鍵盘

先決定好掃描橫列或直行

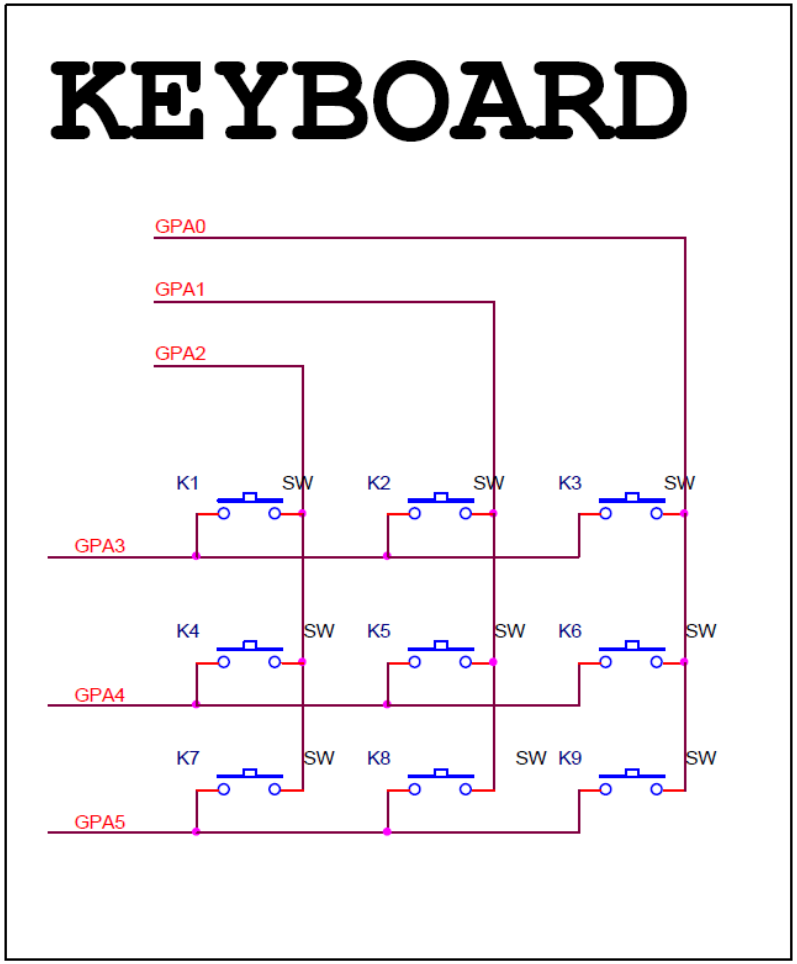
以掃描直行為例

- Column control : GPA2, 1, 0
- 寫入內容:設定要掃描的直行的位元為0 (同時只能有一個bit為0)

- Row control : GPA 3, 4, 5
  - 讀取內容:當相對應之bit為0時，代表該位置之按鍵被壓按

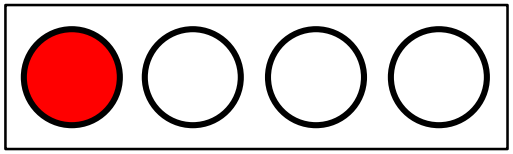
- 當相對應的兩個bits都是0時，代表該按鍵被選取

Key1 (K1)= GPA3 + GPA2
Key2 (K2)= GPA3 + GPA1
Key3 (K3)= GPA3 + GPA0
Key4 (K4)= GPA4 + GPA2
Key5 (K5)= GPA4 + GPA1
Key6 (K6)= GPA4 + GPA0
Key7 (K7)= GPA5 + GPA2
Key8 (K8)= GPA5 + GPA1
Key9 (K9)= GPA5 + GPA0

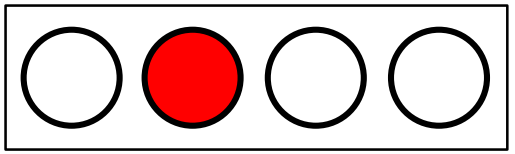


舉例

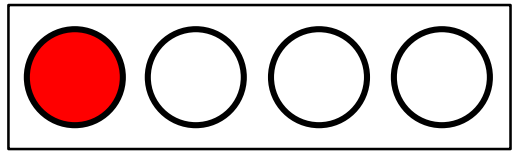
- 初始狀態 單色LED燈 最左邊燈是亮的

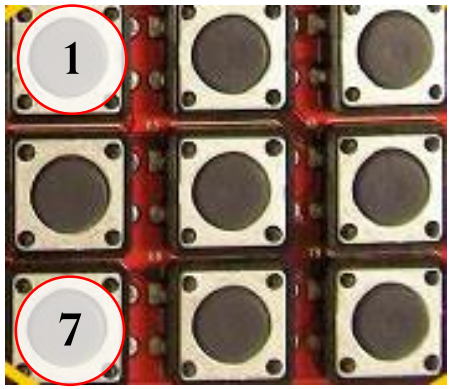


- 按1號鍵，燈號往右移(以此類推)



