Lab4 STM32 GPIO System

0516076 施威綸

1. Lab objectives 實驗目的

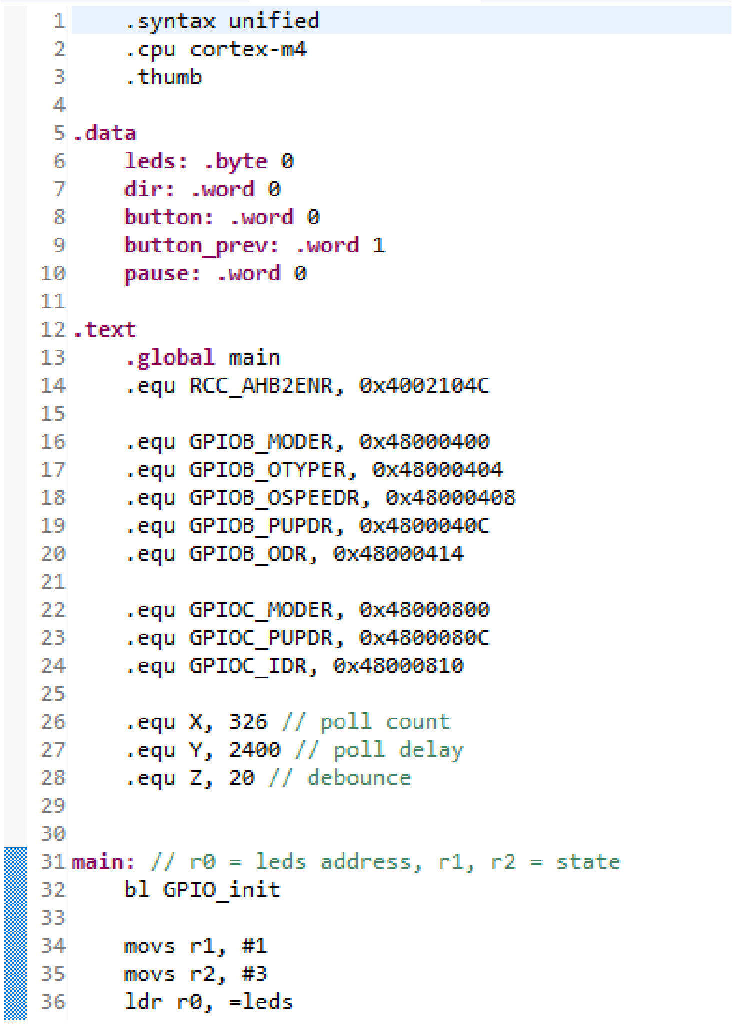
- Understand the principle of using STM32 basic I/O port
- Design a simple LED marquee program
- Understand the use of buttons and DIP switches

2. Steps 實驗步驟

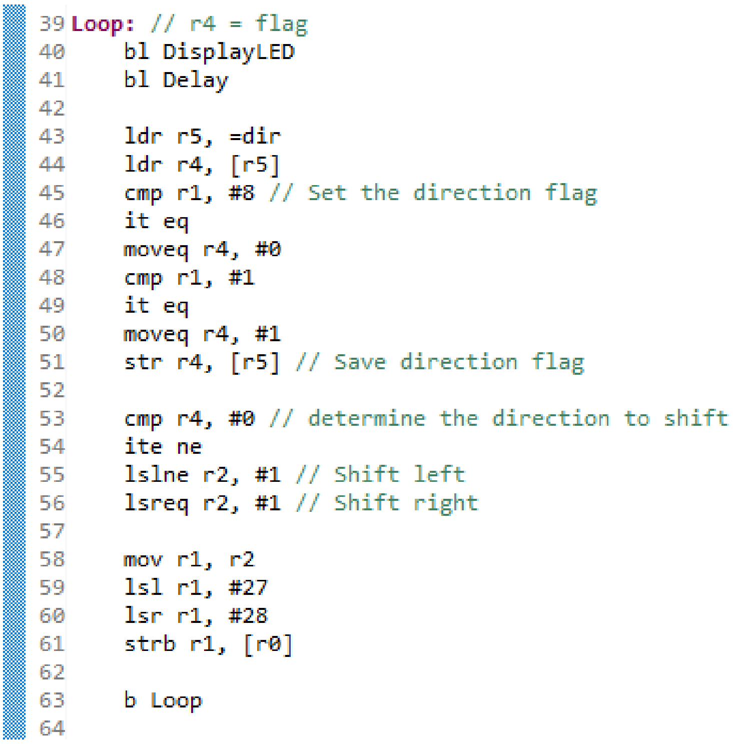
2.1. LED pattern displayer

Refer to the tutorial on the lecture slide for finishing the initialization of GPIO output and constructing 4 active low LED circuits. (Turn off the LED when GPIO output “1”, and turn on when GPIO output “0”)

Connect the LEDs to PB3, PB4, PB5, PB6 on the board.

