

MCSL Lab03

STM32 GPIO System

0410001
Hong-Shuo Chen

1. Experiment Purpose

- Understand the principle of the simple output and input by using STM32 GPIO port
- Design the simple program of the LED pattern displayer
- Understand the principle of the push button and switch

2. Background theory

Please refer to the lecture of GPIO and STM32L4x6 Reference manual

3. Experiment Procedure

3.1. Lab4.1: LED pattern displayer

Please 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”)

Note: Please connect the LEDs to **PB3**, **PB4**, **PB5**, **PB6** on board.

Please complete the program below and let the LEDs blink as the pattern requirement defined.

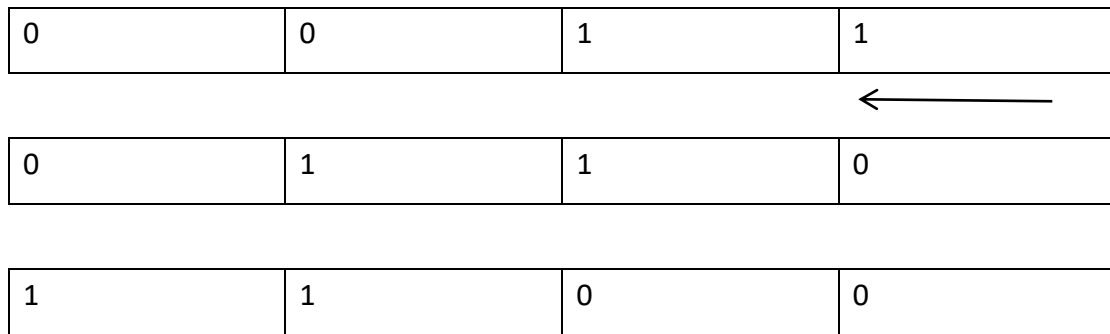
3.1.1. Pattern requirement

“1” represents that LED is on, and “0” represents LED is off.

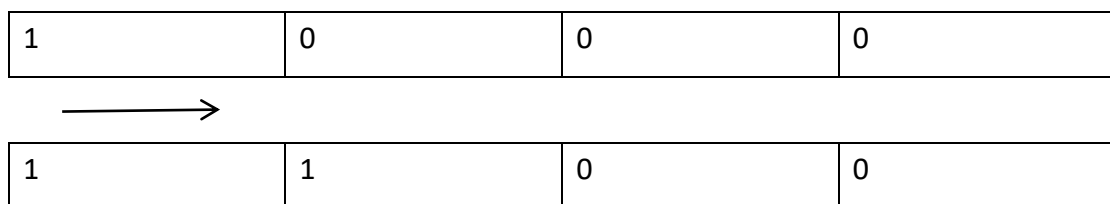
Initial state: The rightest LED is on.

0	0	0	1
---	---	---	---

Then the LED shift left in order every one second. At this time, there should be two LED illuminated.



Change the shifting direction to right when the LEDs' state is "1 0 0 0".



Change the shifting direction to left when the LEDs' state back to the initial state (0 0 0 1). Repeat the process above.

Please complete the program below and use the variable "leds" to record the LEDs' states. Using function "DisplayLED" to output the "leds" value to the LEDs to display.

Note: You may need to use LSL or LSR instructions to shift bits.

```
.data
    leds: .byte 0
.text
    .global main
main:
    BL    GPIO_init
    MOVS  R1, #1
    LDR   R0, =leds
    STRB  R1, [R0]
Loop:
    //TODO: Write the display pattern into leds variable
    BL    DisplayLED
    BL    Delay
    B     Loop
GPIO_init:
    //TODO: Initial LED GPIO pins as output
    BX  LR
DisplayLED:
    BX  LR
Delay:
    //TODO: Write a delay 1sec function
    BX  LR
```

3.2. Lab4.2 Push button

Please 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 by a click on button (click once to stop scrolling and once more to restart scrolling)

