        flag = 0;
    }
    SysTick_reset();
}

}

void EXTI15_10_IRQHandler(void) {
    if (is_pending_EXTI (BUTTON_PIN)) {
        flag= 1;
        clear_pending_EXTI(BUTTON_PIN); // cleared by writing '1'
    }
}

// Initialiization
void setup(void)
{
    RCC_PLL_init();
    SysTick_init(1000); //systick period : 1000=1ms , 2000=0.5ms, 500=2ms
    sevensegment_init();
    EXTI_init(GPIOC, BUTTON_PIN, FALL, 0);
}

```

code: main

Flag distinguishes whether button is pressed or not. If the flag is 0, seven-segment starts to display up-counting from 0 to 9. If the flag is 1, seven-segment display 0.

code: EXTI15_10_IRQHandler

The BUTTON_PIN is pin 13 so, it operates in External Line[15:10]. When the BUTTON_PIN interrupt, flag is on and cleared by "clear_pending_EXTI(BUTTON_PIN)".

code: SysTick_init()

"SysTick_init" is initialized and control the systick period. If I insert '1000', then 7-segment display 1 sec per number.

code: sevensegment_init()

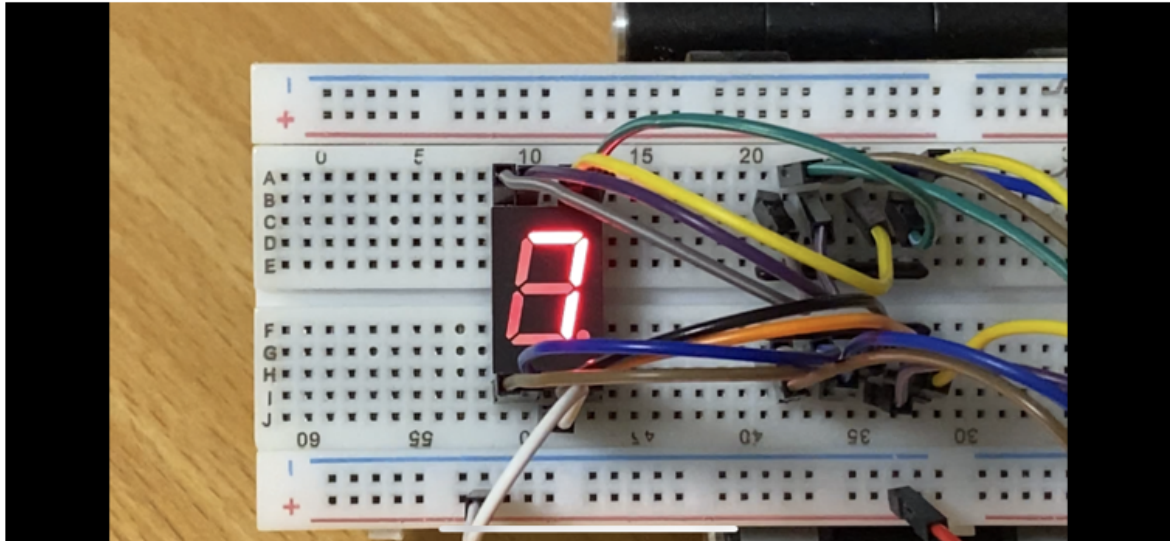
"sevensegment_init" is initialized digital in and out. To match the configuration, the button pin is pull-up mode, and the 7 segment pins are push-pull, NO pull-up-pull-down and medium-speed mode.

code: EXTI_init

Exti_init initialize port, pin number, trig type, and priority.

Results

Experiment images and results



demo video: <https://youtu.be/HSpYlvjLOF0>

Analysis

Through this experiment, I use the SysTick interrupt functions to using external interrupts and periodically execute delay. In addition, I understand the difference between Polling and Interrupt.

Reference

Polling and Interrupt

- https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=sheep_horse&logNo=221450970976

Lecture provided

- <https://github.com/ykkimhgu/EC-student/tree/main/tutorial/tutorial-student>

Troubleshooting

Q1:

SysTick_init() function is not the function which decide delay time. How to set delay time?

A1:

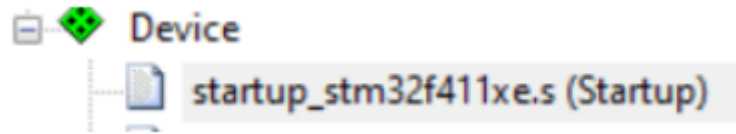
SysTick_init() function is the function which set the time period about SysTick_Handler(). Accordingly, since the time period is 1ms, a delay of 1 second may be implemented by continuously executing the handler function 1000 times.

Q2:

Why does the function "void EXTI15_10_IRQHandler(void)" operating without declaring like "EXTI15_10_IRQHandler()" in the "main" function? For example, "setup" function declared in "main"!

A2:

Because, "EXTI15_10_IRQHandler" is in the device program.