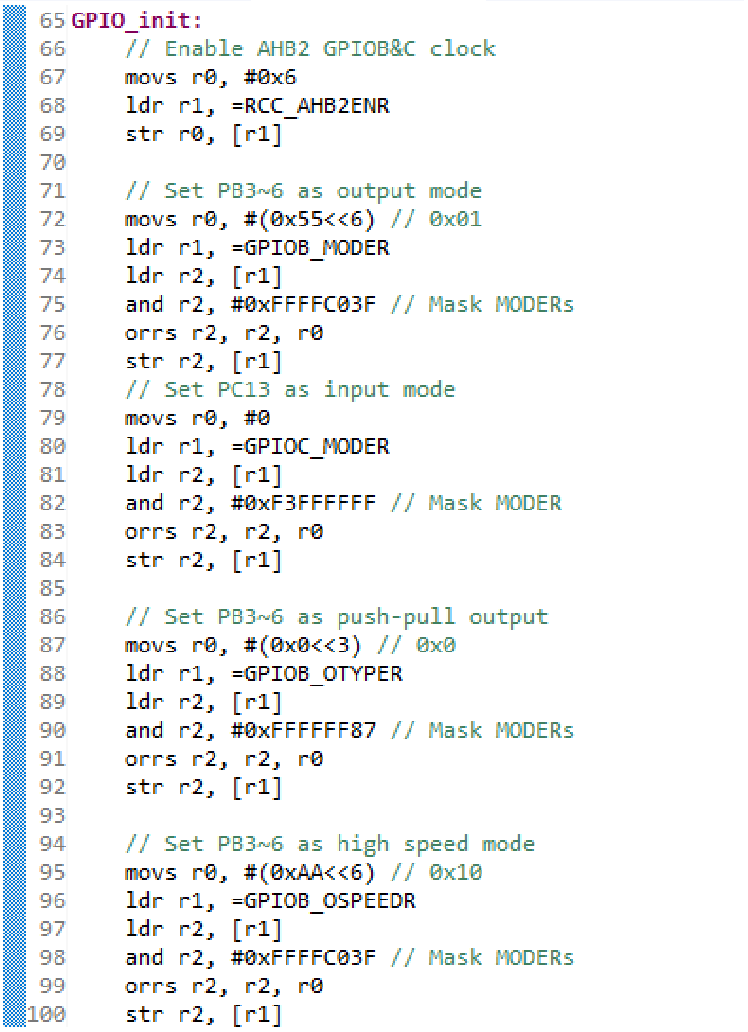


```
    .syntax unified
    .cpu cortex-m4
    .thumb

.data
    leds: .byte 0
    dir: .word 0
    button: .word 0
    button_prev: .word 1
    pause: .word 0

.text
    .global main
    .equ RCC_AHB2ENR, 0x4002104C

    .equ GPIOB_MODER, 0x48000400
    .equ GPIOB_OTYPER, 0x48000404
    .equ GPIOB_OSPEEDR, 0x48000408
    .equ GPIOB_PUPDR, 0x4800040C
    .equ GPIOB_ODR, 0x48000414

    .equ GPIOC_MODER, 0x48000800
    .equ GPIOC_PUPDR, 0x4800080C
    .equ GPIOC_IDR, 0x48000810

    .equ X, 326 // poll count
    .equ Y, 2400 // poll delay
    .equ Z, 20 // debounce


main: // r0 = leds address, r1, r2 = state
    bl GPIO_init

    movs r1, #1
    movs r2, #3
    ldr r0, =leds
```

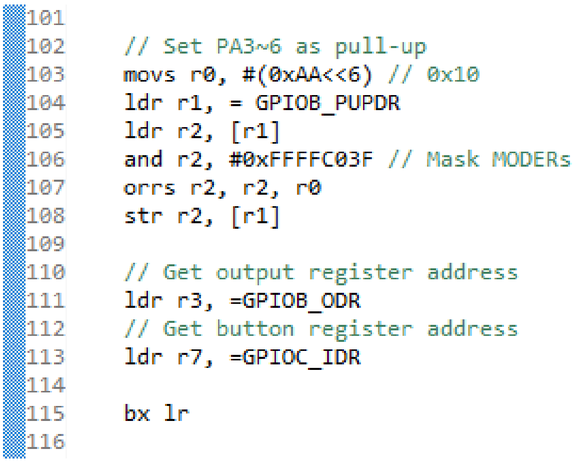


```
Loop: // r4 = flag
    bl DisplayLED
    bl Delay

    ldr r5, =dir
    ldr r4, [r5]
    cmp r1, #8 // Set the direction flag
    it eq
    moveq r4, #0
    cmp r1, #1
    it eq
    moveq r4, #1
    str r4, [r5] // Save direction flag

    cmp r4, #0 // determine the direction to shift
    ite ne
    lslne r2, #1 // Shift left
    lsreq r2, #1 // Shift right

    mov r1, r2
    lsl r1, #27
    lsr r1, #28
    strb r1, [r0]

    b Loop

