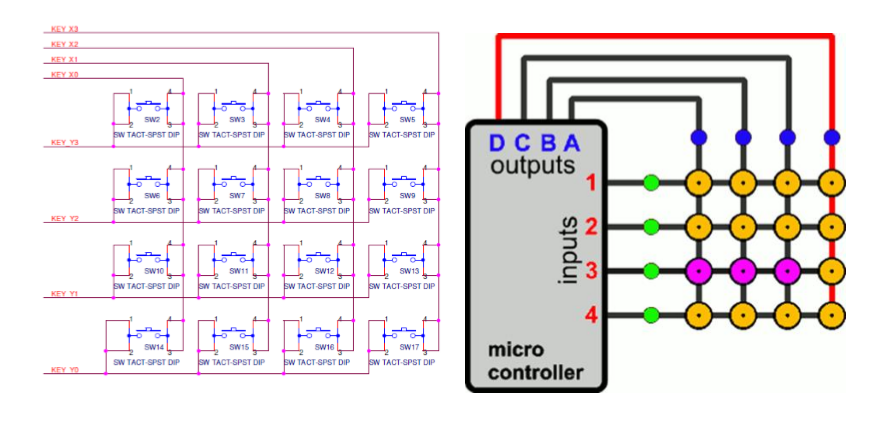
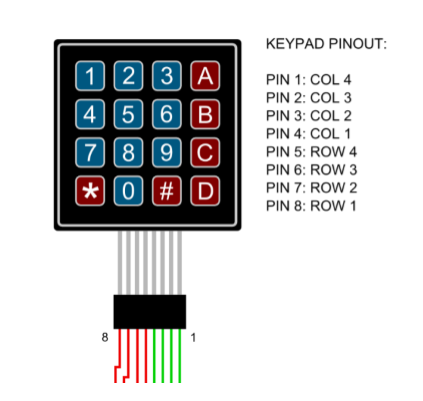
實驗六 7-Seg LED

0410137劉家麟 0416324 胡安鳳

# 實驗目的

* 了解STM32使用原理
* 了解如何使用 C code 控制 STM32
* 設計 7-Seg LED 和 keypad 程式

# 實驗原理

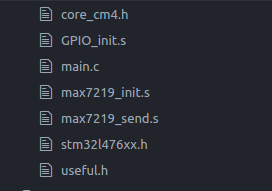
Keypad 電路組成如下，主要是一個 4x4 的鍵盤按鈕所組成會用到 4 個 Input pin 與 4 個 Output pin，其控制原理是利用 Output pin 掃描的方式來決定目前所選擇 到的是哪一行按鍵，例如當 KEY X0~3 輸出 1000 而此時若 KEY Y0~3 所讀到的 值是 1000 的話則代表 SW14 按鈕被按下。  
The circuit diagram of keypad is given below. You’re supposed to use 4 input pins and 4 output pins. Use output pins to determine which row you’re scanning. For example, when output value of KEY X0~3 is 1000 and input value of KEY Y0~3 is 1000, then we can say that SW14 is pressed.  
  


