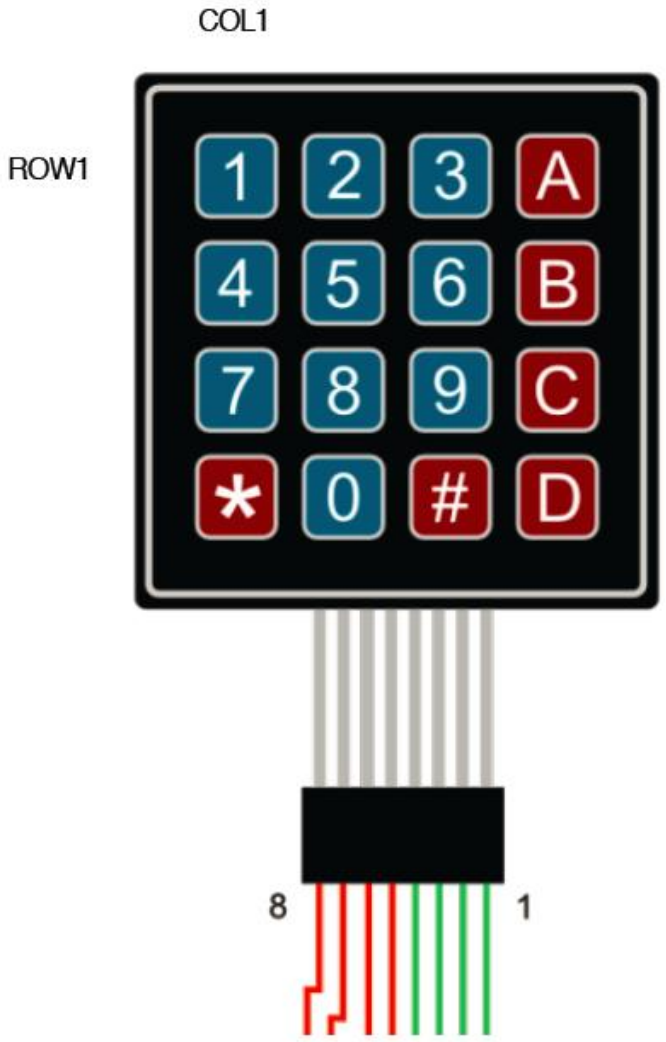
ARM GPIO Keypad Control

Part of this document is by
Prof. Shiao-Li Tsao
CS Dept., NCTU

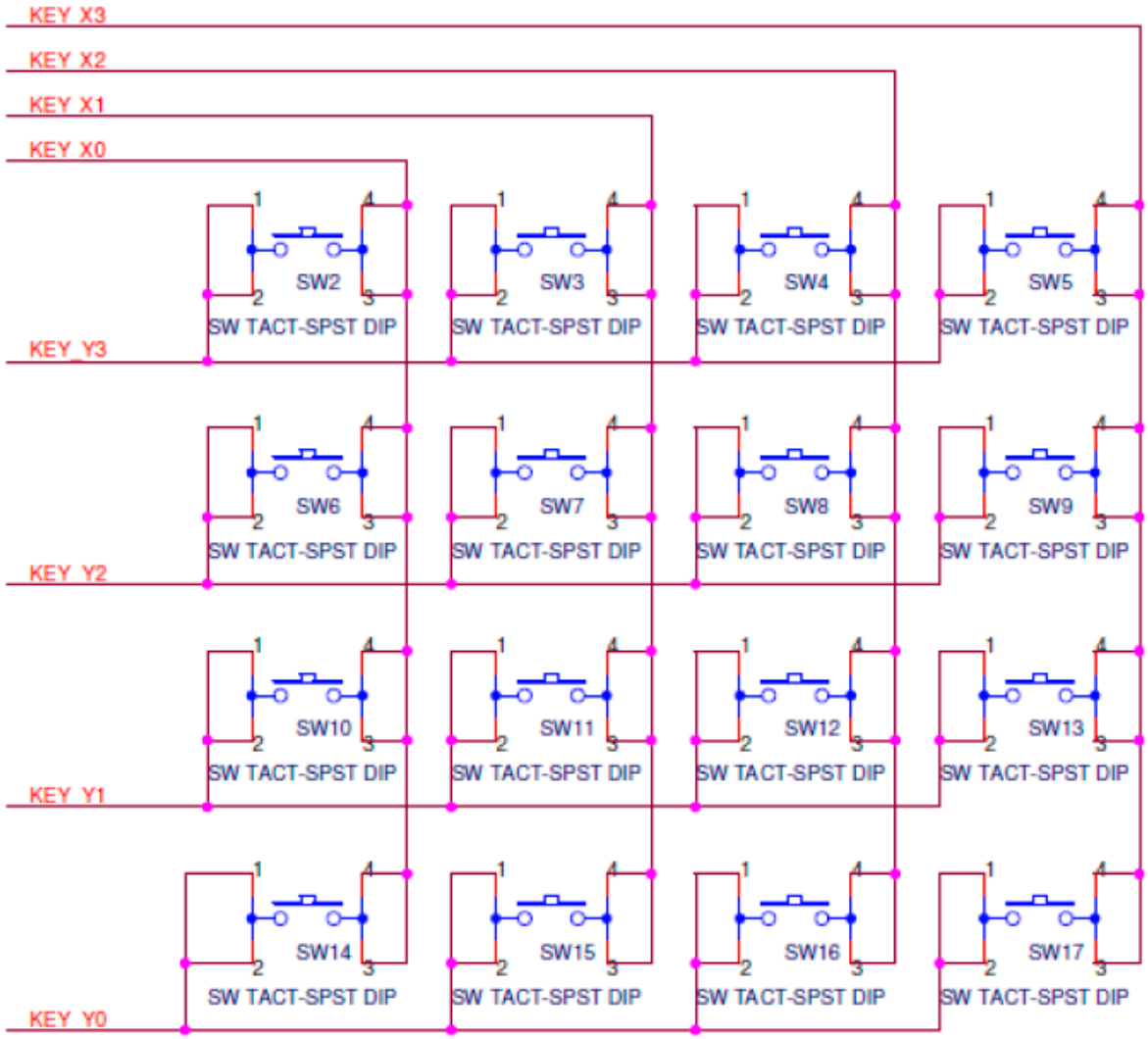