```

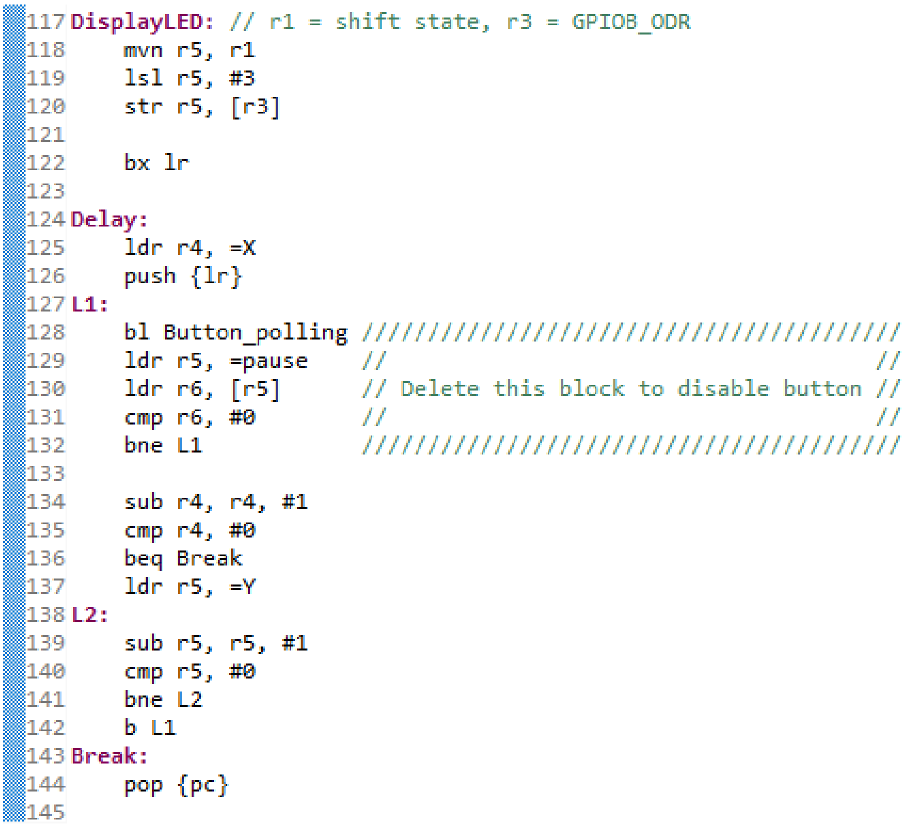
Initial the bus clock and setup the GPIO registers and pins



```
GPIO_init:
    // Enable AHB2 GPIOB&C clock
    movs r0, #0x6
    ldr r1, =RCC_AHB2ENR
    str r0, [r1]

    // Set PB3~6 as output mode
    movs r0, #(0x55<<6) // 0x01
    ldr r1, =GPIOB_MODER
    ldr r2, [r1]
    and r2, #0xFFFFC03F // Mask MODERs
    orrs r2, r2, r0
    str r2, [r1]
    // Set PC13 as input mode
    movs r0, #0
    ldr r1, =GPIOC_MODER
    ldr r2, [r1]
    and r2, #0xF3FFFFFF // Mask MODER
    orrs r2, r2, r0
    str r2, [r1]

    // Set PB3~6 as push-pull output
    movs r0, #(0x0<<3) // 0x0
    ldr r1, =GPIOB_OTYPER
    ldr r2, [r1]
    and r2, #0xFFFFFF87 // Mask MODERs
    orrs r2, r2, r0
    str r2, [r1]

    // Set PB3~6 as high speed mode
    movs r0, #(0xAA<<6) // 0x10
    ldr r1, =GPIOB_OSPEEDR
    ldr r2, [r1]
    and r2, #0xFFFFC03F // Mask MODERs
    orrs r2, r2, r0
    str r2, [r1]
```



```

    // Set PA3~6 as pull-up
    movs r0, #(0xAA<<6) // 0x10
    ldr r1, = GPIOB_PUPDR
    ldr r2, [r1]
    and r2, #0xFFFFC03F // Mask MODERs
    orrs r2, r2, r0
    str r2, [r1]

    // Get output register address
    ldr r3, =GPIOB_ODR
    // Get button register address
    ldr r7, =GPIOC_IDR

    bx lr

