Lab4 STM32 GPIO System

實驗四 STM32 GPIO System

0516322朱蝶、0516059劉嘉豪

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
了解STM32基本輸出入I/O port使用原理
設計簡易LED跑馬燈程式
了解按鈕與指撥開關使用原理

2. Lab principle 實驗原理

Please refer to the course GPIO lecture slide and STM32L4x6 Reference manual.

請參考上課 GPIO 講義與 STM32L4x6 Reference manual。

3. Steps 實驗步驟

3.1. Lab4.1 : LED pattern displayer (40%)

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.

參考講義上的教學完成4個 GPIO output 初始化，並在麵包板上完成4個為 **Active Low** 的LED燈電路 (當GPIO輸出 "1" 時 燈暗，"0" 代表燈亮)。
Note: LED 需連接至實驗板上的 PB3, PB4, PB5, PB6

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

完成依以下 pattern 閃爍的跑馬燈程式。

3.1.1. Pattern requirement

"1" represents that LED is on, and "0" represents LED is off.
"1" 代表 LED 亮，"0" 代表 LED 暗

Initial state: The rightest LED is on.
初始狀態：最右邊的LED亮

t = 0s	0	0	0	1

Then the LED shift left in order every one second. At this time, there should be two LED illuminated.
接著每一秒鐘 LED 依序往左位移，此時會有2個 LED 亮

t = 1s	0	0	1	1
t = 2s	0	1	1	0
t = 3s	1	1	0	0

Change the shifting direction to right when the LEDs' state is "1 0 0 0".
當 LED 亮至最左邊時 ("1 0 0 0") 的下一秒改變位移方向，由左至右

t = 4s	1	0	0	0
t = 5s	1	1	0	0
t = 6s	0	1	1	0
t = 7s	0	0	1	1
t = 8s	0	0	0	1

Change the shifting direction to left when the LEDs' state back to the initial state ("0 0 0 1"). Repeat the process above.
當回至初始狀態 ("0 0 0 1") 後在改變位移方向，並重複以上步驟

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.

完成以下程式碼，並利用 "leds" 這個變數紀錄目前位移數值，與 "DisplayLED" 函式輸出 "leds" 數值顯示至4個LED上。
Note: 需用位移指令LSL或LSR進行數值位移

```
.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:
   //TODO: Display LED by leds
   BX LR

Delay:
   //TODO: Write a delay 1 sec function
   BX LR
```

1. GPIO_init

First, we declare the GPIO port B at text section. From the powerpoint in the lecture and the manual on the Internet, we can find out the address and offset, thus define the addresses. Onesec is set to 80M because in the powerpoint, the frequency is 80M Hz.

0x4800 0400 - 0x4800 07FF	1 KB	GPIOB	*Section 7.4.13: GPIO register map*

```
offset  register       description
======  ========       ===========
 0x00   GPIO_MODER     port mode register
 0x04   GPIO_OTYPER    output type register
 0x08   GPIO_OSPEEDR   output speed (slew-rate) register
 0x0c   GPIO_PUPDR     pull-up/pull-down register
 0x10   GPIO_IDR       input data register
 0x14   GPIO_ODR       output data register
 0x18   GPIO_BSRR      bit set/reset register
 0x1c   GPIO_LCKR      configuration lock register
 0x20   GPIO_AFRL      alternate function low register (pins 0-7)
 0x24   GPIO_AFRH      alternate function high register (pins 8-15)
```

By default, our CPU runs on 4MHz, 1 cycle=0.25*10^-6 s
because our delay has sub, cmp, bne, so it is 1+2+1=4cycles=1*10^-6s
so onesec is 10^6.

```
.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 onesec, 1000000
```

