第 39 組　　報告撰寫者:0316306 周煥然

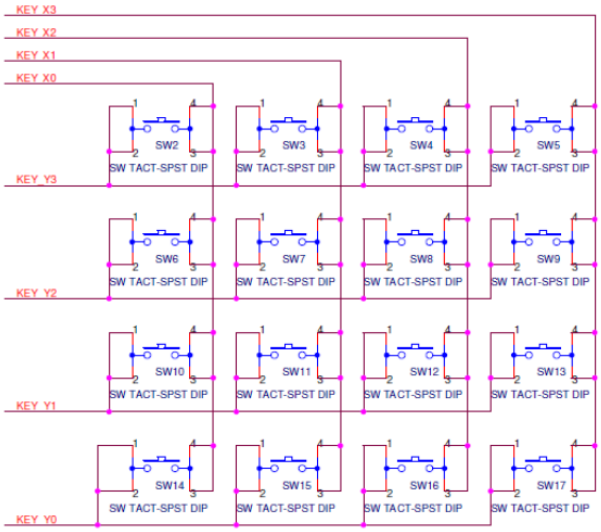
實驗名稱： 實驗六 STM32 Keypad Scanning

實驗目的：

1. 了解 STM32 使用原理
2. 了解如何使用 C code 控制 STM32
3. 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。



實驗步驟：

Lab6.0: Max7219 displayer

將 Lab5 所完成的 GPIO_init()與 MAX7219_send()改成可以被 C 所呼叫的 assembly function，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。

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 lab6.0 所實做的 display()將所對應的數字顯示在兩顆七段顯示器上，無按則不顯

示。

Lab6.2 ： 處理單鍵或多按鍵

利用keypad 輸入數字並在七段顯示器顯示，當按多按鍵時，會將按鍵值相加並顯示出來(按1、2、A 則顯示 10)，若八顆7-seg LED皆輸入滿了，則無法再輸入數字直到按下消除鍵(C)，若輸入的值會使顯示結果超出第八顆7-seg LED，則此輸入無效，直到按下消除鍵。

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，輸入數值範圍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，keypad依舊顯示111不會改變)，當按下運算子(+ - * / =)時，會將原先顯示在keypad的數字消除掉，等待數字輸入，當輸入完數字和運算符號按下等於後，顯示答案(keypad答案可顯示超過三位數和負數)，最後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)，當錯誤運算輸入順序(ex:100 - - 9 or + * 100 -9)按等於時請顯示-1。

實驗結果:

Lab6.0: Max7219 displayer

若在C中要使用assembly的function，則須在C中將function宣告為extern，而在assembly檔中將對應的function宣告為global。並在assembly中的將欲使用的register push進stack中，並在return時pop回去，如此即可在C裡面call assembly中的function。

```
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)
{
    if(num_digs > 8)
        return -1;
    else
    {
        for(int i = 0; i < num_digs; i++)
        {
            int tmp = data / 10;
            int i_tmp = data - tmp * 10;
            char c_tmp = i_tmp ;
            char add = i + 1;
            data = data / 10;
            max7219_send(add, c_tmp);
        }
    }
    return 0;
}
```

Lab6.1: Keypad Scanning

output pin 設為pull up 而input pin設為pull down。在讀時每次將三個 output pin 輸出1，另一個 output pin輸出0。而若 input pin 讀到1則表示該鍵被按下，判斷被按下為何鍵並輸出至LED上。

```
GPIOB->BSRR = X0;
GPIOB->BRR = X1;
GPIOB->BRR = X2;
GPIOB->BRR = X3;

if(GPIOB->IDR&1<<6  )
{
    display_1(1, 1);
    flag = 1;
}
if(GPIOC->IDR&1<<7 )
{
    display_1(4, 1);
    flag = 1;
}
if(GPIOA->IDR & 1<<9)
{
    display_1(7, 1);
    flag = 1;
}
if(GPIOA->IDR & 1<<8)
{
    display_1(15, 2);
    flag = 1;
}

GPIOB->BRR = X0;
GPIOB->BSRR = X1;
GPIOB->BRR = X2;
GPIOB->BRR = X3;
if(GPIOB->IDR&1<<6  )
{
    display_1(2, 1);
    flag = 1;
}
if(GPIOC->IDR&1<<7 )
{
    display_1(5, 1);
    flag = 1;
}
if(GPIOA->IDR & 1<<9)
{
    display_1(8, 1);
    flag = 1;
}
if(GPIOA->IDR & 1<<8)
{
    display_1(0, 1);
    flag = 1;
}
```