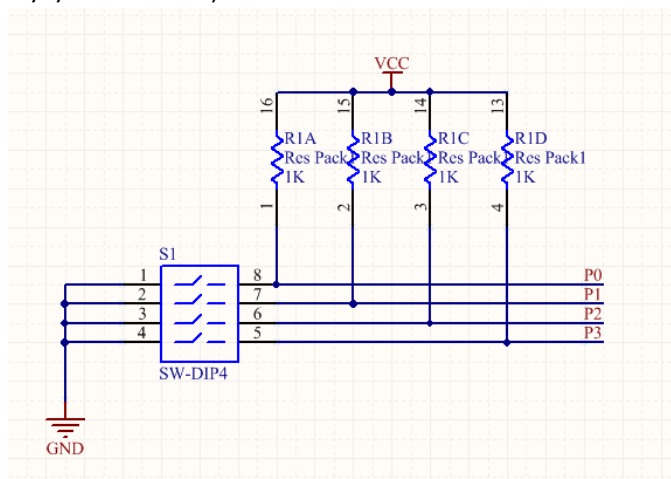
Note: The user button on board is connected to PC13. Please refer to the lecture slides or STM32L476 datasheet to complete the initialization of GPIOC.

3.2.1 Debounce

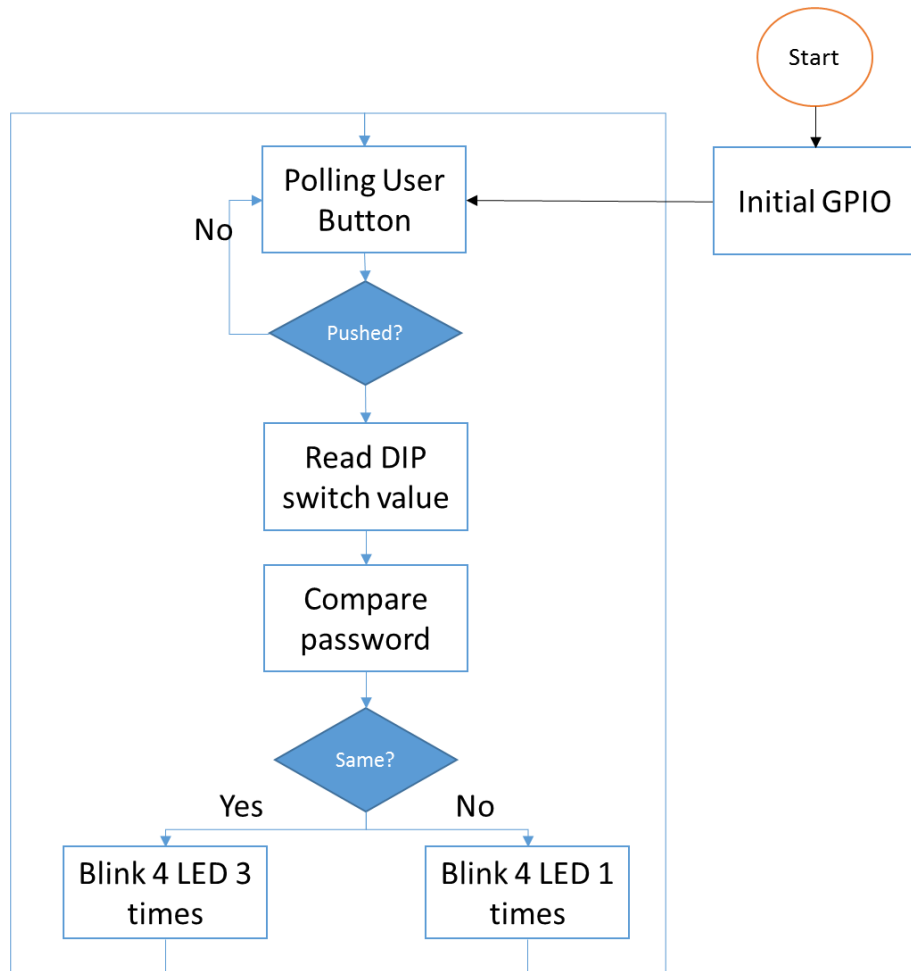
Please solve the button bounce problem using software debounce.

3.3. Lab4.3 password lock

Please use breadboard to construct an active low DIP switch circuit and connect P0~P3 to GPIO pins on board. (You could choose the pins by yourselves)



Please declare a 1 byte global variable “password” and implement a simple 4 bits coded lock. Referring to the process below:

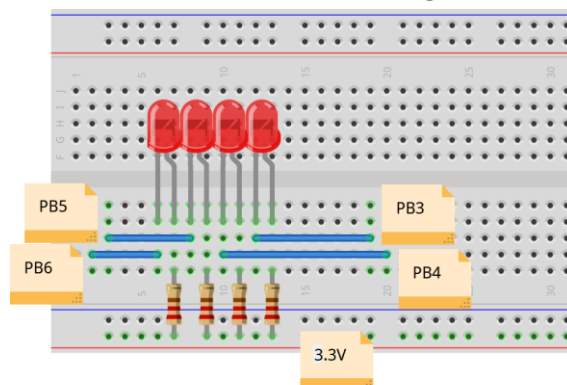


Note: Defining DIP switch ON as “1”, OFF as “0”. Thus, when user input “ON ON OFF OFF”, it’s code is “1 1 0 0”. Please set the blink frequency to 0.5s.