```

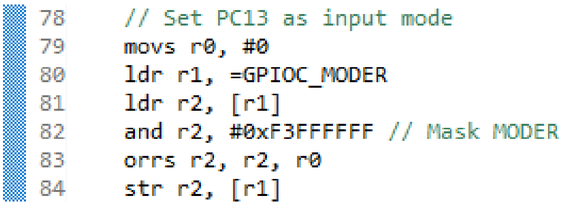


```
DisplayLED: // r1 = shift state, r3 = GPIOB_ODR
    mvn r5, r1
    lsl r5, #3
    str r5, [r3]

    bx lr

Delay:
    ldr r4, =X
    push {lr}
L1:
    bl Button_polling /////////////////////////////////////////
    ldr r5, =pause    //                                     //
    ldr r6, [r5]      // Delete this block to disable button //
    cmp r6, #0        //                                     //
    bne L1            /////////////////////////////////////////

    sub r4, r4, #1
    cmp r4, #0
    beq Break
    ldr r5, =Y
L2:
    sub r5, r5, #1
    cmp r5, #0
    bne L2
    b L1
Break:
    pop {pc}

```

2.2. Push button

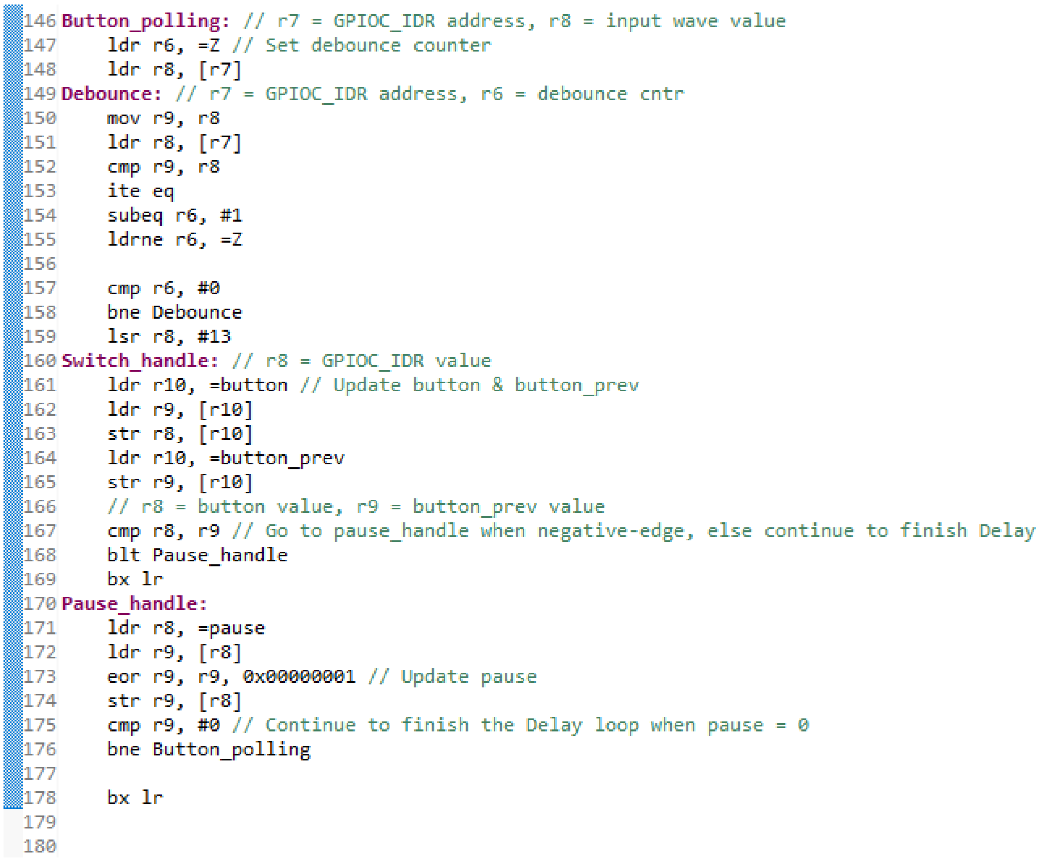
Initialize GPIO PC13 as pull-up input and design a program to polling the state of the user button on board. Controlling the scrolling of the LEDs (Lab4.1) stop and start by a click on the button. (click once to stop scrolling and once more to start scrolling)

The user button on board is connected to PC13. Refer to the lecture slides or STM32L476 datasheet to complete the initialization of GPIOC.