And use their address to set up the mode in GPIO_init
Reference is the powerpoint in the lecture.

```
GPIO_init:
      //TODO: Initial LED GPIO pins as output
      ldr r0, =RCC_AHB2ENR
      ldr r1, [r0]
      mov r1, 0x00000002 // 1 on second bit enables port b
      str r1, [r0]

     //enable the port b GPIOB_MODER for output mode
     // need 01 in those modes
      ldr r0, =GPIOB_MODER
      ldr r1, [r0] //init value 0xFFFF FEBF
      mov r1, 0xFFFFD57F //FFFF 1101 0101 0111 1111
      str r1,[r0]

      //otype is default to push pull , no need to change

      //set speed , defulat 0x00000000 low speed
      // now use high speed 10
      ldr r0, =GPIOB_OSPEEDR
      mov r1, 0x00002A80  //0010 1010 1000 0000
      str r1, [r0]

      //r2 led output value address
      ldr r2, =GPIOB_ODR
      BX LR
```

2. DisplayLED

We initialized LEDS with 0xfff3, because the requirement is active low led lights, and at first, the rightest led needs to be on, we set 2 and 3 as 0, others are 1.
We then store it into [r2].

```
DisplayLED:
      mov r1, 0xfff3 //1111 1111 1111 0011
      strh r1, [r2]
      bx lr
```

Then, run in a loop. We first switch_left, and then switch_right.

```
Loop:
      //TODO: Write the display pattern into leds variable
      BL DisplayLED
      switch_left:
      mov r4, 0x0
      b goleft

      switch_right:
      mov r5, 0x0
      b goright
      B           Loop
```

In goleft, we do delays and keep left shift LEDs pattern. And do inversely in goright.

```
goleft:
      ldr r3, =onesec
      bl Delay
      lsl r1, r1, #1
      cmp r4, #3
      it eq
      moveq r1,0xff3f //just light up the leftmost led
      // 1111 1111 0011 1111
      strh r1,[r2] //store to output value

      add r4, r4, #1
      cmp r4,#4
      beq switch_right
      bne goleft
```

In Delay, we sub r3, which is from 100M, and continue delaying until r3 is zero.

```
Delay:
      //TODO: Write a delay 1 sec function
      sub r3, r3, #1
      cmp r3, #0
      bne Delay
      bx lr
```

3. Problems and solutions

I think the initial part really costs us a lot of time because we don't know how to set those value at first. After searching in the Internet and saw the notes , we gradually understand what we should set in initialization part.
As for the hardware part, it's actually a really small device and everything is tiny. The wire is hard to cut and it was not easy to put them on the board, but after some frustration and a lot of time, we fortunately made it.
Why the led is active low (0 is light) is because when it is low voltage, there will be current pass by since there is voltage difference, so the led will light up.

3.2. Lab4.2 : Push button (20%)

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 (Lab4.1) stop and start by a click on button. (click once to stop scrolling and once more to start 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.

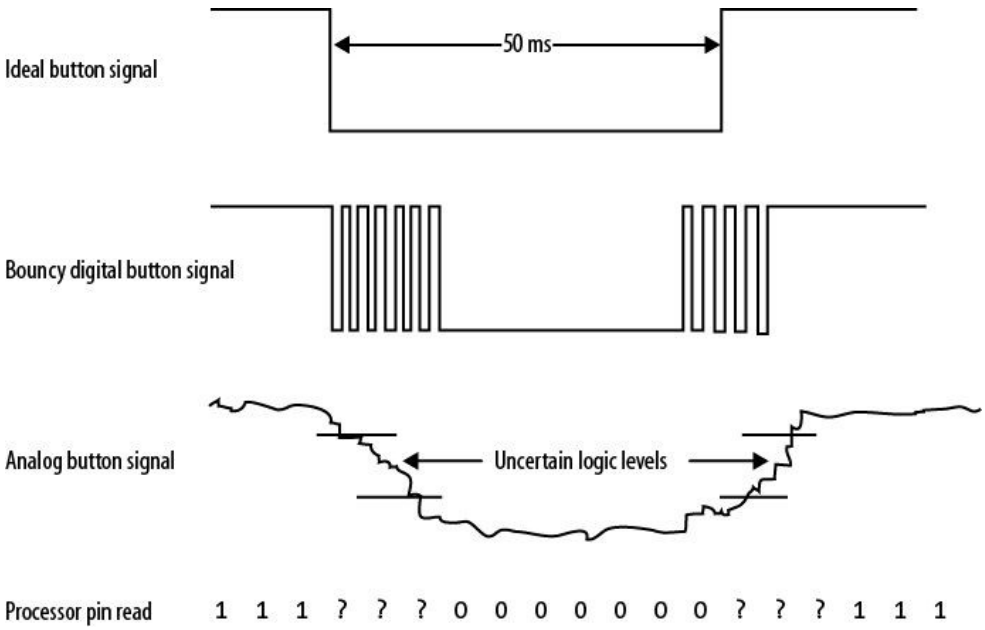
初始化 GPIO PC13 為 pull-up input，並設計一程式輪詢 (polling) 實驗板上的 User button 狀態，藉由按下按鈕去控制跑馬燈 (Lab4.1) 的停止與啟動。(按一次停止再按一次啟動)