```
DCD      EXTI15_10_IRQHandler          ; External Line[15:10]s
```

Appendix

ecGPIO.c

```
/*-----\
@ Embedded Controller by Young-Keun Kim - Handong Global University
Author      : SSS Lab
Created     : 05-03-2021
Modified    : 10-10-2022 by DongMin Kim
Language/ver : C++ in Keil uVision

Description : Distributed to Students for LAB_GPIO
/-----*/

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

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

void LED_toggle(void){
    GPIOA->ODR ^= 1<< LED_PIN;
}

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

    GPIO_mode(Port, pin, mode);
}
```

```

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

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

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

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

int GPIO_read(GPIO_TypeDef *Port, int pin){

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

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

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

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

```

```

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

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

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

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

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

}

void multipleLED(uint32_t num){
    int count = 0;
    int number[4][4] = {
        {1,0,0,0}, // PA5 LED is turned on and others turned off
        {0,1,0,0}, // PA6 LED is turned on and others turned off
        {0,0,1,0}, // PA7 LED is turned on and others turned off
        {0,0,0,1}, // PB6 LED is turned on and others turned off
    };

    GPIO_write(GPIOA, 5, number[num][0]);
    GPIO_write(GPIOA, 6, number[num][1]);
    GPIO_write(GPIOA, 7, number[num][2]);
    GPIO_write(GPIOB, 6, number[num][3]);

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

void sevensegment_init(void){

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

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

    //Set 7segment_PIN to NO PULL-UP, PULL-DOWN Mode
    GPIO_pupd(GPIOA, 5, NONE); // no pull-up, pull-down
    GPIO_pupd(GPIOA, 6, NONE);

```

```

GPIO_pupd(GPIOA, 7, NONE);
GPIO_pupd(GPIOA, 8, NONE);
GPIO_pupd(GPIOA, 9, NONE);
GPIO_pupd(GPIOB, 6, NONE);
GPIO_pupd(GPIOB,10, NONE);
GPIO_pupd(GPIOC, 7, NONE);

//Set 7segment_PIN to Push-Pull Mode
GPIO_otype(GPIOA, 5, PUSH_PULL); //push-pull
GPIO_otype(GPIOA, 6, PUSH_PULL);
GPIO_otype(GPIOA, 7, PUSH_PULL);
GPIO_otype(GPIOA, 8, PUSH_PULL);
GPIO_otype(GPIOA, 9, PUSH_PULL);
GPIO_otype(GPIOB, 6, PUSH_PULL);
GPIO_otype(GPIOB,10, PUSH_PULL);
GPIO_otype(GPIOC, 7, PUSH_PULL);

//Set 7segment_PIN to mid speed Mode
GPIO_ospeed(GPIOA, 5, MEDIUM_SPEED ); //mid-speed
GPIO_ospeed(GPIOA, 6, MEDIUM_SPEED );
GPIO_ospeed(GPIOA, 7, MEDIUM_SPEED );
GPIO_ospeed(GPIOA, 8, MEDIUM_SPEED );
GPIO_ospeed(GPIOA, 9, MEDIUM_SPEED );
GPIO_ospeed(GPIOB, 6, MEDIUM_SPEED );
GPIO_ospeed(GPIOB,10, MEDIUM_SPEED );
GPIO_ospeed(GPIOC, 7, MEDIUM_SPEED );
}