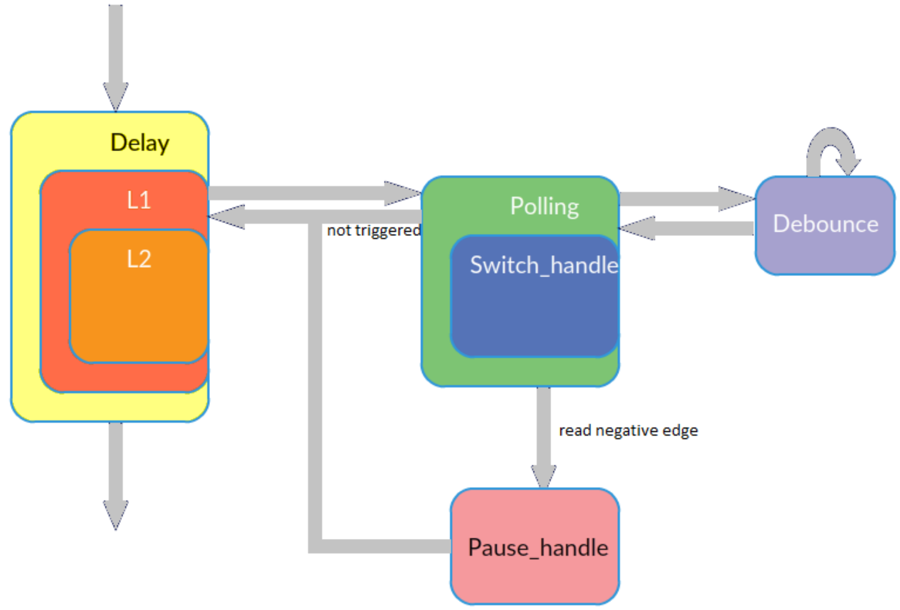
```
    // Set PC13 as input mode
    movs r0, #0
    ldr r1, =GPIOC_MODER
    ldr r2, [r1]
    and r2, #0xF3FFFFFF // Mask MODER
    orrs r2, r2, r0
    str r2, [r1]
```

We use the previous code and add some sections below:



```
Button_polling: // r7 = GPIOC_IDR address, r8 = input wave value
    ldr r6, =Z // Set debounce counter
    ldr r8, [r7]
Debounce: // r7 = GPIOC_IDR address, r6 = debounce cntr
    mov r9, r8
    ldr r8, [r7]
    cmp r9, r8
    ite eq
    subeq r6, #1
    ldrne r6, =Z

    cmp r6, #0
    bne Debounce
    lsr r8, #13
Switch_handle: // r8 = GPIOC_IDR value
    ldr r10, =button // Update button & button_prev
    ldr r9, [r10]
    str r8, [r10]
    ldr r10, =button_prev
    str r9, [r10]
    // r8 = button value, r9 = button_prev value
    cmp r8, r9 // Go to pause_handle when negative-edge, else continue to finish Delay
    blt Pause_handle
    bx lr
Pause_handle:
    ldr r8, =pause
    ldr r9, [r8]
    eor r9, r9, 0x00000001 // Update pause
    str r9, [r8]
    cmp r9, #0 // Continue to finish the Delay loop when pause = 0
    bne Button_polling

    bx lr
```

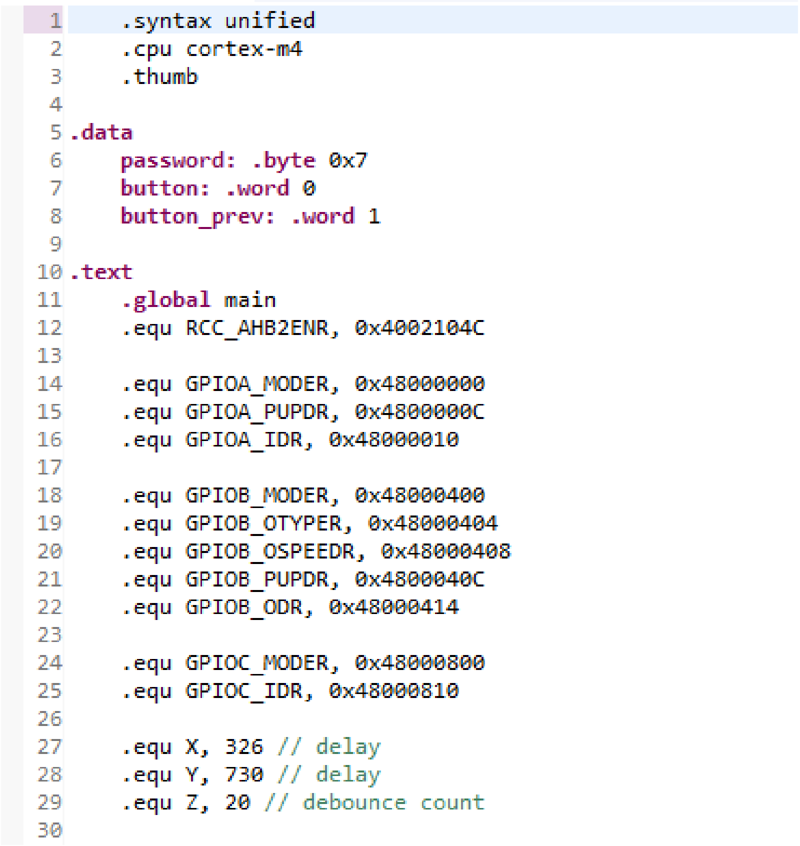
The flow chart of button function.



