微處理機 **LAB 6 Interrupt**

Due ： 兩週後 早上 8:00

PART 1. (10%) 實作題

Lab 6.1.1 Timer interrupt – Change LED sparkling rate:

```c
#include <stdio.h>
#include "stm321476xx.h"
#include "helper_functions.h"
#include "7seg.h"
#include "keypad.h"
#include "led_button.h"
#include "timer.h"

// Define pins for Led (default use on-board Led PA5)
#define LED_gpio GPIOA
#define LED_pin 5

// Define pins for button (default use on-board button PC13)
#define BUTTON_gpio GPIOC
#define BUTTON_pin 13

// Define Counter timer
#define COUNTER_timer TIM2

void SysTick_Handler() {
    if(SysTick->CTRL & SysTick_CTRL_COUNTFLAG_Msk){
        // Toggle LED display
        toggle_output(LED_gpio, LED_pin);
    }
}

int main()
{
    // Cause we want to use floating points we need to init FPU
    FPU_init();
    if(init_led(LED_gpio, LED_pin) != 0){
        // Fail to init Led
        return -1;
    }
    if(init_button(BUTTON_gpio, BUTTON_pin) != 0){
        // Fail to init button
        return -1;
    }
    // Configure SysTick clk to 10 Mhz and interrupt every 0.5s
    SystemClock_Config_Interrupt(10, 5000000);

    // button_press_cycle_per_second (How many button press segments in a second)
    int button_press_cycle_per_second = 20;
    // Use to state how many cycles to check per button_press_cycle
    int debounce_cycles = 50;
    // Use to state the threshold when we consider a button press
    int debounce_threshold = debounce_cycles*0.7;
    // Used to implement negative edge trigger 0=not-presses 1=pressed
    int last_button_state=0;


    int blink_frequencies[] = {5000000, 1000000, 3000000};
    int num_blink_frequencies = sizeof(blink_frequencies) / sizeof(blink_frequencies[0]);
    int current_blink_frequency_index = 0;
    int blink_frequency = blink_frequencies[1];
    while(1) {
        for(int a=0;a<button_press_cycle_per_second;a++){
            // Simple Debounce without interrupt
            int pos_cnt=0;
            for(int a=0;a<debounce_cycles;a++){
                // If button press add count
                if(read_gpio(BUTTON_gpio, BUTTON_pin)==0){
                    pos_cnt++;
                }
                delay_without_interrupt(1000/(button_press_cycle_per_second*debounce_cycles));
            }
            // Check if need to change state
            if(pos_cnt>debounce_threshold){
                if(last_button_state==0){
                    // Pressed button - Pos edge
                    current_blink_frequency_index = (current_blink_frequency_index + 1) % num_blink_frequencies;
                    blink_frequency = blink_frequencies[current_blink_frequency_index];
                    SystemClock_Config_Interrupt(10, blink_frequency);
                }
```

```
                else{
                    // Pressed button - Continued pressing
                    // Do nothing
                }
                last_button_state = 1;
            }
            else{
                if(last_button_state==0){
                    // Released button - Not pressing
                    // Do nothing
                }
                else{
                    // Released button - Neg edge
                     // Do nothing
                }
                last_button_state = 0;
            }
        }
    }
    while(1);
    return 0;
}
```

