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
```

```
39 Loop: // r4 = flag
40     bl DisplayLED
41     bl Delay
42 
43     ldr r5, =dir
44     ldr r4, [r5]
45     cmp r1, #8 // Set the direction flag
46     it eq
47     moveq r4, #0
48     cmp r1, #1
49     it eq
50     moveq r4, #1
51     str r4, [r5] // Save direction flag
52 
53     cmp r4, #0 // determine the direction to shift
54     ite ne
55     lslne r2, #1 // Shift left
56     lsreq r2, #1 // Shift right
57 
58     mov r1, r2
59     lsl r1, #27
60     lsr r1, #28
61     strb r1, [r0]
62 
63     b Loop
64 
```

Initial the bus clock and setup the GPIO registers and pins

```
65 GPIO_init:
66     // Enable AHB2 GPIOB&C clock
67     movs r0, #0x6
68     ldr r1, =RCC_AHB2ENR
69     str r0, [r1]
70 
71     // Set PB3~6 as output mode
72     movs r0, #(0x55<<6) // 0x01
73     ldr r1, =GPIOB_MODER
74     ldr r2, [r1]
75     and r2, #0xFFFFC03F // Mask MODERs
76     orrs r2, r2, r0
77     str r2, [r1]
78     // Set PC13 as input mode
79     movs r0, #0
80     ldr r1, =GPIOC_MODER
81     ldr r2, [r1]
82     and r2, #0xF3FFFFFF // Mask MODER
83     orrs r2, r2, r0
84     str r2, [r1]
85 
86     // Set PB3~6 as push-pull output
87     movs r0, #(0x0<<3) // 0x0
88     ldr r1, =GPIOB_OTYPER
89     ldr r2, [r1]
90     and r2, #0xFFFFFF87 // Mask MODERs
91     orrs r2, r2, r0
92     str r2, [r1]
93 
94     // Set PB3~6 as high speed mode
95     movs r0, #(0xAA<<6) // 0x10
96     ldr r1, =GPIOB_OSPEEDR
97     ldr r2, [r1]
98     and r2, #0xFFFFC03F // Mask MODERs
99     orrs r2, r2, r0
100    str r2, [r1]
```

```
101
102     // Set PA3~6 as pull-up
103     movs r0, #(0xAA<<6) // 0x10
104     ldr r1, = GPIOB_PUPDR
105     ldr r2, [r1]
106     and r2, #0xFFFFC03F // Mask MODERs
107     orrs r2, r2, r0
108     str r2, [r1]
109
110     // Get output register address
111     ldr r3, =GPIOB_ODR
112     // Get button register address
113     ldr r7, =GPIOC_IDR
114
115     bx lr
116
117 DisplayLED: // r1 = shift state, r3 = GPIOB_ODR
118     mvn r5, r1
119     lsl r5, #3
120     str r5, [r3]
121
122     bx lr
123
124 Delay:
125     ldr r4, =X
126     push {lr}
127 L1:
128     bl Button_polling ////////////////////////////////////////
129     ldr r5, =pause    //                                    //
130     ldr r6, [r5]      // Delete this block to disable button //
131     cmp r6, #0        //                                    //
132     bne L1            ////////////////////////////////////////
133
134     sub r4, r4, #1
135     cmp r4, #0
136     beq Break
137     ldr r5, =Y
138 L2:
139     sub r5, r5, #1
140     cmp r5, #0
141     bne L2
142     b L1
143 Break:
144     pop {pc}
145
```

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)

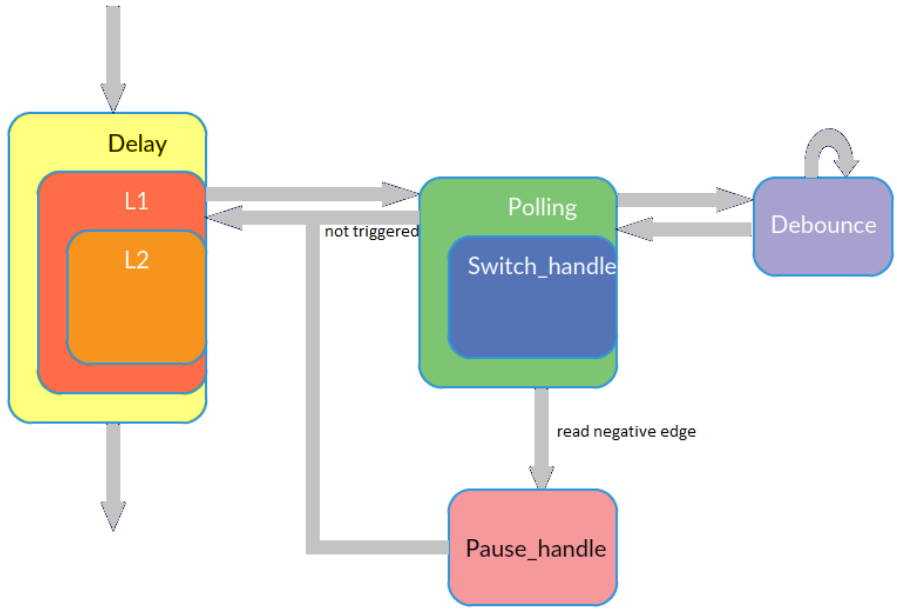
The user button on board is connected to PC13. Refer to the lecture slides or STM32L476 datasheet to complete the initialization of GPIOC.