2.3. Password lock

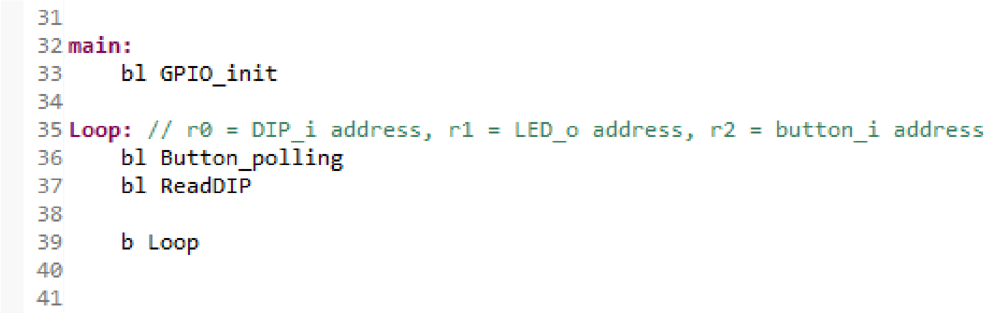
Use breadboard to construct an active low DIP switch circuit and connect P0~P3 to GPIO pins on board.

Declare a 1-byte global variable “password” and implement a simple 4 bits coded lock.



```
    .syntax unified
    .cpu cortex-m4
    .thumb

.data
    password: .byte 0x7
    button: .word 0
    button_prev: .word 1

.text
    .global main
    .equ RCC_AHB2ENR, 0x4002104C

    .equ GPIOA_MODER, 0x48000000
    .equ GPIOA_PUPDR, 0x4800000C
    .equ GPIOA_IDR, 0x48000010

    .equ GPIOB_MODER, 0x48000400
    .equ GPIOB_OTYPER, 0x48000404
    .equ GPIOB_OSPEEDR, 0x48000408
    .equ GPIOB_PUPDR, 0x4800040C
    .equ GPIOB_ODR, 0x48000414

    .equ GPIOC_MODER, 0x48000800
    .equ GPIOC_IDR, 0x48000810

    .equ X, 326 // delay
    .equ Y, 730 // delay
    .equ Z, 20 // debounce count

```

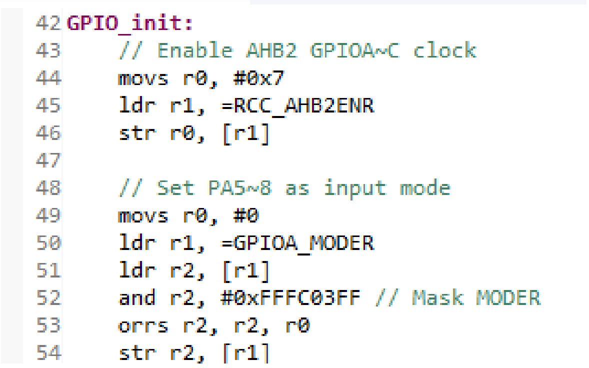
The main loop.



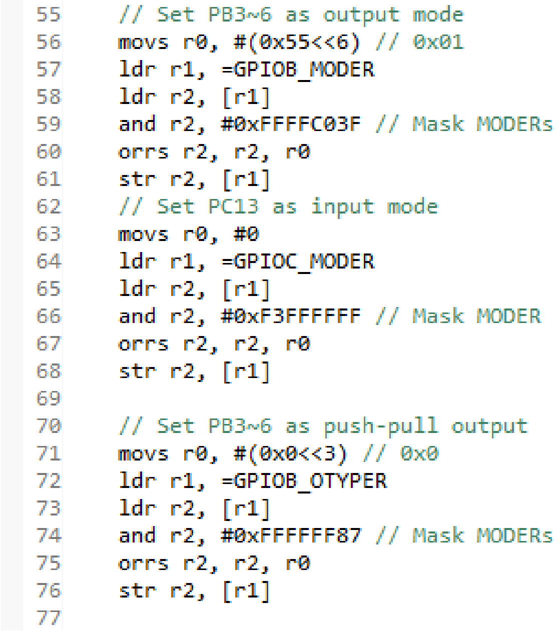
```

main:
    bl GPIO_init

Loop: // r0 = DIP_i address, r1 = LED_o address, r2 = button_i address
    bl Button_polling
    bl ReadDIP

    b Loop


```

Initialization.