# 實驗步驟

## Lab6.0: Lab 6.0: Max7219 displayer (10%)

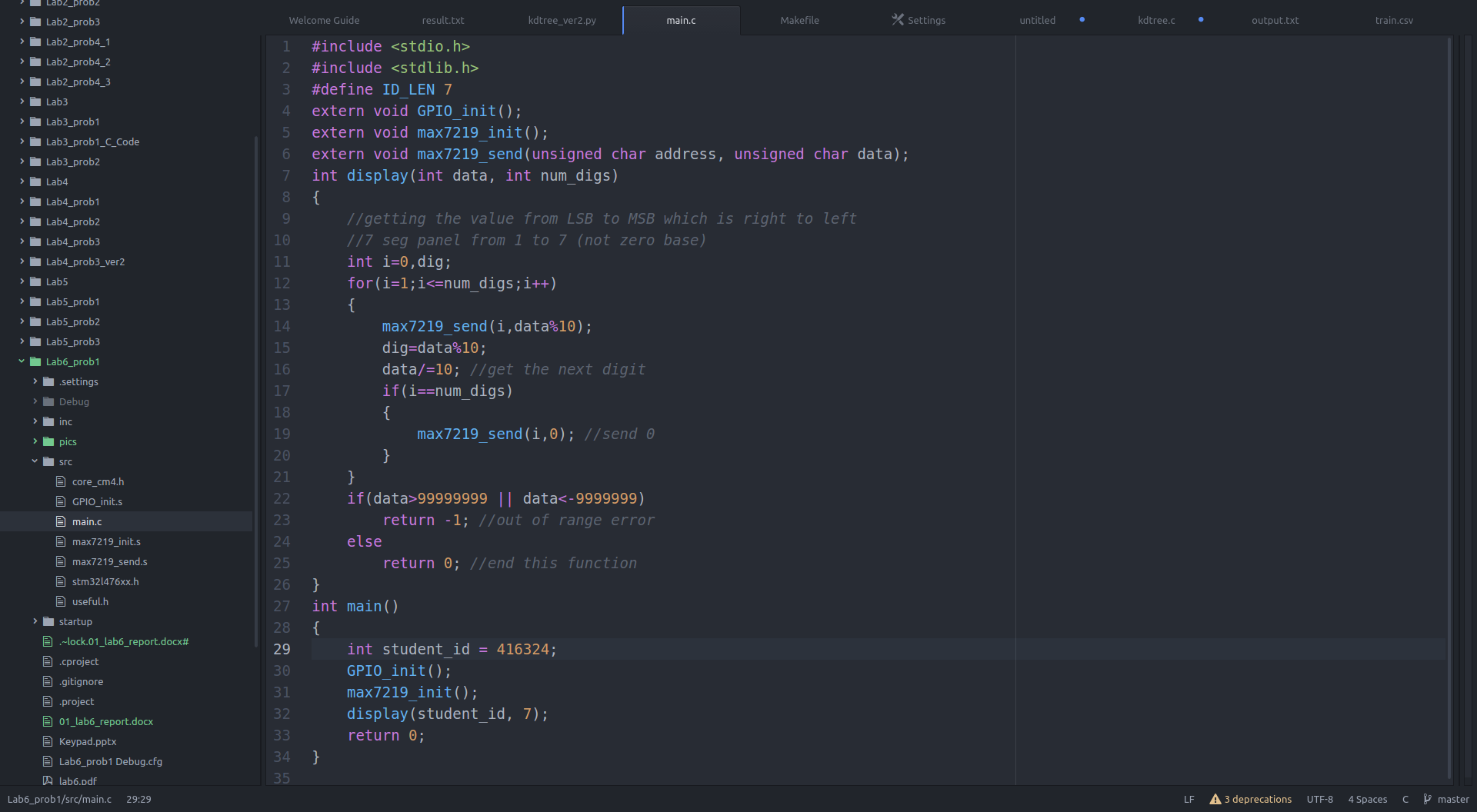
將Lab5所完成的GPIO\_init()與MAX7219\_send()改成可以被C所呼叫的版本， 並新增一個 C file 完成 display function 及利用 max7219\_send()將學號顯示於 7 段顯示器上。  
  
Modify your code in lab5.2 to make it callable by C. Add a C file to complete the code given below, display your student ID on 7-Seg LED.

|  |
| --- |
| //These functions inside the asm file  extern void GPIO\_init();  extern void max7219\_init();  extern void max7219\_send(unsigned char address, unsigned char data);    /\*\*  \* TODO: Show data on 7-seg via max7219\_send  \* Input:  \* data: decimal value  \* num\_digs: number of digits will show on 7-seg  \* Return:  \* 0: success  \* -1: illegal data range(out of 8 digits range)  \*/  int display(int data, int num\_digs)  {    }  void main() {  int student\_id = 01234567;  GPIO\_init(); max7219\_init();  display(student\_id, 8);  } |

****

Reorganize the code for using c language for STM32

Use stm32l476xx.h for all the data section we need.

Call the max7219send function to send data, since the max7219send is defined externally, so that the c program can call the assembly code, which shows a combination b/w the assembly language and the c language.

Than 416324 for the number, since 0416324 will be treated as 416324, so we have to send 0 forcibly.