Keypad Pins



KEYPAD PINOUT:

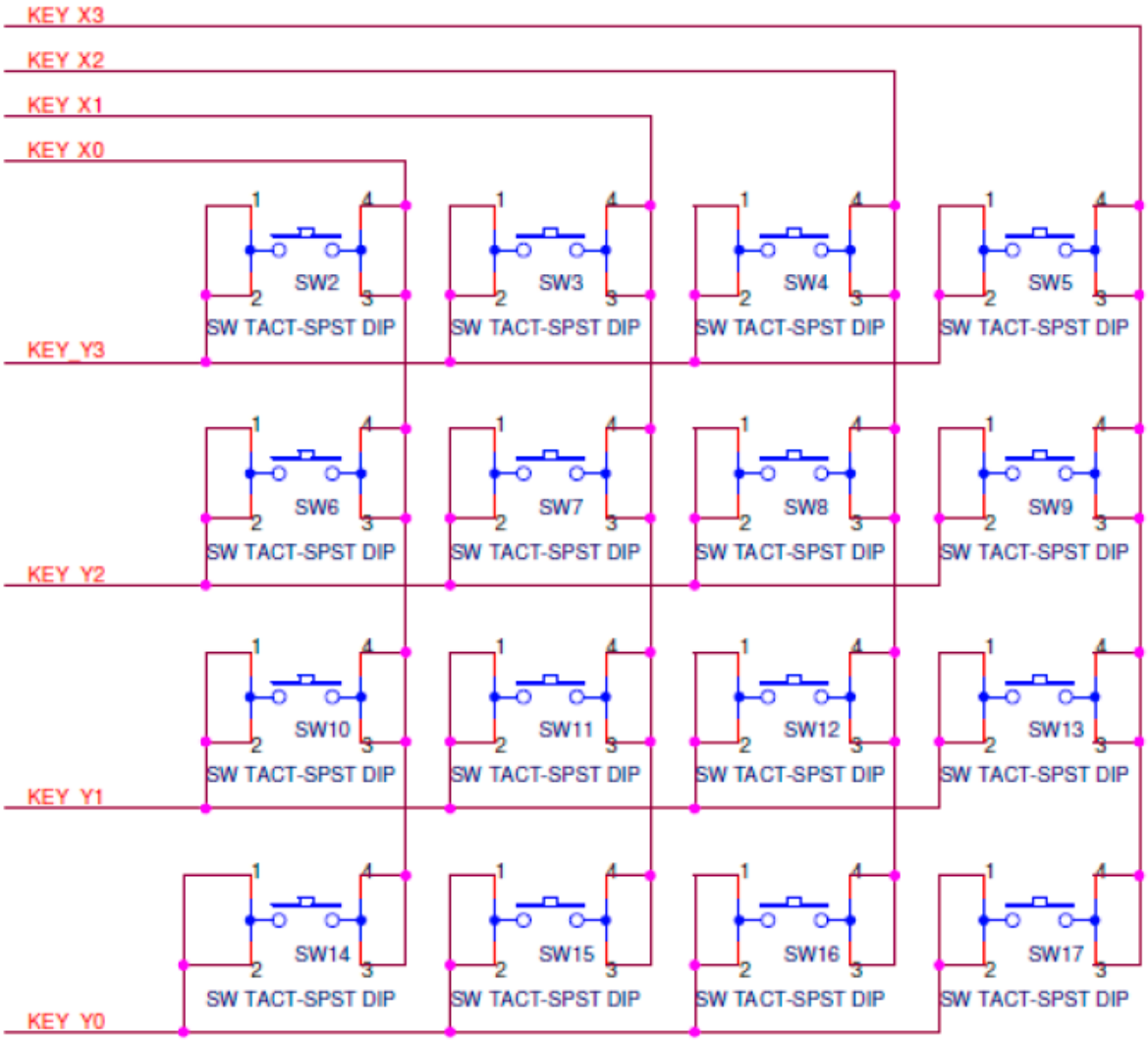
PIN 1: COL 4
PIN 2: COL 3
PIN 3: COL 2
PIN 4: COL 1
PIN 5: ROW 4
PIN 6: ROW 3
PIN 7: ROW 2
PIN 8: ROW 1