PART 2. (40%) 實作題

Lab6.2: Keypad external interrupt

```c
#include <stdio.h>
#include "stm32l476xx.h"
#include "helper_functions.h"
#include "7seg.h"
#include "keypad.h"
#include "led_button.h"
#include "timer.h"

// Define Counter timer
#define COUNTER_timer TIM2

// Define pins for 7seg
#define SEG_gpio GPIOC
#define DIN_pin 3
#define CS_pin 2
#define CLK_pin 1

// Define pins for keypad
#define COL_gpio GPIOA
#define COL_pin 6       // 6 7 8 9
#define ROW_gpio GPIOB
#define ROW_pin 3       // 3 4 5 6

int now_col = 3;
int keyCnt = 0, keyValue = -1;

void SysTick_Handler() {
    if(SysTick->CTRL & SysTick_CTRL_COUNTFLAG_Msk){
        reset_push(COL_gpio, now_col+COL_pin);
        now_col = (now_col+1)%4;
        set_push(COL_gpio, now_col+COL_pin);
    }
}

void EXTIKeyPad_Handler(int r){
    int nowKey = keypad[r][(now_col+3)%4];
    if(nowKey == keyValue){
        keyCnt++;
    }else{
        keyCnt = 0;
    }
```

```
    }
    keyValue = nowKey;
    if(keyCnt >=5){
        keyCnt = 5;
        display_number(SEG_gpio, DIN_pin, CS_pin, CLK_pin, keyValue, 2);
    }
}

void EXTI_Setup(){
    // Enable SYSCFG clock
    RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;
    // Select output bits
    SYSCFG-> EXTICR[0] &= ~SYSCFG_EXTICR1_EXTI3_Msk;
    SYSCFG-> EXTICR[0] |= (1 << SYSCFG_EXTICR1_EXTI3_Pos);
    SYSCFG-> EXTICR[1] &= ~SYSCFG_EXTICR2_EXTI4_Msk;
    SYSCFG-> EXTICR[1] |= (1 << SYSCFG_EXTICR2_EXTI4_Pos);
    SYSCFG-> EXTICR[1] &= ~SYSCFG_EXTICR2_EXTI5_Msk;
    SYSCFG-> EXTICR[1] |= (1 << SYSCFG_EXTICR2_EXTI5_Pos);
    SYSCFG-> EXTICR[1] &= ~SYSCFG_EXTICR2_EXTI6_Msk;
    SYSCFG-> EXTICR[1] |= (1 << SYSCFG_EXTICR2_EXTI6_Pos);

    // Enable interrupt
    EXTI->IMR1 |= EXTI_IMR1_IM3;
    EXTI->IMR1 |= EXTI_IMR1_IM4;
    EXTI->IMR1 |= EXTI_IMR1_IM5;
    EXTI->IMR1 |= EXTI_IMR1_IM6;

    // Enable Falling Edge
    EXTI->FTSR1 |= EXTI_FTSR1_FT3;
    EXTI->FTSR1 |= EXTI_FTSR1_FT4;
    EXTI->FTSR1 |= EXTI_FTSR1_FT5;
    EXTI->FTSR1 |= EXTI_FTSR1_FT6;

    // Enable NVIC
    NVIC_EnableIRQ(EXTI3_IRQn);
    NVIC_EnableIRQ(EXTI4_IRQn);
    NVIC_EnableIRQ(EXTI9_5_IRQn);
}

void EXTI3_IRQHandler(){
    if(EXTI->PR1 & EXTI_PR1_PIF3_Msk){
        EXTIKeyPad_Handler(0);
        EXTI->PR1 |= EXTI_PR1_PIF3_Msk;
    }
}

void EXTI4_IRQHandler(){
    if(EXTI->PR1 & EXTI_PR1_PIF4_Msk){
        EXTIKeyPad_Handler(1);
        EXTI->PR1 |= EXTI_PR1_PIF4_Msk;
    }
}

void EXTI9_5_IRQHandler(){
    if(EXTI->PR1 & EXTI_PR1_PIF5_Msk){
        EXTIKeyPad_Handler(2);
        EXTI->PR1 |= EXTI_PR1_PIF5_Msk;
    }
    if(EXTI->PR1 & EXTI_PR1_PIF6_Msk){
        EXTIKeyPad_Handler(3);
        EXTI->PR1 |= EXTI_PR1_PIF6_Msk;
    }
}

int main()
{
    // Cause we want to use floating points we need to init FPU
    FPU_init();
    if(init_7seg(SEG_gpio, DIN_pin, CS_pin, CLK_pin) != 0){
        // Fail to init 7seg
        return -1;
    }
```

```
    }
    if(init_keypad(ROW_gpio, COL_gpio, ROW_pin, COL_pin) != 0){
        // Fail to init keypad
        return -1;
    }

    // Set Decode Mode to non-decode mode
    send_7seg(SEG_gpio, DIN_pin, CS_pin, CLK_pin, SEG_ADDRESS_DECODE_MODE,0xFF);
     // Set Scan Limit to digit 0 only
    send_7seg(SEG_gpio, DIN_pin, CS_pin, CLK_pin, SEG_ADDRESS_SCAN_LIMIT,0xFF);
     // Wakeup 7seg
    send_7seg(SEG_gpio, DIN_pin, CS_pin, CLK_pin, SEG_ADDRESS_SHUTDOWN,0x01);

    // Configure SysTick clk to 10 Mhz and interrupt every 0.1s
    SystemClock_Config_Interrupt(10, 10000);
    // Init Interrupts
    EXTI_Setup();

    while(1);
    return 0;
}
```