## Lab6.1: Keypad Scanning

利用 4 個 input GPIO 與 4 個 output GPIO pin 連接 keypad，當按下 keypad 利 用兩顆七段顯示器顯示所對應的數字。

Note: keypad 所使用到的 GPIO 請利用 C 語言的方式初始化，各 GPIO register address 與 structure define 請參考 stm32l476xx.h

Use 4 input GPIO pins and 4 output GPIO pins to connect with keypad. Show the corresponding number of pressed button on 7-Seg LED.

Note: Use C to init GPIO used by keypad. Please refer to stm32l476xx.h for GPIO register address and structure define.

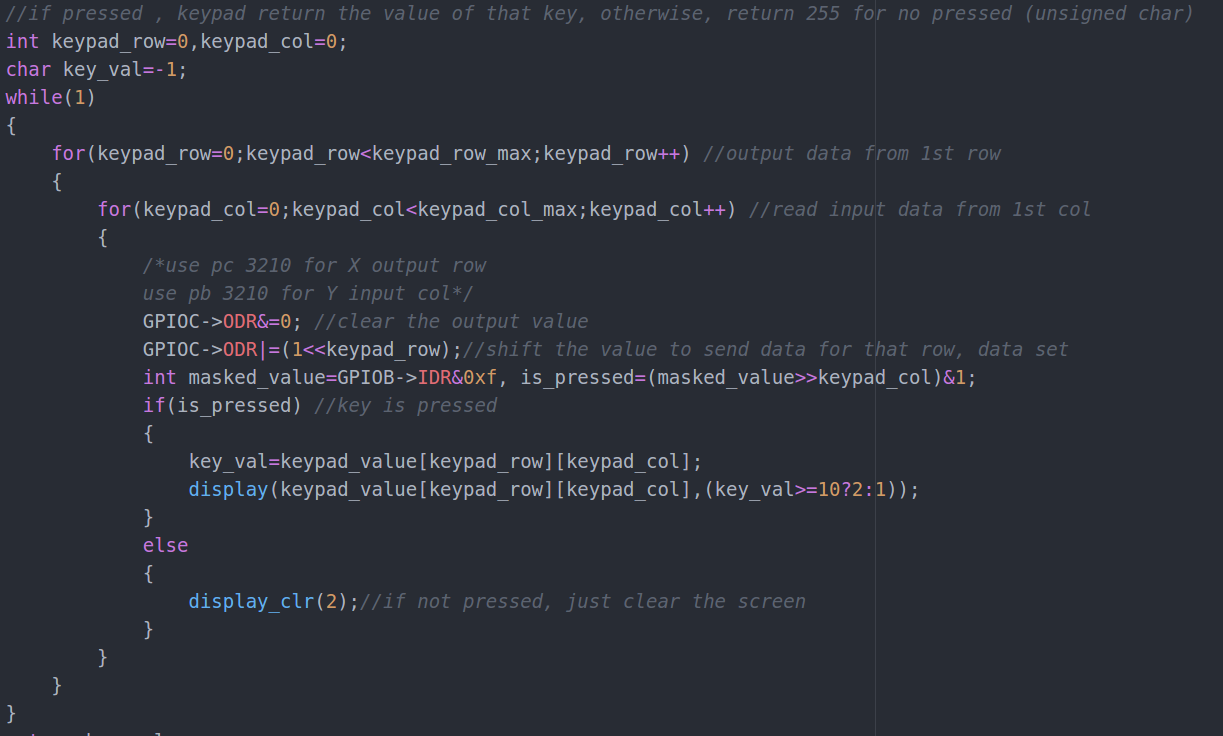
|  |
| --- |
| #include “stm32l476xx.h”  //TODO: define your gpio pin  #define X0  #define X1  #define X2  #define X3  #define Y0  #define Y1  #define Y2  #define Y3    unsigned int x\_pin[4] = {X0, X1, X2, X3};  unsigned int y\_pin[4] = {Y0, Y1, Y2, Y3};    /\* TODO: initial keypad gpio pin, X as output and Y as input \*/  void keypad\_init()  {  }    /\* TODO: scan keypad value  \* return:  \* >=0: key pressed value  \* -1: no key press  \*/  char keypad\_scan()  {  } |

各按鍵對應值為：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | X0 | X1 | X2 | X3 |
| Y0 | 1 | 2 | 3 | 10 |
| Y1 | 4 | 5 | 6 | 11 |
| Y2 | 7 | 8 | 9 | 12 |
| Y3 | 15 | 0 | 14 | 13 |

Scan the data by bitwise operation and shifting, then send the value.