```
78     // Set PC13 as input mode
79     movs r0, #0
80     ldr r1, =GPIOC_MODER
81     ldr r2, [r1]
82     and r2, #0xF3FFFFFF // Mask MODER
83     orrs r2, r2, r0
84     str r2, [r1]
```

We use the previous code and add some sections below:

```
146 Button_polling: // r7 = GPIOC_IDR address, r8 = input wave value
147     ldr r6, =Z // Set debounce counter
148     ldr r8, [r7]
149 Debounce: // r7 = GPIOC_IDR address, r6 = debounce cntr
150     mov r9, r8
151     ldr r8, [r7]
152     cmp r9, r8
153     ite eq
154     subeq r6, #1
155     ldrne r6, =Z
156
157     cmp r6, #0
158     bne Debounce
159     lsr r8, #13
160 Switch_handle: // r8 = GPIOC_IDR value
161     ldr r10, =button // Update button & button_prev
162     ldr r9, [r10]
163     str r8, [r10]
164     ldr r10, =button_prev
165     str r9, [r10]
166     // r8 = button value, r9 = button_prev value
167     cmp r8, r9 // Go to pause_handle when negative-edge, else continue to finish Delay
168     blt Pause_handle
169     bx lr
170 Pause_handle:
171     ldr r8, =pause
172     ldr r9, [r8]
173     eor r9, r9, 0x00000001 // Update pause
174     str r9, [r8]
175     cmp r9, #0 // Continue to finish the Delay loop when pause = 0
176     bne Button_polling
177
178     bx lr
179
180
```

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]
```

```
55      // Set PB3~6 as output mode
56      movs r0, #(0x55<<6) // 0x01
57      ldr r1, =GPIOB_MODER
58      ldr r2, [r1]
59      and r2, #0xFFFFC03F // Mask MODERs
60      orrs r2, r2, r0
61      str r2, [r1]
62      // Set PC13 as input mode
63      movs r0, #0
64      ldr r1, =GPIOC_MODER
65      ldr r2, [r1]
66      and r2, #0xF3FFFFFF // Mask MODER
67      orrs r2, r2, r0
68      str r2, [r1]
69
70      // Set PB3~6 as push-pull output
71      movs r0, #(0x0<<3) // 0x0
72      ldr r1, =GPIOB_OTYPER
73      ldr r2, [r1]
74      and r2, #0xFFFFFF87 // Mask MODERs
75      orrs r2, r2, r0
76      str r2, [r1]
77
```

```
78      // Set PB3~6 as high speed mode
79      movs r0, #(0xAA<<6) // 0x10
80      ldr r1, =GPIOB_OSPEEDR
81      ldr r2, [r1]
82      and r2, #0xFFFFC03F // Mask MODERs
83      orrs r2, r2, r0
84      str r2, [r1]
85
86      // Set PA3~6 as pull-up
87      movs r0, #(0xAA<<6) // 0x10
88      ldr r1, = GPIOB_PUPDR
89      ldr r2, [r1]
90      and r2, #0xFFFFC03F // Mask MODERs
91      orrs r2, r2, r0
92      str r2, [r1]
93
94      // Get DIP input address
95      ldr r0, =GPIOA_IDR
96      // Get LED output address
97      ldr r1, =GPIOB_ODR
98      // Get button input address
99      ldr r2, =GPIOC_IDR
100
101     // Reset LED output register
102     ldr r3, [r1]
103     and r3, 0xFFFFFF87 // Mask MODERs
104     orr r3, 0x00000078
105     str r3, [r1]
106
107     bx lr
```