4. Results and Discussion

4.1. Lab4.1: LED pattern displayer

Hardware Design: It is an active low LED circuit. When the output of the PB3, PB4, PB5, PB6 are high voltage, the LEDs will be turned off, because there are no voltage differences between LEDs and resistors. On the other hand, when the outputs are low, there are voltage differences between the LEDs, so the LEDs will be turned on. This is the active low circuit. The figure below is what I implement in Lab4.1.



Part of my code with annotation and further explanation

GPIO_init:

```
//TODO: Initial LED GPIO pins as output
// STM32L476 have port A~H GPIO port connect on AHB2 bus
// Enable AHB2 clock
```

6.4.17 AHB2 peripheral clock enable register (RCC_AHB2ENR)

Address offset: 0x4C

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access

Note: When the peripheral clock is not active, the peripheral registers read or write access is not supported.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	RNG EN	Res.	AESEN
													rw		rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	ADCEN	OTGFSEN	Res.	Res.	Res.	Res.	GPIOH EN	GPIOG EN	GPIOF EN	GPIOE EN	GPIOD EN	GPIOC EN	GPIOB EN	GPIOA EN
		rw	rw					rw	rw	rw	rw	rw	rw	rw	rw

```
movs    r0, #0x2
ldr      r1, =RCC_AHB2ENR
str      r0, [r1]
```

```
// Set PB3, PB4, PB5, PB6 as output mode
```

7.4.1 GPIO port mode register (GPIOx_MODER) (x = A..H)

Address offset: 0x00

Reset values:

- 0xABFF FFFF for port A
- 0xFFFF FEBF for port B
- 0xFFFF FFFF for ports C..G,
- 0x0000 000F for port H

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODE15[1:0]		MODE14[1:0]		MODE13[1:0]		MODE12[1:0]		MODE11[1:0]		MODE10[1:0]		MODE9[1:0]		MODE8[1:0]	
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODE7[1:0]		MODE6[1:0]		MODE5[1:0]		MODE4[1:0]		MODE3[1:0]		MODE2[1:0]		MODE1[1:0]		MODE0[1:0]	
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

Bits 2y+1:2y **MODEy[1:0]**: Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O mode.

- 00: Input mode
- 01: General purpose output mode
- 10: Alternate function mode
- 11: Analog mode (reset state)

```
movs    r0, #0x1540
ldr     r1, =GPIOB_MODER
ldr     r2, [r1]
and     r2, #0xFFFFC03F //Mask MODER3,4,5,6
orrs    r2, r2, r0
str     r2, [r1]
```

7.4.3 GPIO port output speed register (GPIOx_OSPEEDR) (x = A..H)

Address offset: 0x08

Reset value:

- 0x0C00 0000 for port A
- 0x0000 0000 for the other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OSPEED15[1:0]		OSPEED14[1:0]		OSPEED13[1:0]		OSPEED12[1:0]		OSPEED11[1:0]		OSPEED10[1:0]		OSPEED9[1:0]		OSPEED8[1:0]	
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OSPEED7[1:0]		OSPEED6[1:0]		OSPEED5[1:0]		OSPEED4[1:0]		OSPEED3[1:0]		OSPEED2[1:0]		OSPEED1[1:0]		OSPEED0[1:0]	
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

Bits 2y+1:2y **OSPEEDy[1:0]**: Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O output speed.

- 00: Low speed
- 01: Medium speed
- 10: High speed
- 11: Very high speed

Note: Refer to the device datasheet for the frequency specifications and the power supply and load conditions for each speed.

```
// Default PB3, PB4, PB5, PB6 is Pull-up output, no need to set
// Set PB3, PB4, PB5, PB6 as high speed mode
movs r0, #0x2A80
ldr      r1, =GPIOB_OSPEEDR
strh r0,[r1]

ldr      r1, =GPIOB_ODR
BX LR
```

Delay:

```
//TODO: Write a delay 1sec function
ldr      r3, =X
L1: ldr      r4, =Y
L2: subs r4, #1
    bne L2
    subs r3, #1
    bne L1
BX LR
```

$$\text{Time} \approx 2 + (2 + (1 + 3) * Y - 2 + 1 + 3) * X - 2 + 1$$

By default, our CPU(STM32L476) runs on 4MHz, 1cycle = 0.25u, So I set X = Y = 1000(0x3e8), approximately there will generate 4 million cycles.