Don’t forget to clear the screen when nothing is pressed.



## Lab6.2 處理單或多按鍵(30%)

利用 keypad 輸入數字並在七段顯示器顯示，各按鍵對應值為：

Show pressed button of keypad on 7-Seg LED. Each value of corresponding button is given below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | X0 | X1 | X2 | X3 |
| Y0 | 1 | 2 | 3 | 10 |
| Y1 | 4 | 5 | 6 | 11 |
| Y2 | 7 | 8 | 9 | 12 |
| Y3 | C | 0 | C | 13 |

當按多按鍵時，會將按鍵值相加並顯示出來(按 1、5、9 則顯示 15)，若準備顯 示的值>99999999，則不更動原本七段顯示器上顯示的數字，直到按下消除鍵 (C)。

When multiple buttons are pressed, show the sum of values that buttons pressed representing. If shown value is greater than 99999999, don’t modify the number showing on 7-Seg LeD until button C is pressed.

In Lab 6.2, we use FSM and hash map to implement the multi-button problem, and I will discuss both these two ways in detail later. Besides, there are some errors which I’ll also talk about at the part (3) and part (4).

1. **Hash map**

We use hash map to solve the multi-buttons problem.  
The index of the hash map means the key value of the buttons pressed. If the button was pressed, we light up the hash\_map[key\_value] to be 1.

(Using two for loop to signal from rows and receive output from columns.)

|  |
| --- |
| **if**(is\_pressed){ //key is pressed  nothing\_is\_pressed=0;  clear=0;  key\_val=keypad\_value[keypad\_row][keypad\_col];  **if**(key\_val<14){  hash\_map[key\_val]=1; //if that key is pressed, mark the hash\_map value to be 1  }**else**{  out\_sum=0;  clear=1;  }  sum=0;  **for**(**int** i=0;i<14;i++){ //clear pressed  **if**(hash\_map[i]){  sum+=i;  }  }  } |

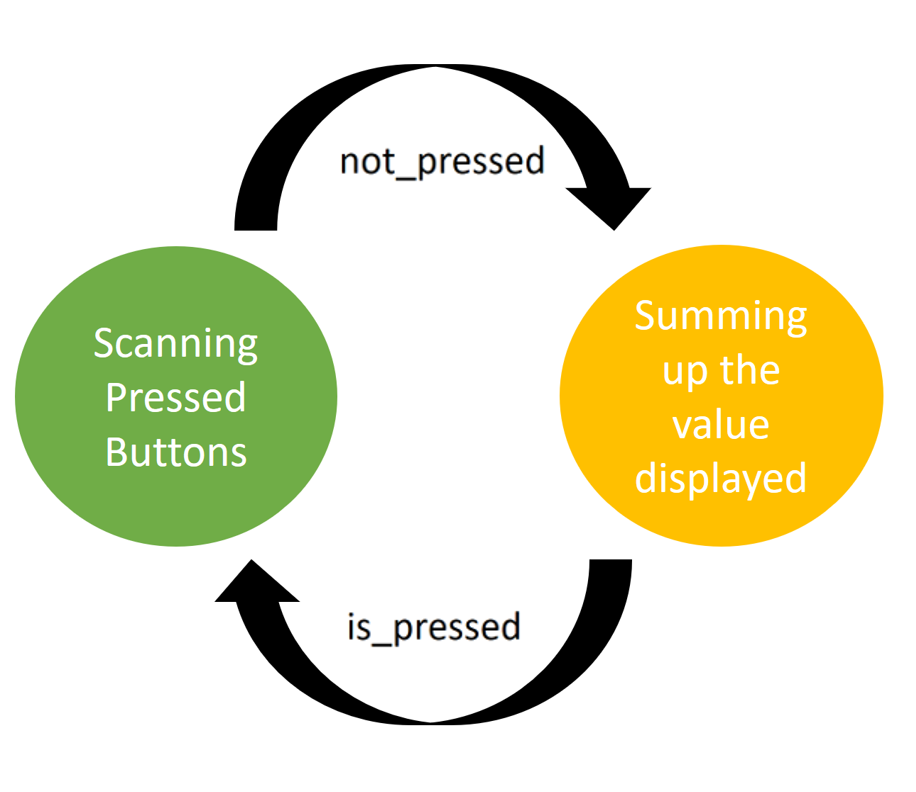
Because we put all these code in while (1), it always run into this part (is\_pressed) while the buttons are pressed.