Keypad Circuits



掃描16個keypad，最少需要幾個Pin?

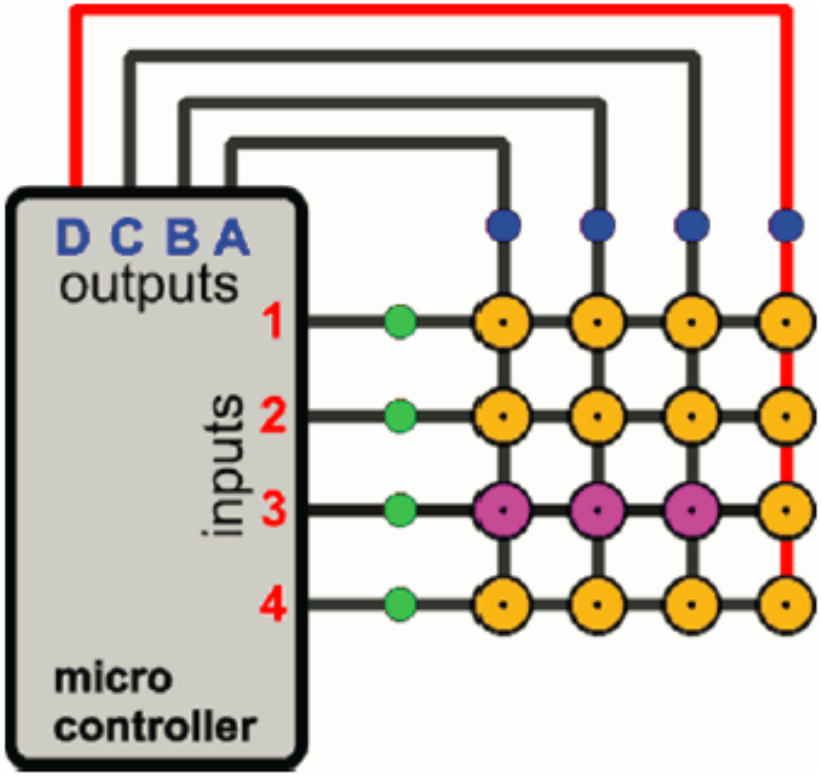
Keypad Circuits



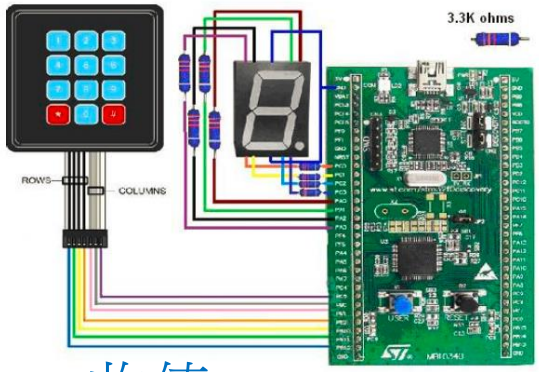
掃描16個keypad，最少需要幾個Pin?