The results of leds are as below. The address of leds is 0x20000000. When the leds equals to 0, it means there is no shift. When the leds equals to 1, it means there is one shift, and two LEDs will be turned on. When the leds equals to 6, it means it shift back from left to right, and two LEDs in the middle will be turned on. There are totally 8 conditions, and we could use one byte to record it as the program does. And the LEDs blinks as expected which there is a one second delay between each pattern.

Address	0 - 3
0000000020000000	00000000

Address	0 - 3
0000000020000000	01000000

Address	0 - 3
0000000020000000	02000000

Address	0 - 3
0000000020000000	04000000

Address	0 - 3
0000000020000000	06000000

4.2. Lab4.2 Push button

Hardware Design: It is the same as Lab4.1. But we use the button on the board which is connected to PC13.

GPIO_init:

```
//TODO: Initial LED GPIO pins as output

movs    r0, #0x6

ldr     r1, =RCC_AHB2ENR

str     r0,[r1]

// Set PC13 MODER as input

ldr     r1, =GPIOC_MODER

ldr     r0,[r1]

ldr     r2, =#0xF3FFFFFF

and     r0,r2

str     r0,[r1]

// Set data register address

ldr     r2, =GPIOC_IDR

ldr     r1, =GPIOB_ODR

BX LR

ldr     r3,[r2] // Read input data register


movs    r4,#1

lsl     r4,#13

ands    r3,r4

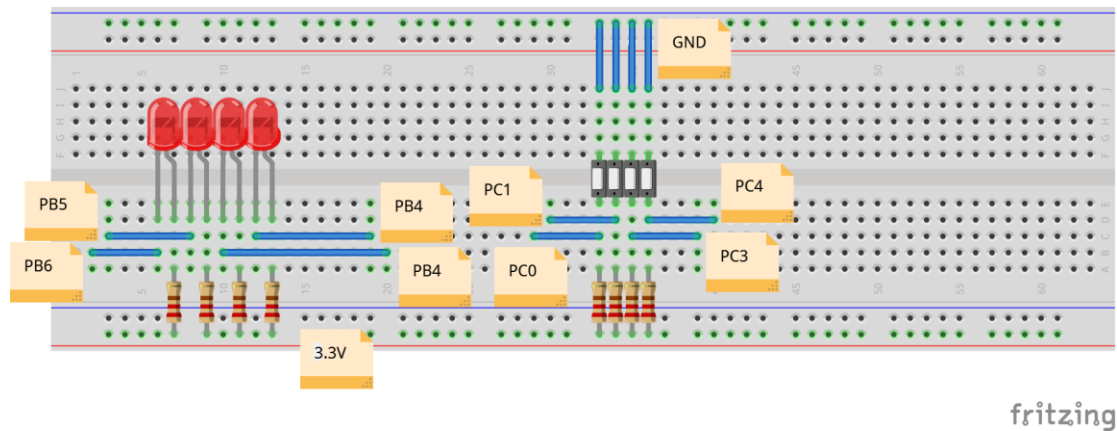
beg     do_pushed
```

The default value of the input is 1. If we press the button, the value will change to 0.

➤  IDR	0x00002000	00000000000000000000 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
---	------------	--

4.3. Lab4.3 password lock

Hardware Design: It is active low circuit. When switch 'ON' Px get GND level (0), 'OFF' get VCC level('1')



When the switch is set as on on off off, because it is active low the input will be 0 0 1 1 as follow figure. The result value will be store in the IDR.

```
> IDR 0x00002003 00000000000000000000_0_1_0_0_0_0_0_0_0_0_0_1_1
```

My password is set as 1100. And the switches should be on on off off.

5. Reviews and Applications

This is the first time in this class that we use the material besides the board, such as resistors, LEDs, switches and so on which is really interesting. I think we would use further component in order to do more things. The setting of the GPIO is much more complex as I predicted. I think the main reason is that we use the Assembly Language to implement it. However, it helps me know much more detailed thing behind the board and can thoroughly know how the board works. It really a nice Lab for me.