微處理機系統實習　　Lab6

班級：資訊三丁 學號：D1009212 姓名：邱柏宇

一、【實驗目的】：

What was your design? What were the concepts you have used for your design?

1. 重構 draw bitmap

上一個 lab 中有提到重構 LCD.h 變成 NewLCD.h，但發現到在這次 lab 的場景中還是有些地方不好用，因此加上了一些新功能，例如 draw_bitmap_in_buffer ，可以將 bitmap 的 16 進制進行轉換，讓其可以不用像原生只能印在 y 為 8 整除的情況，並且可以自己設定寬高。

2. 小綠人

關於這次小綠人的作法，是先手繪 19 張圖片，每張 16x16 pixel

那為甚麼我要把它們圖片併在一起勒，因為如果使用原生的 draw_LCD 需要 128x8 uint8_t 的 bitmap，如果使用 draw_bitmap 也是需要花費很多空間才能放下，為了節省記憶體資源，我打算只在陣列中放入 16x16 pixel 的大小，並且在繪製時等比例放大。

19x16=304，所以我只需要將寬 304 高 16 pixel 轉換成一張很大張的 bitmap 就可以，這裡寫了一個 python 腳本去轉換，就可以做到 304x16/2 uint8_t 大小的 bitmap 就存下了 19 張圖片，比原生 draw_LCD 省了大概 8 倍。

```python
if __name__ == "__main__":
    # 輸入檔案名稱, 寬, 高
    if len(sys.argv) == 4:
        filename = sys.argv[1]
        width = int(sys.argv[2])
        height = int(sys.argv[3])
    else:
        print("請輸入檔案名稱和長寬")
        sys.exit()
    bmp_img = Image.open(filename)
    bmp_bin = list(bmp_img.getdata())
    # 15 轉成 0，其他轉成 1
    bmp_bin = [0 if x == 15 else 1 for x in bmp_bin]

    # 壓縮成 16 進制
    bmp_hex = []
    for i in range(0, width * int(height / 8)):
        temp_bin = ''
        temp_hex = 0
        for j in range(0, 8):
            temp_bin += str(bmp_bin[i + j  * width])
        # 順序反轉
        temp_bin = temp_bin[::-1]
        temp_hex = f'{hex(int(str(temp_bin), 2))}'
        bmp_hex.append(temp_hex)

    # 標準化格式
    for i in range(len(bmp_hex)):
```

3. 中斷跟繪製

首先用 frame 去記錄現在要印出哪一個動作，然後在陣列進行偏移去取 16x8 出來等比例放大後繪製出來，也用到上次所提到的動態更新，去避免不必要的 lcdWriteData，timer0_sec % 14 是因為紅燈和綠燈加起來為 14 秒循環，所以透過 mod 就可以知道現在該綠色還是紅色。

```c
void TMR1_IRQHandler(void) {
    CloseSevenSegment();
    clear_lcd_buffer();
    if (timer0_sec % 14 < 9) {
        draw_green_man(frame);
        frame++;
        frame %= 18;
        control_rgb(1, 0, 1);
        ShowSevenSegment(0, 9 - timer0_sec % 14);
    } else if (timer0_sec % 14 >= 9 && timer0_sec % 14 < 14) {
        draw_green_man(18);
        control_rgb(1, 1, 0);
        ShowSevenSegment(3, 14 - timer0_sec % 14);
    }
    show_lcd_buffer();
    timer0_100ms++;
    if (timer0_100ms % 10 == 0) {
        timer0_sec++;
        timer0_sec %= 60;
    }
    TIMER_ClearIntFlag(TIMER1);
}
```

二、【遭遇的問題】：

What problems you faced during design and implementation?

要放入很多 bitmap 是會爆記憶體的

大約兩個 128x64x2 大小的 uint8_t 就會爆掉

三、【解決方法】：

How did you solve the problems?

用上述提到的在繪製時放大就可以不用存這麼大張。

四、【未能解決的問題】：

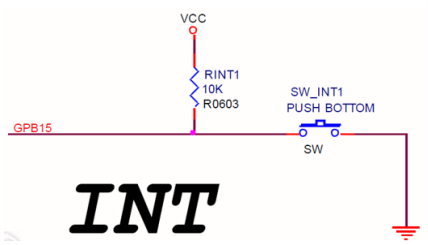
Was there any problem that you were unable to solve? Why was it unsolvable?

蜂鳴器現在是用 delay，也許可以用成 Timer。

五、【投影片的問題】：

1. GPIO_EnableEINT1(PB, 15, GPIO_INT_RISING);的 GPIO_INT_RISING 是甚麼意思?
後面 5 個各代表甚麼?

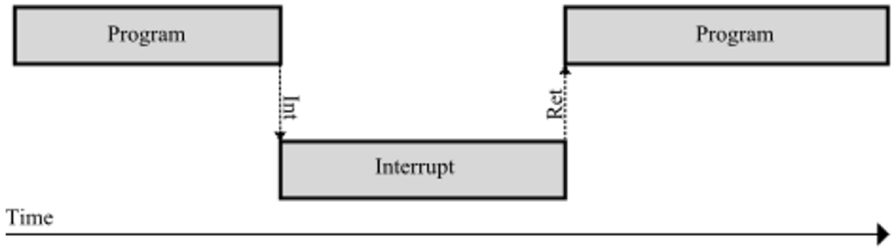
GPIO_EnableEINT1 用來設定中斷的觸發是甚麼模式，並且中斷按鈕 GPB15 在設計上是有一個上拉電阻的，所以在這次案例中使用 GPIO_INT_RISING 其實就是按下去一瞬間中斷。



RISING 是 LOW 到 HIGH、FALLING 是 HIGH 到 LOW、BOTH_EDGE 是 RISING 和 FALLING 都會觸發
HIGH 是高、LOW 是低

也可以 trace code 後看到官方的解釋。

```
/*---------------------------------------------------------------------------------------------------*/
#define GPIO_INT_RISING         0x00010000UL /*!< Interrupt enable by Input Rising Edge */
#define GPIO_INT_FALLING        0x00000001UL /*!< Interrupt enable by Input Falling Edge */
#define GPIO_INT_BOTH_EDGE      0x00010001UL /*!< Interrupt enable by both Rising Edge and Falling Edge */
#define GPIO_INT_HIGH           0x01010000UL /*!< Interrupt enable by Level-High */
#define GPIO_INT_LOW            0x01000001UL /*!< Interrupt enable by Level-Level */
```



2. 甚麼是 TMR0_OPERATING_MODE?後面 4 個模式各代表甚麼?

總共有四種，分別是 ONESHOT, PERIODIC/TOGGLE, COTINUOUS

```
#define TIMER_ONESHOT_MODE              (0UL << TIMER_TCSR_MODE_Pos)        /*!< Timer working in one-shot mode */
#define TIMER_PERIODIC_MODE             (1UL << TIMER_TCSR_MODE_Pos)        /*!< Timer working in periodic mode */
#define TIMER_TOGGLE_MODE               (2UL << TIMER_TCSR_MODE_Pos)        /*!< Timer working in toggle-output mode */
#define TIMER_CONTINUOUS_MODE           (3UL << TIMER_TCSR_MODE_Pos)        /*!< Timer working in continuous counting mode */
```

TIMER 就像是計數器，而這四種模式就是在計數上碰到中斷時會發生甚麼事。
ONESHOT 就會停在那邊
PERIODIC/TOGGLE 會返回到一開始
COTINUOUS 則是會持續到計數器溢位然後回到一開始
這裡提供老師的圖補充