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。

Microcontroller to Keypad

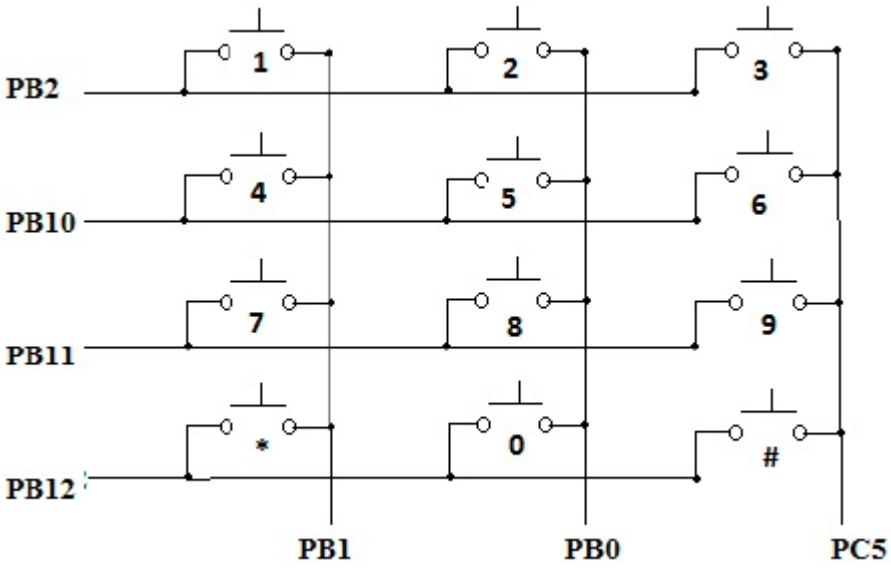


Example



送值 收值

SET PINS PC5	PB0	PB1	PRESSED KEY	OUTPUT PINS PB2	PB10	PB11	PB12
1	0	0	#	0	0	0	1
			9	0	0	1	0
			6	0	1	0	0
			3	1	0	0	0
0	1	0	0	0	0	0	1
			8	0	0	1	0
			5	0	1	0	0
			2	1	0	0	0
0	0	1	*	0	0	0	1
			7	0	0	1	0
			4	0	1	0	0
			1	1	0	0	0



Assembly Code for Keypad Control

```
int main(void)
{ initgpio();

while(1)
{   GPIOC->BSRR = GPIO_Pin_5;//set bit as high
    GPIOB->BRR = GPIO_Pin_0;//set bit as low
    GPIOB->BRR = GPIO_Pin_1;//set bit as low
{   if(GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_12))
    display(3);
    if(GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_11))
    display(6);
    if(GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_10))
    display(9);
    if(GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_2))
    display(11); }
    GPIOC->BRR = GPIO_Pin_5;//set bit as low
    GPIOB->BSRR = GPIO_Pin_0;//set bit as high
    GPIOB->BRR = GPIO_Pin_1;//set bit as low
```

送值

收值

Reference manual STM32L4x6 IO周邊register操作相關參考文件 => P267 GPIO port bit set/reset register (GPIOx_BSRR)

所以BSRR 設1時，會是1，變成high; BRR設1時，會是0，變成low

Assembly Code for Keypad Control

```
{   if(GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_12))
    display(2);
    if(GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_11))
    display(5);
    if(GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_10))
    display(8);
    if(GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_2))
    display(0);}
    GPIOC->BRR = GPIO_Pin_5;//set bit as low
    GPIOB->BRR = GPIO_Pin_0;//set bit as low
    GPIOB->BSRR = GPIO_Pin_1;//set bit ashigh
{   if(GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_12))
    display(1);
    if(GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_11))
    display(4);
    if(GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_10))
    display(7);
    if(GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_2))
    display(10);}}}
```

設定GPIOB_OTYPER

Reference *manual STM32* *P265*

7.4.2 GPIO port output type register (GPIOx_OTYPER) (x = A..H)

Address offset: 0x04

Reset value: 0x0000 0000

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.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