void sevenssegment_decode(uint8_t num){

    // 7-segments Reversed TruthTable
    int number[10][8] = {

        // A B C D E F G DP
        {0,0,0,0,0,0,1,1}, //zero

        {1,0,0,1,1,1,1,1}, //one
        {0,0,1,0,0,1,0,1}, //two
        {0,0,0,0,1,1,0,1}, //three
        {1,0,0,1,1,0,0,1}, //four
        {0,1,0,0,1,0,0,1}, //five
        {0,1,0,0,0,0,0,1}, //six
        {0,0,0,1,1,1,1,1}, //seven
        {0,0,0,0,0,0,0,1}, //eight
        {0,0,0,0,1,0,0,1}, //nine
    };

    GPIO_write(GPIOA, 8, number[num][0]); // A
    GPIO_write(GPIOB, 10, number[num][1]); // B
    GPIO_write(GPIOA, 7, number[num][2]); // C
    GPIO_write(GPIOA, 6, number[num][3]); // D
    GPIO_write(GPIOA, 5, number[num][4]); // E
    GPIO_write(GPIOA, 9, number[num][5]); // F
    GPIO_write(GPIOC, 7, number[num][6]); // G
    GPIO_write(GPIOB, 6, number[num][7]); // DP

}

void reverse_sevenssegment_decode(uint8_t num){

```

```

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

    {0,0,1,0,0,1,0,1}, //two
    {1,0,0,1,1,1,1,1}, //one
    {0,0,0,0,0,0,1,1}, //zero

};

GPIO_write(GPIOA, 8, number[num][0]); // A
GPIO_write(GPIOB, 10, number[num][1]); // B
GPIO_write(GPIOA, 7, number[num][2]); // C
GPIO_write(GPIOA, 6, number[num][3]); // D
GPIO_write(GPIOA, 5, number[num][4]); // E
GPIO_write(GPIOA, 9, number[num][5]); // F
GPIO_write(GPIOC, 7, number[num][6]); // G
GPIO_write(GPIOB, 6, number[num][7]); // DP

}

```

ecEXTI.h

```

#ifndef __EC_EXTI_H
#define __EC_EXTI_H

#include "stm32f411xe.h"

#define FALL 0
#define RISE 1
#define BOTH 2

#define PAx_PIN 0
#define PBx_PIN 1
#define PCx_PIN 2
#define PDx_PIN 3

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

void EXTI_init(GPIO_TypeDef *Port, int pin, int trig, int priority);
void EXTI_enable(uint32_t pin);
void EXTI_disable(uint32_t pin);
uint32_t is_pending_EXTI(uint32_t pin);

```

```

void clear_pending_EXTI(uint32_t pin);

#ifdef __cplusplus
}
#endif /* __cplusplus */

#endif

```

ecEXTI.c

```

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

void EXTI_init(GPIO_TypeDef *Port, int Pin, int trig_type, int priority){

    // SYSCFG peripheral clock enable
    RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;

    // Connect External Line to the GPIO
    int EXTICR_port;
    if (Port == GPIOA) EXTICR_port = 0;
    else if (Port == GPIOB) EXTICR_port = 1;
    else if (Port == GPIOC) EXTICR_port = 2;
    else if (Port == GPIOD) EXTICR_port = 3;
    else EXTICR_port = 4;

    if(Port == GPIOA)
    {
        SYSCFG -> EXTICR[Pin/4] &= ~(0xF << 4*(Pin%4)); // PC13 ~(1111<<4)
        SYSCFG -> EXTICR[Pin/4] |= PAX_PIN <<4*(Pin%4) ; // 0x 0020 (0000 0000
0010 0000) choose MUX
    }
    else if(Port == GPIOB)
    {
        SYSCFG -> EXTICR[Pin/4] &= ~(0xF << 4*(Pin%4)); // PC13 ~(1111<<4)
        SYSCFG -> EXTICR[Pin/4] |= PBx_PIN <<4*(Pin%4) ; // 0x 0020 (0000 0000
0010 0000) choose MUX
    }
    else if(Port == GPIOC)
    {
        SYSCFG -> EXTICR[Pin/4] &= ~(0xF << 4*(Pin%4)); // PC13 ~(1111<<4)
        SYSCFG -> EXTICR[Pin/4] |= PCx_PIN <<4*(Pin%4) ; // 0x 0020 (0000 0000
0010 0000) choose MUX
    }