Note: 開發板上的 User button 是連接在 PC13 上，請自行參考講義或 STM32L476 datasheet 完成 GPIOC 初始化。

3.2.1. switch bounce 開關彈跳

Please solve the button bounce problem using software debounce.

請使用軟體去彈跳，解決按鍵彈跳問題。



Mechanical bounce of button switch：

The button is a mechanical device. After pressing, the internal knot will bounce back and forth for a few milliseconds before it stabilizes. During the time when the bounce is removed, both low and high signals are detected, causing false positives.

按鍵開關之機械彈跳現象：

按鍵是機械裝置，按壓後，在穩定之前，內部連結會在幾毫秒間來回彈跳。在消除彈跳的這段時間裡，low 和 high 的訊號都會偵測到，造成誤判。

1. Initialize PC13

We need to plus these declaration because of the use of port C. And the declaration is based on the addresses and the offsets. The button is an input device, so this time, we need to declare IDR. Here, we set the interval_cnt as 20,0000, and it's a quarter of a second.

```
.equ GPIOC_MODER  ,   0x48000800
.equ GPIOC_OTYPER ,   0x48000804
.equ GPIOC_OSPEEDR,   0x48000808
.equ GPIOC_PUPDR  ,   0x4800080c
.equ GPIOC_IDR    ,   0x48000810
.equ interval_cnt, 200000
```

We need to add these initialization, and we load the address of ODR and IDR into r2 and r4.

```
//enable port b and c to do the tasks
ldr r0, =RCC_AHB2ENR
mov r1, 0x6 //portc+b
str r1, [r0]

//enable port c GPIOC_MODER for input mode
ldr r0, =GPIOC_MODER
ldr r1, [r0]
//clear pc13 to zero
and r1, r1, 0xf3ffffff //1111 0011 1111 1111
str r1,    [r0]

ldr r2, =GPIOB_ODR
ldr r4, =GPIOC_IDR
BX LR
```

2. Debouncing

Because there's debouncing problem, which is after pressing, the internal knot will bounce back and forth for a few milliseconds before it stabilizes, we check the button every cycle, and set a threshold for it. If the input of the button is zero, then we accumulate the threshold, when it is one, we then move the value to one, and it means to accumulate again, since it is not stable. If it achieved the threshold, we view it as stable.

```
check_button: //check every cycle, and accumulate 1
      ldr r7, [r4] //fetch the data from button
      lsr r7, r7, #13
      and r7, r7, 0x1 //filter the signal
      cmp r7, #0
      it eq
      addeq r0, r0 ,#1 //accumulate until the threshold

      cmp r7, #1 //not stable, go back to accumulate again
      it eq
      moveq r0, #1

      cmp r0, #1000 //threshold achieved BREAKDOWN!
      it eq
      eoreq r6, r6, #1 //r6^=1

      b check_end
```