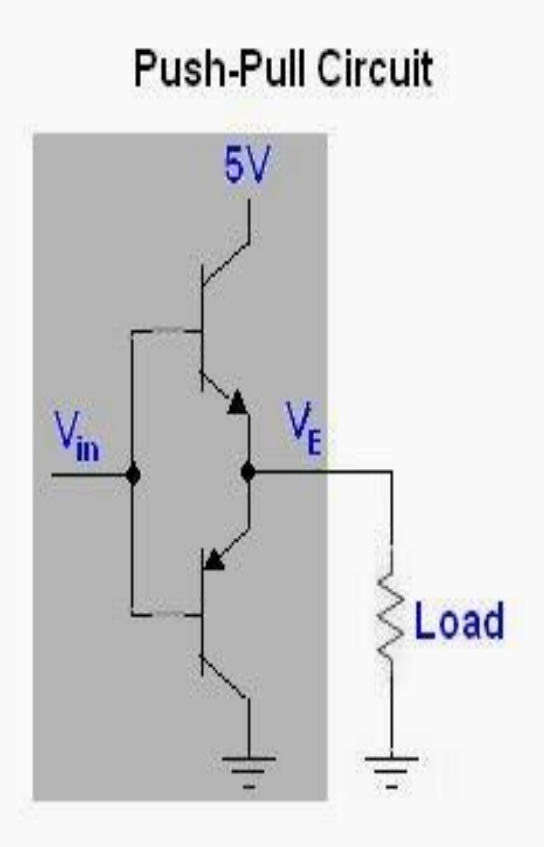
Bits 15:0 **OTy:** Port x configuration bits (y = 0..15)

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

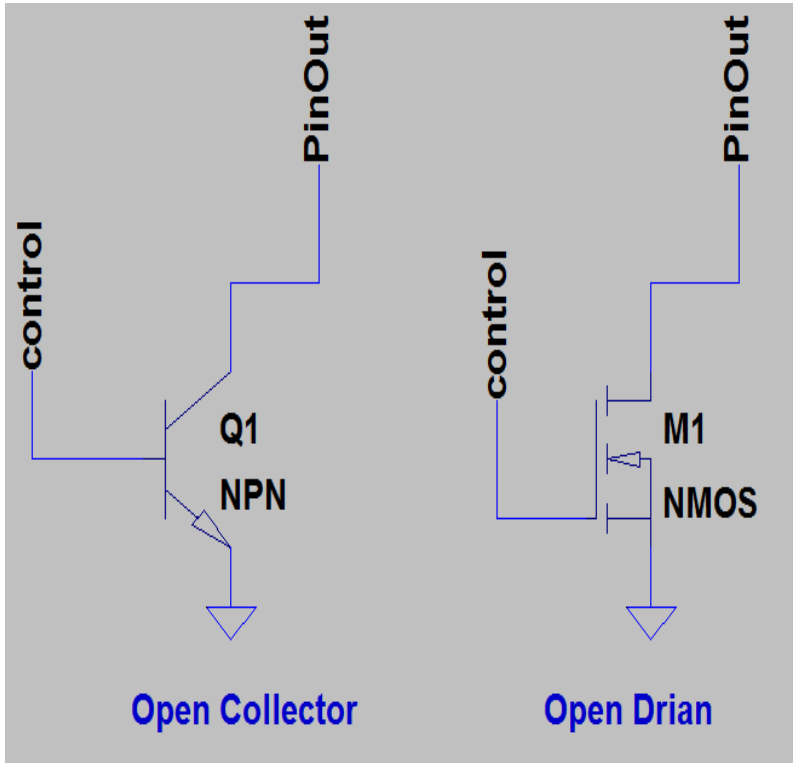
0: Output push-pull (reset state)

1: Output open-drain

Push-Pull v.s. Open-Drain



Current sink & current pull



Only current sink, no current pull

GPIO IDR

7.4.5 GPIO port input data register (GPIOx_IDR) (x = A..H)

Address offset: 0x10

P267

Reset value: 0x0000 XXXX (where X means undefined)

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.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **IDy:** Port input data bit (y = 0..15)
These bits are read-only. They contain the input value of the corresponding I/O port.