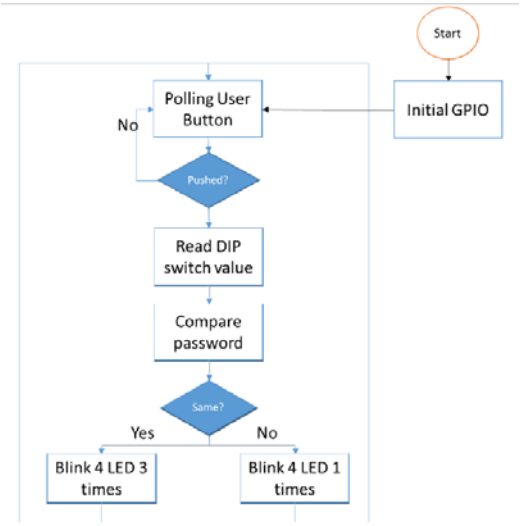
Omit the button part. It’s the part of checking password below.

```
134 ReadDIP: // r0 = DIP_i address
135     ldr r3, =password
136     ldrb r4, [r3]
137     ldr r3, [r0]
138     orr r3, 0xFFFFFE1F // Get clear value of DIP input
139     mvn r3, r3 // Input value inversion
140     lsr r3, #5
141     cmp r3, r4
142     bne Access_denied
```

```
143 Access_confirmed:
144     push {lr}
145     bl BlinkLED
146     bl BlinkLED
147     bl BlinkLED
148     pop {pc}
149 Access_denied:
150     push {lr}
151     bl BlinkLED
152     pop {pc}
153
154 BlinkLED: // r1 = LED_o address
155     push {lr}
156     bl Delay // Open light
157     ldr r3, [r1]
158     and r3, 0xFFFFFF87
159     str r3, [r1]
160     bl Delay // Close light
161     orr r3, 0x00000078
162     str r3, [r1]
163
164     pop {pc}
```

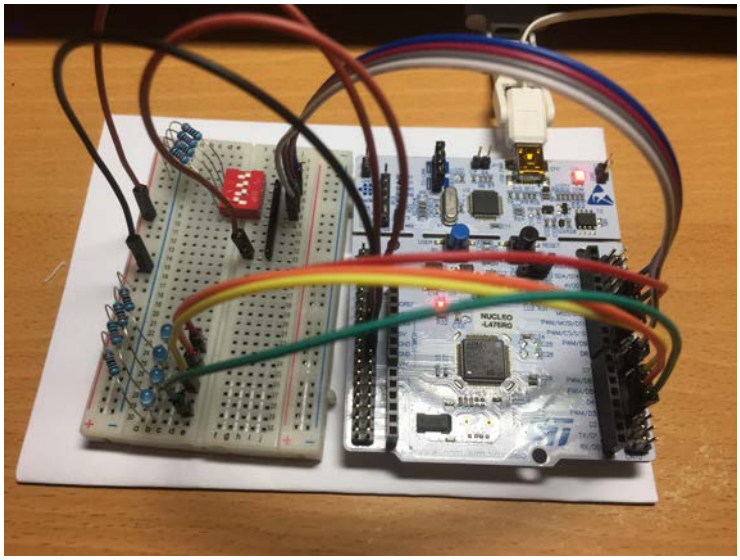
Last but not least, the delay function.

```
165 Delay:
166     ldr r3, =X
167 L1:
168     sub r3, r3, #1
169     cmp r3, #0
170     beq Break
171     ldr r4, =Y
172 L2:
173     sub r4, r4, #1
174     cmp r4, #0
175     bne L2
176     b L1
177 Break:
178     bx lr
179
180
```



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.