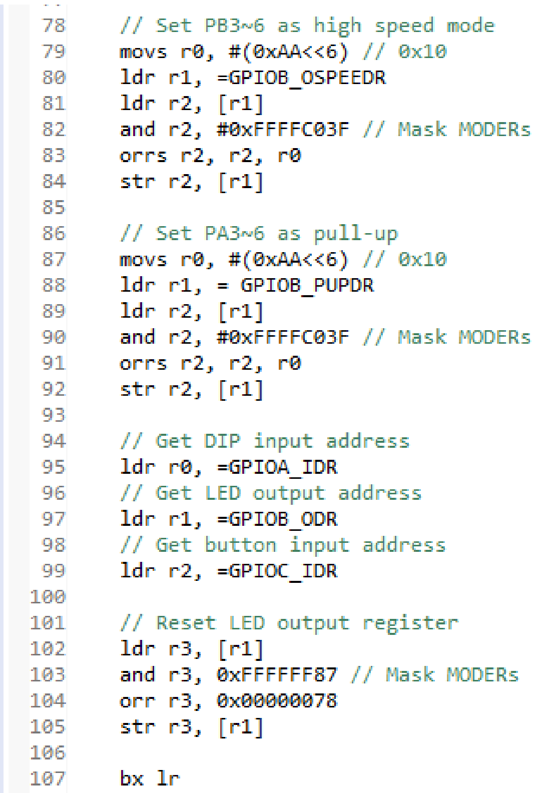
DIP switch connects PA5~8, LEDs connect PB3~6, and button is at PC13.



```
GPIO_init:
    // Enable AHB2 GPIOA~C clock
    movs r0, #0x7
    ldr r1, =RCC_AHB2ENR
    str r0, [r1]

    // Set PA5~8 as input mode
    movs r0, #0
    ldr r1, =GPIOA_MODER
    ldr r2, [r1]
    and r2, #0xFFFC03FF // Mask MODER
    orrs r2, r2, r0
    str r2, [r1]
```

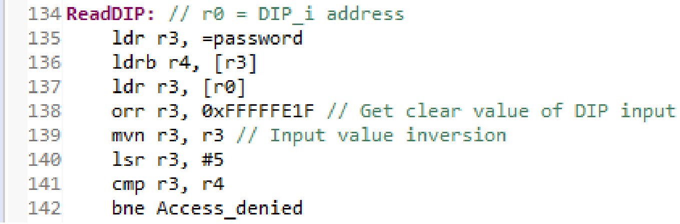


```
    // Set PB3~6 as output mode
    movs r0, #(0x55<<6) // 0x01
    ldr r1, =GPIOB_MODER
    ldr r2, [r1]
    and r2, #0xFFFFC03F // Mask MODERs
    orrs r2, r2, r0
    str r2, [r1]
    // Set PC13 as input mode
    movs r0, #0
    ldr r1, =GPIOC_MODER
    ldr r2, [r1]
    and r2, #0xF3FFFFFF // Mask MODER
    orrs r2, r2, r0
    str r2, [r1]

    // Set PB3~6 as push-pull output
    movs r0, #(0x0<<3) // 0x0
    ldr r1, =GPIOB_OTYPER
    ldr r2, [r1]
    and r2, #0xFFFFFF87 // Mask MODERs
    orrs r2, r2, r0
    str r2, [r1]

```

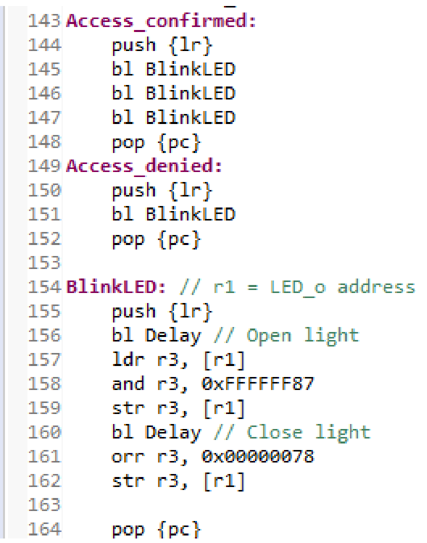


```
    // Set PB3~6 as high speed mode
    movs r0, #(0xAA<<6) // 0x10
    ldr r1, =GPIOB_OSPEEDR
    ldr r2, [r1]
    and r2, #0xFFFFC03F // Mask MODERs
    orrs r2, r2, r0
    str r2, [r1]

    // Set PA3~6 as pull-up
    movs r0, #(0xAA<<6) // 0x10
    ldr r1, = GPIOB_PUPDR
    ldr r2, [r1]
    and r2, #0xFFFFC03F // Mask MODERs
    orrs r2, r2, r0
    str r2, [r1]

    // Get DIP input address
    ldr r0, =GPIOA_IDR
    // Get LED output address
    ldr r1, =GPIOB_ODR
    // Get button input address
    ldr r2, =GPIOC_IDR

    // Reset LED output register
    ldr r3, [r1]
    and r3, 0xFFFFFF87 // Mask MODERs
    orr r3, 0x00000078
    str r3, [r1]

    bx lr
```

Omit the button part. It’s the part of checking password below.