Lab6.2: 處理單鍵或多按鍵

Output pin 設為 open drain，input pin 設為 pull up。讀值時將三個 output pin 設為 1，另一個設為 0，而如果 input pin 讀到 0 時則表示該鍵被按下，使用一個變數將目前所按下的鍵值相加並顯示在 LED 上。

```
result+=1;
if(init == 1)
{
    if(result <= 9)
        shift = 1;
    else if(result <= 99)
        shift = 2;
    else
        shift = 3;
}
used_bits += shift;
if(flag2 == 1)
{
for(int i = 0; i < shift; i++)
    {
        display_result *= 10;
    }
}
display_result += result;
flag2 = 1;
init = 1;
display_2(display_result, used_bits);
```

```
result=-1;
GPIOB->BRR = X0;
GPIOB->BSRR = X1;
GPIOB->BSRR = X2;
GPIOB->BSRR = X3;

if((GPIOB->IDR&1<<6) ==0 )
{
    //display_1(1, 1);
    result += 1;
}
if((GPIOC->IDR&1<<7) ==0 )
{
    //display_1(4, 1);
    result += 4;
}
if((GPIOA->IDR & 1<<9)==0 )
{
    //display_1(7, 1);
    result += 7;
}
if((GPIOA->IDR & 1<<8) ==0)
{
    //display_1(15, 2);
    display_result = 0;
    used_bits = 0;
    shift = 0;
    flag = 0;
    init = 0;
    display_2(0, 0);
    continue;
}
GPIOB->BSRR = X0;
GPIOB->BRR = X1;
GPIOB->BSRR = X2;
GPIOB->BSRR = X3;
if((GPIOB->IDR&1<<6) ==0 )
{
    //display_1(2, 1);
    result += 2;
}
if((GPIOC->IDR&1<<7) ==0 )
{
    //display_1(5, 1);
    result += 5;
}
```

Lab6.3: 設計簡易計算機

使用兩個 stack 來實作先乘除後加減的計算機，其中一個存放數字(stack)另一個存放運算子(ostack)，每當輸入運算子的時候，就會做判斷與運算，如果此刻 ostack 最上面的運算子為乘或除，則將 stack 最上面的數與此刻 LED 上顯示的數做運算，並將結果與輸入的運算子存回 stack 中。若最上面的運算子為加或減，則判斷輸入的運算子，若為加減，則可以將 stack 最上面的兩個數字做運算並將結果與 LED 上的數字存入 stack 中，若輸入的運算子為乘除，則將 LED 上的數字及輸入之運算子存入 stack 內。因此，stack 最多只需要儲存兩個數字即可完成運算。

```
GPIO_init();
max7219_init();
keypad_init();
FPU_init();
display(0,0);
double stack[4];
int top = -1;
int ostack[3];
int otop = -1;
int state = 0;
```

```
if(ostack[otop] == 12)  //*
{
    result = stack[top--] * number;
    stack[++top] = result;
    ostack[otop] = key;
}
else if(ostack[otop] == 13)  //*
{
    result = stack[top--] / number;
    stack[++top] = result;
    ostack[otop] = key;
}
////////////////// stack[top] == + or -
else if(key == 10)    //+
{
    result = stack[top--] + number;
    stack[++top] = result;
    ostack[otop] = key;
}
else if(key == 11)   //-
{
    result = stack[top--] - number;
    stack[++top] = result;
    ostack[++otop] = key;
}
```

```
else if(key == 12)  //*
{
    stack[++top] = number;
    ostack[++otop] = key;
}
else if(key == 13)  ///
{
    stack[++top] = number;
    ostack[++otop] = key;
}
else if(key == 14)  //c
{
    top = -1;
    otop = -1;
    display(0,0);
}
```

心得討論與應用聯想：

此次實驗主要是使用 C 語言結合 assembly 來控制 STM32 板，並且學習如何讀取 keypad，使用 C 語言最大的好處是程式的可讀性提高許多，而且也可以將 code 簡化一些，適合來寫一些架構上的東西，而細部的 function 則可用 assembly 實做能有較快的執行速度。