In goleft, we make the threshold to be zero, and do delays. After that, we see if the button is pressed or not, if r6 is zero, then stop moving, branch to stop_move_left, if not, keep left shift LEDs pattern. And do inversely in goright.

```
goleft:
      ldr r3, =interval_cnt
      mov r0, #0 //threshold 1000 stable 1000 secs
      bl Delay

       cmp r6, #0 //1 move 0 stop_move_left
      beq stop_move_left

      lsl r1, r1, #1
      cmp r9, #3
      it eq
      moveq r1,0xff3f //just light up the leftmost led
      // 1111 1111 0011 1111
      add r9, r9, #1

     stop_move_left:
      strh r1,[r2]
      cmp r9,#4  //compare if need to switch direction
      beq switch_right
      bne goleft
```

In Delay, we sub r3, and branch to check_button to check, then check_button will branch to check_end, then we compare r3 with zero, if not, go back to delay.

```
Delay:
      //TODO: Write a delay 1 sec function
      sub r3, r3, #1
      b check_button
check_end:
      cmp r3, #0
      bne delay
      bx lr
```

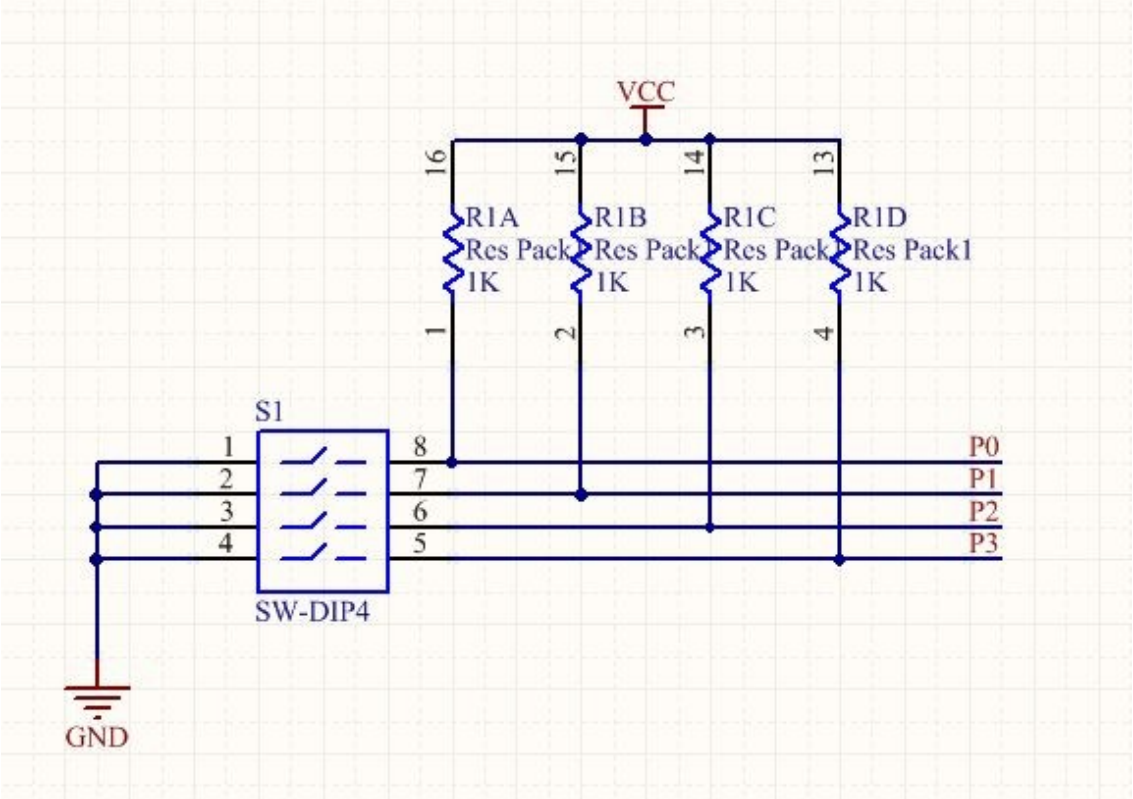
3. Problems and solutions

When we finish the third problem and want to check this problem again, we found out that pressing the buttom didn't stop the led lightbulbs, and we even go back and check whether the code is wrong or not. Actually, it's because we switch the DIP on. When we switch it back to all down, then the button did work. But we don't know why, is it because lsl #13 can't shift all those numbers in the DIP input?