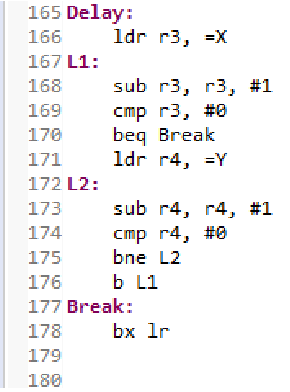


```
ReadDIP: // r0 = DIP_i address
    ldr r3, =password
    ldrb r4, [r3]
    ldr r3, [r0]
    orr r3, 0xFFFFFE1F // Get clear value of DIP input
    mvn r3, r3 // Input value inversion
    lsr r3, #5
    cmp r3, r4
    bne Access_denied
```

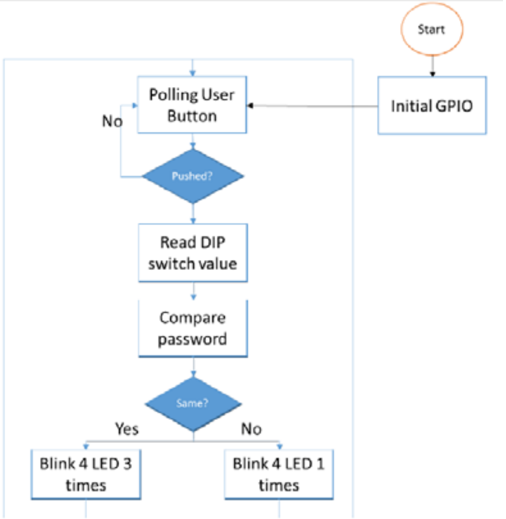


```
Access_confirmed:
    push {lr}
    bl BlinkLED
    bl BlinkLED
    bl BlinkLED
    pop {pc}
Access_denied:
    push {lr}
    bl BlinkLED
    pop {pc}

BlinkLED: // r1 = LED_o address
    push {lr}
    bl Delay // Open light
    ldr r3, [r1]
    and r3, 0xFFFFFF87
    str r3, [r1]
    bl Delay // Close light
    orr r3, 0x00000078
    str r3, [r1]

    pop {pc}
```

Last but not least, the delay function.

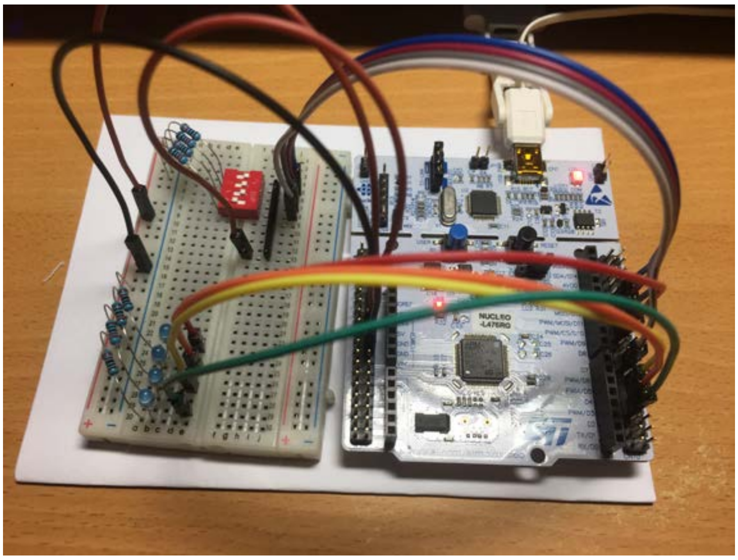


```
Delay:
    ldr r3, =X
L1:
    sub r3, r3, #1
    cmp r3, #0
    beq Break
    ldr r4, =Y
L2:
    sub r4, r4, #1
    cmp r4, #0
    bne L2
    b L1
Break:
    bx lr


```



3. Results and analysis 實驗結果與分析

Here is the circuit, which LED and DIP switch are active low.



4. Conclusions and ideas 心得討論與應用聯想

This lab is a completely different world from previous labs. It took me a lot of time from understanding the circuit to get used of assembly programming logic. It’s hard to believe I spent nights on it. But it still gave me a large sense of accomplishment after seeing the result.