- 按7號鍵，燈號往左移(以此類推)





```
int32_t main (void)
{
        GPIO_T * tGPIO_A,*tGPIO_C;
        uint16_t act[3]={0xfffb,0xfffd,0xfffe};                 //鍵盤: 指定第一行 第二行 第三行掃描
        uint16_t led[4] = {0xefff,0xdfff,0xbfff,0x7fff};        //選取四顆LED GPC12~15
        uint16_t i = 0;
        uint32_t u32Reg;
        uint32_t u32Reg_temp;

        tGPIO_C = (GPIO_T *)((uint32_t)GPIOA + (2*0x40));
        tGPIO_A = (GPIO_T *)((uint32_t)GPIOA + (0*0x40));
        tGPIO_C->DOUT = led[i];                                 //寫入暫存器時須用DOUT欄位
        u32Reg = (uint32_t)&GPIOA->PIN + (0*0x40);              //讀取暫存器時須用PIN欄位
                                                //  將暫存器位址 存到變數u32Reg中   (0*0x40)代表GPA
```

```
        while(1)
        {
                tGPIO_A->DOUT = act[0];          // 掃描第一行
                u32Reg_temp=inpw(u32Reg);        // 用inpw讀取暫存器內容
                if((u32Reg_temp & 0x8) == 0)      //檢查第一個鍵是否觸發 bit 3 若是 往左一個燈號
                {
                        i=(i+1)%4;
                        tGPIO_C->DOUT = led[i];
                }
                if((u32Reg_temp & 0x20) ==0)      //檢查第七個鍵是否觸發  bit 5 若是 往右一個燈號
                {
                        i=(i-1);
                        i=i%4;
                        tGPIO_C->DOUT = led[i];
                }
                 DrvSYS_Delay(100000); //防彈跳
        }
}
```

補充 - Bit Operation

AND: &

c = a & b :

```
  a = 0x00101001
& b = 0x10001100
  c = 0x00001000
```

OR: |

c = a | b :

```
  a = 0x00101001
| b = 0x10001100
  c = 0x10101101
```

XOR: ^

c = a ^ b :

```
  a = 0x00101001
^ b = 0x10001100
  c = 0x10100101
```

NOT: ~

c = ~a :

```
~ a = 0x00101001
  c = 0x11010110
```

Left shift: <<

c = a << 2:

```
  a = 0x00101001
  c = 0x10100100
```

Right shift: >>

Unsigned int: 0

c = a >> 2:

```
  a = 0x10010100
  c = 0x00100101
```

Right shift: >>

signed int: signed bit

c = a >> 2:

```
  a = 0x10010100
  c = 0x11100101
```

- 問題:偵測X中第P個位元是0或1
  - X = 01001101
  - P : 所要處理的位元, 0 <= P <=7
- 方法:

```
M=0x1;
B=  X & ( M <<  P);
if( B == 0 ) Ans = 0;
else  Ans = 1;
```

- 舉例:

偵測X中第3個位元是0或1
X = 0x01001101
M = 0x00000001
B = X & (M << 3)

Step1. M = (M << 3) = 0x00001000

Step2. B = X & M

```
  X = 0x01001101
& ( M << 3 ) = 0x00001000
-----------------------
  B = 0x00001000
```

- 問題:將X中第P個位元設定成0或1
  - X = 01001101
  - P : 所要處理的位元, 0 <= P <=7
- 方法:

  M=0x1;

  Case1: B = X & ~(M << P); 設定成0

  Case2: B = X | (M << P);設定成1
  - 舉例:

將X的第三個bit變成0
X = 0x01001101
M = 0x00000001
B = X & ~(M << 3)

Step1. M = (M << 3) = 0x00001000

Step2. M = ~M = 0x11111011

Step3. B = X & M

X = 0x01001101
& ~(M << 3) = 0x11111011
B = 0x01001001

- 問題:將X中第P個位元0變1或1變0
  - X = 01001101
  - P: 所要處理的位元, 0 <= P <=7
- 方法:

  M=0x1;

  B = X ^ (M << P);
  - 舉例:

將X的第三個bit變成0

X = 0x01001101

M = 0x00000001

B = X ^ (M << 3)

Step1. M = (M << 3) = 0x00001000

Step2. B = X ^ M

X = 0x01001101

^ (M << 3) = 0x00001000

B = 0x01001001