PART 3. (50%) 實作題

Lab6.3: 製作簡單鬧鐘

```
#include <stdio.h>
#include "stm32l476xx.h"
#include "helper_functions.h"
#include "7seg.h"
#include "keypad.h"
#include "led_button.h"
#include "timer.h"

// Define Counter timer
#define COUNTER_timer TIM2

// Define pins for 7seg
#define SEG_gpio GPIOC
#define DIN_pin 3
#define CS_pin 2
#define CLK_pin 1

// Define pins for keypad
#define COL_gpio GPIOA
#define COL_pin 6 // 6 7 8 9
#define ROW_gpio GPIOB
#define ROW_pin 3 // 3 4 5 6

// Define pins for button (default use on-board button PC13)
#define BUTTON_gpio GPIOC
#define BUTTON_pin 13

int num = 0, tmp = 0;

int alarm_state = 0;
// state 0: wait for input
// state 1: alarm

// button_press_cycle_per_second (How many button press segments in a second)
#define button_press_cycle_per_second  20
// Use to state how many cycles to check per button_press_cycle
#define debounce_cycles  50
// Use to state the threshold when we consider a button press
#define debounce_threshold  (debounce_cycles*0.7)
// Used to implement negative edge trigger 0=not-presses 1=pressed
int last_button_state = 0;

//
void NVIC_Configuration() {
```

```c
void NVIC_Configuration() {
    NVIC_SetPriority(EXTI15_10_IRQn, 0); // 最高優先順序
    NVIC_SetPriority(SysTick_IRQn, 1);  // 次高優先順序

    // Enable SYSCFG clock
    RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;

    // Select output bits
    SYSCFG-> EXTICR[3] &= ~SYSCFG_EXTICR4_EXTI13_Msk;
    SYSCFG-> EXTICR[3] |= (2 << SYSCFG_EXTICR4_EXTI13_Pos);

    // Enable interrupt
    EXTI->IMR1 |= EXTI_IMR1_IM13;

    // Enable Falling Edge
    EXTI->FTSR1 |= EXTI_FTSR1_FT13;

    // Enable NVIC
    NVIC_EnableIRQ(EXTI15_10_IRQn);
}


// SysTick timer中斷處理函數
void SysTick_Handler(void) {
  if (SysTick->CTRL & SysTick_CTRL_COUNTFLAG_Msk && alarm_state == 1) {
    if(num > 0)
        num--;
    if (num <= 0)
    {
        timer_init(TIM2,383,20);
        timer_start(TIM2);
        display_number(SEG_gpio, DIN_pin, CS_pin, CLK_pin, 0, 1);
    }
  }
}

// User button 外部中斷處理函數
void EXTI15_10_IRQHandler(void){
    if(EXTI->PR1 & EXTI_PR1_PIF13_Msk && alarm_state == 1){
        for(int a=0;a<button_press_cycle_per_second;a++){
            // Simple Debounce without interrupt
            int pos_cnt=0;
            for(int a=0;a<debounce_cycles;a++){
                // If button press add count
                if(read_gpio(BUTTON_gpio, BUTTON_pin)==0){
                    pos_cnt++;
                }
                delay_without_interrupt(1000/(button_press_cycle_per_second*debounce_cycles));
            }
            // Check if need to change state
            if(pos_cnt>debounce_threshold){
                // Pressed
                num = 0;
                alarm_state = 0;
                timer_stop(TIM2);
                EXTI->PR1 |= EXTI_PR1_PIF3_Msk;
                last_button_state = 1;
                tmp = 0;
                num = 0;
            }
            else{
                if(last_button_state==0){
                    // Released button - Not pressing
                    // Do nothing
                }
                else{
                    // Released button - Neg edge
                    // Do nothing
                }
                last_button_state = 0;
            }
        }
    }
}

int keypad_space[16][2] = {{0, 0}, {0, 1}, {0, 2}, {0, 3},
                           {1, 0}, {1, 1}, {1, 2}, {1, 3},
                           {2, 0}, {2, 1}, {2, 2}, {2, 3},
                           {3, 0}, {3, 1}, {3, 2}, {3, 3}};