After light up all the hash map value that had been pressed, we sum all pressed value up and store in sum. Yet if the “C” button was pressed, the out\_sum will be clean to 0 while the Boolean value of clear which used to control display will be true.

By this way, we can solve the total value of multiple buttons pressed at the same time. And, not yet, we still need to solve the problem of display value which sums all pressed value up cumulatively.

1. **Finite State Machine**

Thus, we design a Finite State Machine which contains two states for scanning pressed buttons and for summing up the value displayed.



(state 0: scanning, state 1: summing up)

Just like what I drew above. The states are controlled by whether the button is pressed. Once the buttons are all released, the FSM will get into summing up state and clean all the hash map to be 0.

|  |
| --- |
| **if**(nothing\_is\_pressed){  state=1; //if not pressed, just clear the screen  **for**(**int** i=0;i<14;i++)  hash\_map[i]=0;  } |

And in summing up state, the code is below.

|  |
| --- |
| out\_sum+=sum;  sum = 0;  state = 0; |

Adding sum to the value we’d like to display, clean sum back to zero, and also set state to scanning.

1. **The problem of blinking**

The blinking problem was quite severe, and I’ve figured out some ways to improve the situation.

I thought the reason of blinking is that we do too many codes between display, so the following are the two approach I tried.

First is that I use switch-case instead of if-else. The latter scans all lines, while the former just jump to its cases. And the second way is displaying more often, so that I put 3 displays in while(1) loop.

The blink problem still exists, but I think it’s quite better and it’s fine if we just use human’s eyes rather than recording it by camera.

1. **Weird error with bool and int**

This bug was really out of my mind. After I designed my FSM, I found out that there are only two states. Hence, I substitute bool for int to save space, and it ended in disaster. It was quite fine with int, but when it changed to bool, the keypad couldn’t work unless pressing “C” button first. I spent almost two hours on this bug, and finally found the difference in these two version of code. (I didn’t save the old version; thus, what I can use is only my weak memories.)

## BONUS Lab6.3 設計簡易計算機

寫出一個可先乘除後加減的計算機。  
輸入數值時，最多三位數字，數值範圍 1~999，若多於三位，則再輸入數字時沒反應（原本 111，再多按一個數字，7-SEG LED 依舊顯示 111 不會改變)；   
當按下運算子(+ - \* / =)時，會將原先顯示在 7-SEG LED 的數字消除掉，等待數字輸入；  
當連續按運算子 (ex:100 - - 9)答案依舊需正確算出；  
當輸入完數字和運算子按下等於後，顯示答案（7-SEG LED 答案可顯示超過三 位數和負數）；  
按下消除鍵後才開始新的運算（消除鍵無論何時按下皆會消除顯示數字，並重新 開始運算） 範例影片如下：  
<https://goo.gl/rn8srq>  
  
Design a calculator first doing multiplication and division then do addition and subtraction. Requirements are given below.   
  
Input value should be in the range of 1~999. If input value is already 3 digits, don’t give any responds to button pressed after that.   
  
When operator is pressed, clear the number shown on 7-Seg LED and wait for next number input.   
  
If operator is pressed more than 1 time, answer output should be right though.   
  
After “equal” is pressed, show the answer(negative number and number greater than 999 should be shown).   
  
Example video link is given above.   
  
各按鍵對應值為：   
Each value of corresponding button is given below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | X0 | X1 | X2 | X3 |
| Y0 | 1 | 2 | 3 | + |
| Y1 | 4 | 5 | 6 | - |
| Y2 | 7 | 8 | 9 | \* |
| Y3 | = | 0 | C | / |