7seg.h | 7seg.c | helper_functions.c | main.c

```
1  #include "helper_functions.h"
2
3  int read_gpio(GPIO_TypeDef* gpio, int pin){
4      return (gpio->IDR >> pin) & 1;
5  }
```

程式語法

- cnt += read_gpio(ROW_gpio, ROW_pin+x);
- 同義於 cnt = cnt + read_gpio回傳值

7seg.c | helper_functions.c | main.c | keypad.c

```
3  int read_gpio(GPIO_TypeDef* gpio, int pin){
4      return (gpio->IDR >> pin) & 1;
5  }
```

Lab4 KeyPad
範例程式

Code provided by NCTU CS 謝明恩 吳赫倫
Slide made by NCTU ME 助教 林穎毅
20170412

Lab4.1 Single Press

- 利用4個input GPIO與4個output GPIO pin連接keypad，當按下keypad利用Max7219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。
- 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

Lab4.1 Single Press C Code

```
#include "stm321476xx.h"
//TODO: define your gpio pin
#define KEY_PORT GPIOX
#define X0 GPIO_PIN_X
#define X1
#define X2
#define X3
#define Y0
#define Y1
#define Y2
#define Y3

unsigned int x_pin = {X0, X1, X2, X3};
unsigned int y_pin = {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()
{
}
```

Lab4.1 Single Press C Code

```
int main()
{       GPIO_init();
        max7219_init();
        keypad_init();
        while(1){
             flag_keypad=GPIOB->IDR&10111<<5;
                        if(flag_keypad!=0){
                                   k=45000;
                        while(k!=0){
                             flag_debounce=GPIOB->IDR&10111<<5;
                             k--;
                        }
           if(flag_debounce!=0){
                  for(int i=0;i<4;i++){ //scan keypad from first column
                      position_c=i+8;
                      if(i==3)position_c++;
                 //set PA8,9,10,12(column) low and set pin high  from PA8
                  GPIOA->ODR=(GPIOA->ODR&0xFFFFE8FF)|1<<position_c;
                   for(int j=0;j<4;j++){ //read input from first row
                 position_r=j+5;
                  if(j==3) position_r++;
                   flag_keypad_r=GPIOB->IDR&1<<position_r;
        if(flag_keypad_r!=0)display(Table[j][i]);
                    }                          }             }
```

Lab4.1 Single Press C Code

```
void keypad_init()
{
      // SET keypad gpio OUTPUT //
            RCC->AHB2ENR = RCC->AHB2ENR|0x2;
           //Set PA8,9,10,12 as output mode
            GPIOA->MODER= GPIOA->MODER&0xFDD5FFFF;
           //set PA8,9,10,12 is Pull-up output
            GPIOA->PUPDR=GPIOA->PUPDR|0x1150000;
           //Set PA8,9,10,12 as medium speed mode
            GPIOA->OSPEEDR=GPIOA->OSPEEDR|0x1150000;
           //Set PA8,9,10,12 as high
            GPIOA->ODR=GPIOA->ODR|10111<<8;

      // SET keypad gpio INPUT //
           //Set PB5,6,7,9 as INPUT mode
            GPIOB->MODER=GPIOB->MODER&0xFFF303FF;
           //set PB5,6,7,9 is Pull-down input
            GPIOB->PUPDR=GPIOB->PUPDR|0x8A800;
           //Set PB5,6,7,9 as medium speed mode
            GPIOB->OSPEEDR=GPIOB->OSPEEDR|0x45400;
}
```

Lab4.1 Single Press

3.2.1. 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

Lab4.2 Calculator HW

- 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)(15%)，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(5%)，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)(10%)，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)(5%)，當錯誤運算輸入順序(ex:100 - - 9 or + * 100 -9)按等於時請顯示-1 (5%)
- 3.4.1. 各按鍵對應值為：

3.4.1. 各按鍵對應值為：

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

Lab4.3 Multiple press- bonus

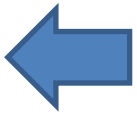
- 當按多按鍵時，會將按鍵值相加並顯示出來(按1、2、A 則顯示 13)，若八顆7-seg LED皆輸入滿了，則無法再輸入數字直到按下消除鍵(C)，若輸入的值會使顯示結果超出第八顆7-seg LED，則此輸入無效，直到按下消除建，範例影片如下：
  - https://goo.gl/HBdaXH
- 補充:
  - 如果按鍵按下後立刻放開,則顯示一次(短按)
  - 若按鍵按下沒立刻放開,則須連續顯示(長按)
  - 記得將非輸出1的pin 腳設成高組抗,避免掃描時發生偵測不到pin腳的狀況