int get_keyboard_input()
{
    for (int i = 0; i < 16; i++)
    {
        int a = keypad_space[i][0];
```

```
        int a = keypad_space[i][0];
        int b = keypad_space[i][1];
        if (check_keypad_input_one(ROW_gpio, COL_gpio, ROW_pin, COL_pin, a, b))
        {
            while (check_keypad_input_one(ROW_gpio, COL_gpio, ROW_pin, COL_pin, a, b) != 0)
                continue;
            return keypad[a][b];
        }
    }
    return -1;
}

int main()
{
    // Cause we want to use floating points we need to init FPU
    FPU_init();

    if (init_7seg(SEG_gpio, DIN_pin, CS_pin, CLK_pin) != 0)
    {
        // Fail to init 7seg
        return -1;
    }
    if (init_keypad(ROW_gpio, COL_gpio, ROW_pin, COL_pin) != 0)
    {
        // Fail to init keypad
        return -1;
    }
    if(init_button(BUTTON_gpio, BUTTON_pin) != 0){
        // Fail to init button
        return -1;
    }
    // Set Decode Mode to non-decode mode
    send_7seg(SEG_gpio, DIN_pin, CS_pin, CLK_pin, SEG_ADDRESS_DECODE_MODE, 0xFF);
    // Set Scan Limit to digit 0 only
    send_7seg(SEG_gpio, DIN_pin, CS_pin, CLK_pin, SEG_ADDRESS_SCAN_LIMIT, 0xFF);
    // Wakeup 7seg
    send_7seg(SEG_gpio, DIN_pin, CS_pin, CLK_pin, SEG_ADDRESS_SHUTDOWN, 0x01);

    // begin with 0
    display_number(SEG_gpio, DIN_pin, CS_pin, CLK_pin, 0, 1);

    GPIO_init_AF();
    timer_enable(TIM2);
    PWM_channel_init();

    while (1)
    {
        if (alarm_state == 0)
        {
            tmp = get_keyboard_input();
            if (tmp != -1)
                num = num * 10 + tmp;
            display_number(SEG_gpio, DIN_pin, CS_pin, CLK_pin, num, num_digits(num));
            for(int a=0;a<button_press_cycle_per_second;a++){
                // Simple Debounce without interrupt
                int pos_cnt=0;
                for(int a=0;a<debounce_cycles;a++){
                    // If button press add count
                    if(read_gpio(BUTTON_gpio, BUTTON_pin)==0){
                        pos_cnt++;
                    }
                    delay_without_interrupt(1000/(button_press_cycle_per_second*debounce_cycles));
                }
                // Check if need to change state
                if(pos_cnt>debounce_threshold){
                    if(last_button_state==0 && num!=0){
                        // Pressed button - Pos edge
                        alarm_state = 1;
                        // 設定中斷優先順序
                        NVIC_Configuration();
                        // Configure SysTick clk to 10 Mhz and interrupt every 1s
                        SystemClock_Config_Interrupt(10, 10000000);
                    }
                    else{
                        // Pressed button - Continued pressing
                        // Do nothing
                    }
                    last_button_state = 1;
                }
                else{
                    if(last_button_state==0){
                        // Released button - Not pressing
                        // Do nothing
                    }
                    else{
```

```
                    }
                    else{
                        // Released button - Neg edge
                        // Do nothing
                    }
                    last_button_state = 0;
                }
            }
        }else if(alarm_state == 1){
            display_number(SEG_gpio, DIN_pin, CS_pin, CLK_pin, num, num_digits(num));
        }
    }
    while(1);
    return 0;
}
```