3.3. Lab4.3 : 密碼鎖 (40%)

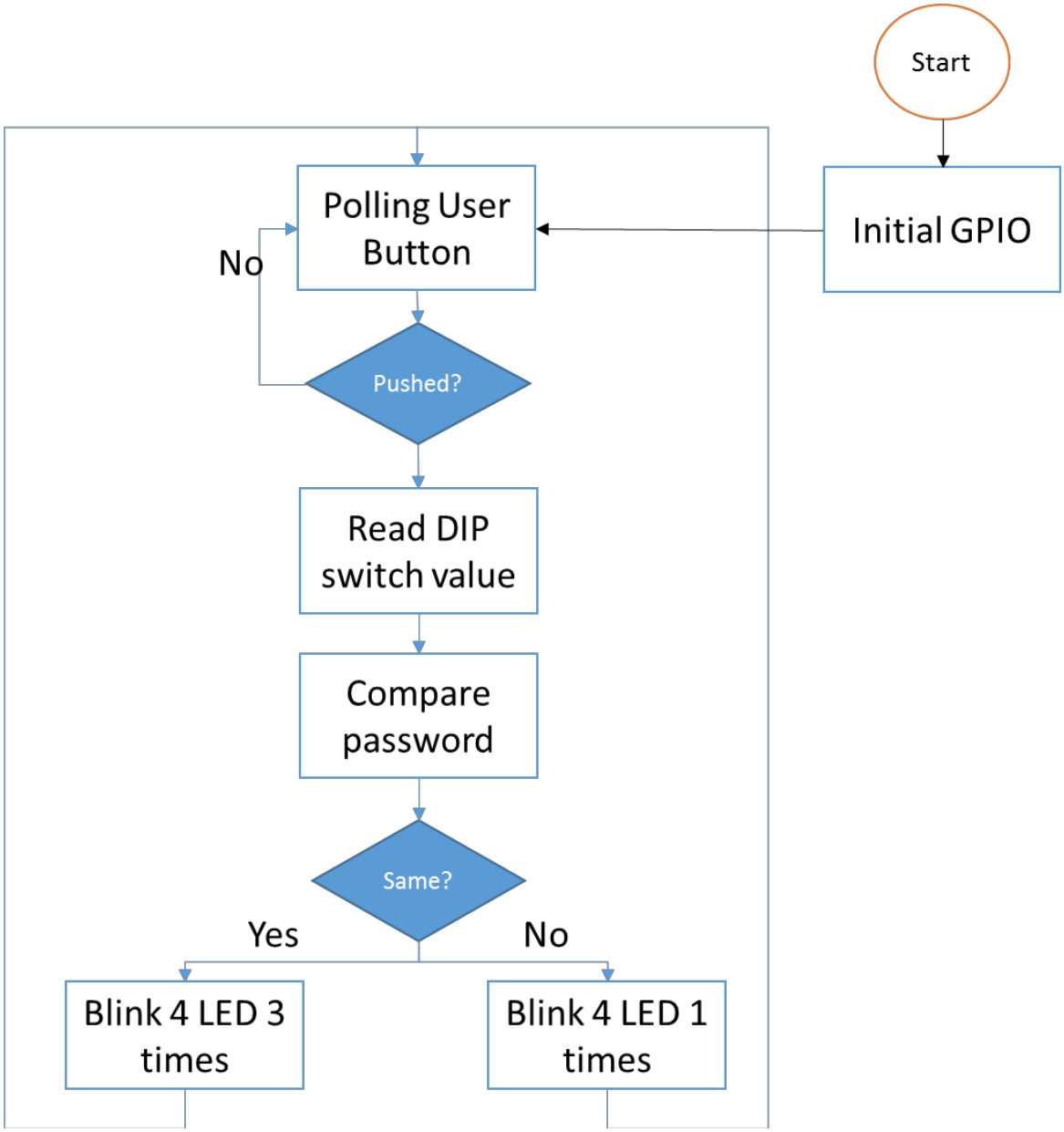
Please use breadboard to construct an active low DIP switch circuit and connect P0~P3 to GPIO pins on board. (You could choose the pins by yourselves)