Lab4.3 Multiple press- bonus

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

main function

```
#include "stm32l476xx.h"
#include "helper_functions.h"
#include "7seg.h"
#include "keypad.h"

  // Define pins for 7seg
  #define SEG_gpio GPIOC
  #define DIN_pin 3
  #define CS_pin 4
  #define CLK_pin 5

// Define pins for keypad
#define COL_gpio GPIOA
#define COL_pin 5       // 5 6 7 8
#define ROW_gpio GPIOB
#define ROW_pin 3       // 3 4 5 6
```



Include .h檔
跟設定pin角

main function

```
int main(){

//#ifdef lab_keypad_single_key

    if(init_7seg(SEG_gpio, DIN_pin, CS_pin, CLK_pin) != 0){
        // Fail to init 7seg
        return -1;
    }

    // Set Decode Mode to Code B decode mode
    send_7seg(SEG_gpio, DIN_pin, CS_pin, CLK_pin, SEG_ADDRESS_DECODE_MODE, 0xFF);
    // Set Scan Limit to all digits
    send_7seg(SEG_gpio, DIN_pin, CS_pin, CLK_pin, SEG_ADDRESS_SCAN_LIMIT, 0x07);
    // Wakeup 7seg
    send_7seg(SEG_gpio, DIN_pin, CS_pin, CLK_pin, SEG_ADDRESS_SHUTDOWN, 0x01);

    if(init_keypad(ROW_gpio, COL_gpio, ROW_pin, COL_pin) != 0){
        // Fail to init keypad
        return -1;
    }
```

起始參數設定

main function

```
while(1){
    int input = 0;
    for(int i=0;i<4;i++){
        for(int j=0;j<4;j++){
            if(check_keypad_input_one(ROW_gpio, COL_gpio, ROW_pin, COL_pin, i, j)){
                input = 1;
                display_number(SEG_gpio, DIN_pin, CS_pin, CLK_pin, keypad[i][j], num_digits(keypad[i][j]));
            }
        }
    }
    if(input == 0){
        display_number(SEG_gpio, DIN_pin, CS_pin, CLK_pin, 0, 0);
    }
}

    return 0;
}
```

根據16宮格讀取資料

check_keypad_input_one

```
int check_keypad_input_one(GPIO_TypeDef* ROW_gpio, GPIO_TypeDef* COL_gpio, int ROW_pin, int COL_pin, int x, int y){
    int cycles = 400;
    // Set Column to push-pull mode
    COL_gpio->OTYPER &= ~(1 << (COL_pin+y));
    // Count the total number of time it is pressed in a certain period
    int cnt = 0;
    for(int a=0;a<cycles;a++){
        cnt += read_gpio(ROW_gpio, ROW_pin+x);
    }
    // Set Column back to open drain mode
    COL_gpio->OTYPER |= (1 << (COL_pin+y));
    // return if the key is pressed(1) or not(0)
    return (cnt > (cycles*0.7));
}
```

把y col設為push-pull(0), 這樣我們才能提供電壓讀取那col裡的數字. 其他col都維持原樣。
ex: if y= 1, then OTYPER = 1101

讀取此y col中x row的值

把y col設為open drain(1), 等同關上. 其他col都維持原樣。
ex: if y= 1, then OTYPER = 0010

init_keypad

```
int init_keypad(GPIO_TypeDef* ROW_gpio, GPIO_TypeDef* COL_gpio, int ROW_pin, int COL_pin){
    // Enable AHB2 Clock
    if(ROW_gpio==GPIOA || COL_gpio==GPIOA){
        RCC->AHB2ENR |= RCC_AHB2ENR_GPIOAEN;
    }
    if(ROW_gpio==GPIOB || COL_gpio==GPIOB){
        RCC->AHB2ENR |= RCC_AHB2ENR_GPIOBEN;
    }
```

init_keypad

```
// First Clear bits(&) then set bits(|)
for(int a=0;a<4;a++){
    // Set GPIO pins to output mode (01)
    COL_gpio->MODER &= ~(0b11 << (2*(COL_pin+a)));
    COL_gpio->MODER |= (0b01 << (2*(COL_pin+a)));
    // Set GPIO pins to very high speed mode (11)
    COL_gpio->OSPEEDR &= ~(0b11 << (2*(COL_pin+a)));
    COL_gpio->OSPEEDR |= (0b11 << (2*(COL_pin+a)));
    // Set GPIO pins to open drain mode (1)
    COL_gpio->OTYPER &= ~(0b1 << (COL_pin+a));
    COL_gpio->OTYPER |= (0b1 << (COL_pin+a));
    // Set Output to high
    set_gpio(COL_gpio, COL_pin+a);
}
```