    // Configure Trigger edge
    if (trig_type == FALL) EXTI->FTSR |= 1<<Pin; // Falling trigger enable
    else if (trig_type == RISE) EXTI->RTSR |= 1<<Pin; // Rising trigger enable
    else if (trig_type == BOTH) { // Both falling/rising trigger
enable
        EXTI->RTSR |= 1<<Pin;
        EXTI->FTSR |= 1<<Pin;
    }

    // Configure Interrupt Mask (Interrupt enabled)
    EXTI->IMR |= 1<<Pin; // not masked

```



```

// NVIC(IRQ) Setting
int EXTI_IRQn = 0;

if (Pin < 5)    EXTI_IRQn = Pin+6;
else if (Pin < 10) EXTI_IRQn = EXTI9_5_IRQn;
else          EXTI_IRQn = EXTI15_10_IRQn;

NVIC_SetPriority(EXTI_IRQn, priority); // EXTI priority
NVIC_EnableIRQ(EXTI_IRQn); // EXTI IRQ enable
}

void EXTI_enable(uint32_t pin) {
    EXTI->IMR |= 1<<pin;    // not masked (i.e., Interrupt enabled)
}
void EXTI_disable(uint32_t pin) {
    EXTI->IMR |= 0<<pin;    // masked (i.e., Interrupt disabled)
}

uint32_t is_pending_EXTI(uint32_t pin){
    //uint32_t EXTI_PRx = ;    // check EXTI pending
    return ((EXTI->PR & (1UL<<pin)) == (1UL<<pin));
}

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

```

ecSysTick.h

```

/**
*****
* @author   DongMin Kim 21800064
* @Mod      -
*****
*/

#ifndef __EC_SYSTICK_H
#define __EC_SYSTICK_H

#include "stm32f4xx.h"
#include "ecRCC.h"
#include <stdint.h>

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

void SysTick_init(uint32_t Ticks);
void SysTick_Handler(void);
void SysTick_counter();
void delay_ms(uint32_t msec);

```

```

void SysTick_reset(void);
uint32_t SysTick_val(void);

#ifdef __cplusplus
}
#endif /* __cplusplus */

#endif

```

ecSysTick.c

```

/**
*****
* @author   DongMin Kim 21800064
* @Mod      -
*****
*/

#include "ecSysTick.h"

#define MCU_CLK_PLL 84000000
#define MCU_CLK_HSI 16000000

volatile uint32_t mSTicks;

//EC_SYSTEM_CLK

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

    // Select processor clock
    // 1 = processor clock; 0 = external clock
    SysTick->CTRL |= SysTick_CTRL_CLKSOURCE_Msk;

    // uint32_t MCU_CLK=EC_SYSTEM_CLK
    // SysTick Reload Value Register
    SysTick->LOAD = (MCU_CLK_PLL / (Ticks)) - 1; // 1ms,
    for HSI PLL = 84MHz.

    // SysTick Current Value Register
    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
    NVIC_EnableIRQ(SysTick_IRQn); // Enable interrupt in NVIC
}

```

```

void SysTick_Handler(void){
    SysTick_counter();
}

void SysTick_counter(){
    mSTicks++;
}

void delay_ms (uint32_t mesc){
    uint32_t curTicks;

    curTicks = mSTicks;
    while ((mSTicks - curTicks) < mesc);

    mSTicks = 0;
}

//void delay_ms(uint32_t msec){
//    uint32_t now=SysTick_val();
//    if (msec>5000) msec=5000;
//    if (msec<1) msec=1;
//    while ((now - SysTick_val()) < msec);
//}

void SysTick_reset(void)
{
    // SysTick Current Value Register
    SysTick->VAL = 0;
}

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

//void SysTick_counter(){
//    mSTicks++;
//    if(mSTicks%1000 == 0) count++;
//}

```