利用麵包板連接 DIP switch的 active low 電路並連接 P0~P3 至實驗板的 GPIO pins (同學可自行決定連接的 pins)



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

在程式中宣告一個 password 1byte 全域變數並實做一個簡易的 4bit 密碼鎖程式，其流程如下:



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.

Note: DIP switch ON 代表 "1", OFF 代表 "0"，若使用者輸入 "ON ON OFF OFF" 則代表 "1 1 0 0"。 Blink 時間間隔 0.5s

1. Debouncing

The Debouncing method is just like problem two, which use a threshold accumulation method. After finish, branch to check_end. If the buttom is pressed, branch to check_lock, otherwise, go back to the loop.

```
check_button: //check every cycle, and accumulate 1
      ldr r7, [r4] //fetch the data from button
      lsr r7, r7, #13
      and r7, r7, 0x1 //filter the signal
      cmp r7, #0
      it eq
      addeq r0, r0 ,#1 //accumulate until the threshold
```

```
      cmp r7, #1 //not stable, go back to accumulate again
      it eq
      moveq r0, #1

      cmp r0, #1000 //threshold achieved BREAKDOWN!
      it eq
      eoreq r6, r6, #1 //r6^=1

      b check_end

check_end:
      cmp r6, #1 //button is pressed, check lock
      beq check_lock

      blink_end:
      mov r6, #0
      B          Loop
```

We load the dip's input value into r7, and 0xf0(1111 0000), to just preserve the 4-7bit, then shift right four bit, let the input be at the most right four bit, so that it can be compared with the password. If the r7, r9 is the same, them blink three times, otherwise, blink once.

```
check_lock:
      ldr r3, =wait_for_input
      ldr  r7, [r8]
      and  r7, 0xf0
      lsr  r7, #4
      ldr  r9, =password
      ldrb r9, [r9]

      cmp r7, r9
      beq led_blink_three
      b led_blink_once
```

This is the example of blinking three times, blinking once is the same as this.

```
led_blink_three:
      ldr r1, =LED_ALLON//ff|1000|0111|
      strh r1, [r2]
      ldr r3, =quarter_sec
      bl delay_quarter_sec

      ldr r1, =LED_ALLOFF //ff|1111|1111|
      strh r1, [r2]
      ldr r3, =quarter_sec
      bl delay_quarter_sec

      ldr r1, =LED_ALLON
      strh r1, [r2]
      ldr r3, =quarter_sec
      bl delay_quarter_sec

      ldr r1, =LED_ALLOFF
      strh r1, [r2]
      ldr r3, =quarter_sec
```

```
    bl delay_quarter_sec

    ldr r1, =LED_ALLON
    strh r1, [r2]
    ldr r3, =quarter_sec
    bl delay_quarter_sec

    ldr r1, =LED_ALLOFF
    strh r1, [r2]
    ldr r3, =quarter_sec
    bl delay_quarter_sec
    b blink_end
```

2. Problems and solutions

1)We did our homework for several days, and the next day, we decided to plug all wires out and do it again, then we found out that no led lightbulbs are blinking. So we were really anxious and couldn't find out what's wrong. Fortunately, we decided to unplug the resistors, and plug it upside down. Oh! it worked! So, we think that the side we first put makes resistance too big, that's why the led lightbulb don't have enough current to light up. Yeah!!!

2)We really don't know how to plug those wires, so we experience all kinds of method many times. We really spent lots of time on this lab! After demo time, we realize how to plug those wires and found out our dip broke up, so after a new dip, it works!

Overall Feedback

We think we are really concentrated on the class, but still have no idea about how to do this lab, so, after lots of experiments and efforts, our conclusion is that we are both talented psychics. 耶：)