Port A 5 6 7 8設為output、high speed and open drain

init_keypad

```
    // First Clear bits(&) then set bits(|)
    for(int a=0;a<4;a++){
        // Set GPIO pins to input mode (00)
        ROW_gpio->MODER &= ~(0b11 << (2*(ROW_pin+a)));
        ROW_gpio->MODER |= (0b00 << (2*(ROW_pin+a)));
        // Set GPIO pins to Pull-Down mode (10)
        ROW_gpio->PUPDR &= ~(0b11 << (2*(ROW_pin+a)));
        ROW_gpio->PUPDR |= (0b10 << (2*(ROW_pin+a)));
    }

    return 0;
}
```

把GPIOB 3 4 5 6設為
input and pull-down

Keypad矩陣

```
const int keypad[4][4] = {
    {1, 2, 3, 10},
    {4, 5, 6, 11},
    {7, 8, 9, 12},
    {15, 0, 14, 13}
};
```

num_digits

```
int num_digits(int x){
    if(x == 0){
        return 1;
    }
    int res = 0;
    while(x){
        res++;
        x /= 10;
    }
    return res;
}
```

決定輸入數字的位數

init_7seg

```
int init_7seg(GPIO_TypeDef* gpio, int DIN, int CS, int CLK){
    // Enable AHB2 Clock
    if(gpio==GPIOA){
        RCC->AHB2ENR |= RCC_AHB2ENR_GPIOAEN;
    }
    else if(gpio==GPIOB){
        RCC->AHB2ENR |= RCC_AHB2ENR_GPIOBEN;
    }
    else if(gpio==GPIOC){
        RCC->AHB2ENR |= RCC_AHB2ENR_GPIOCEN;
    }
    else{
        // Error! Add other cases to suit other GPIO pins
        return -1;
    }
```

init_7seg

```
    // Set GPIO pins to output mode (01)
    // First Clear bits(&) then set bits(|)
    gpio->MODER &= ~(0b11 << (2*DIN));
    gpio->MODER |= (0b01 << (2*DIN));
    gpio->MODER &= ~(0b11 << (2*CS));
    gpio->MODER |= (0b01 << (2*CS));
    gpio->MODER &= ~(0b11 << (2*CLK));
    gpio->MODER |= (0b01 << (2*CLK));

    // Close display test
    send_7seg(gpio, DIN, CS, CLK, SEG_ADDRESS_DISPLAY_TEST, 0x00);

    return 0;
}
```

send_7seg

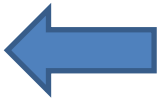
```
void send_7seg(GPIO_TypeDef* gpio, int DIN, int CS, int CLK, int address, int data){
    // The payload to send
    int payload = ((address&0xFF)<<8)|(data&0xFF);

    // Start the sending cycles
    // 16 data-bits + 1 CS signal
    int total_cycles = 16+1;

  for(int a=1;a<=total_cycles;a++){
        // Reset CLK when enter
        reset_gpio(gpio, CLK);

        // Set DIN according to data except for last cycle(CS)
        if(((payload>>(total_cycles-1-a))&0x1) && a!=total_cycles)
            set_gpio(gpio, DIN);
        }
        else{
            reset_gpio(gpio, DIN);
        }
```

send_7seg

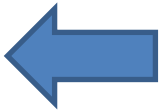
```
        // Set CS at last cycle
        if(a==total_cycles){
            set_gpio(gpio, CS);
        }
        else{
            reset_gpio(gpio, CS);
        }

        // Set CLK when leaving (7seg set data at rising edge)
        set_gpio(gpio, CLK);
    }

    return;
}
```

display_number

```
int display_number(GPIO_TypeDef* gpio, int DIN, int CS, int CLK, int num, int num_digs){
    for(int i=1;i<=num_digs;i++){
        send_7seg(gpio, DIN, CS, CLK, i, num % 10);
        num /= 10;
    }
    for(int i=num_digs+1;i<=8;i++){
        num /= 10;
        send_7seg(gpio, DIN, CS, CLK, i, 15);//blank
    }
    if(num != 0)
        return -1;
    return 0;
}
```



輸出我們要的數值



剩下的位數空白