矩陣式鍵盤與信號 signal 處理

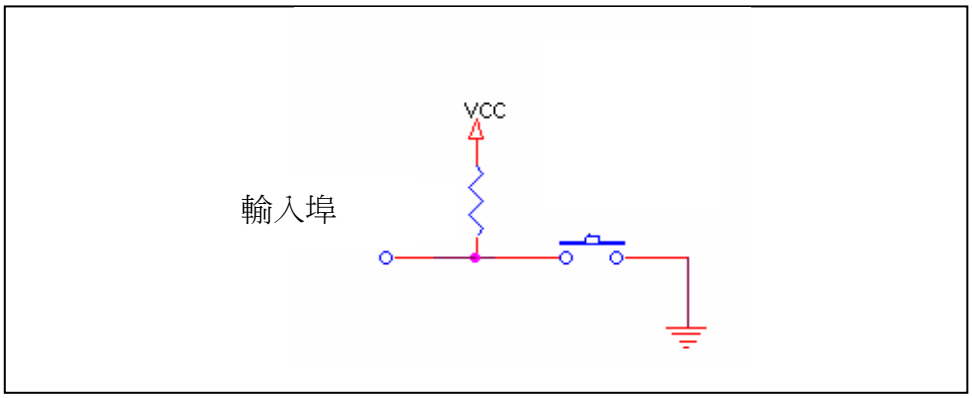
(uClinux-5)

在 uClinux 及信號處理一文中介紹信號的處理方式及實例，但是有一個重大的缺陷，就是許多信號不能重複產生，程式設計過程中，常常需要定時產生信號，以時輪詢一些設備該，這樣的程式在 uClinuxu 作業系統下要如何設計呢？本章將以 ESD44B0_B 並配合 ESDLAB 實驗版，實作設計讀入鍵盤輸入之程式。

一個基本的鍵盤輸入設備，在實用系統內通常是不可缺少的，透過鍵盤的輸入，矩陣式的按鍵輸入為一常用的設計方式。

一、 基本按鍵之電路原理

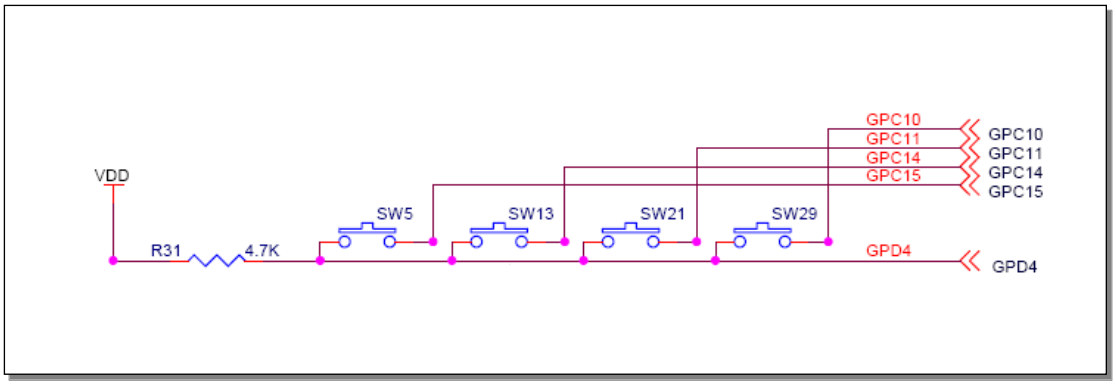
處理器透過讀入輸入設備的狀態，來判斷按鍵被按與否，電路圖如下圖所示：



如果按鍵未被按下，輸入埠的狀態為高電位，此時處理器將會讀到高電位，若按鍵被按下，則輸入埠的狀態為低電位，處理器即會讀到低電位。

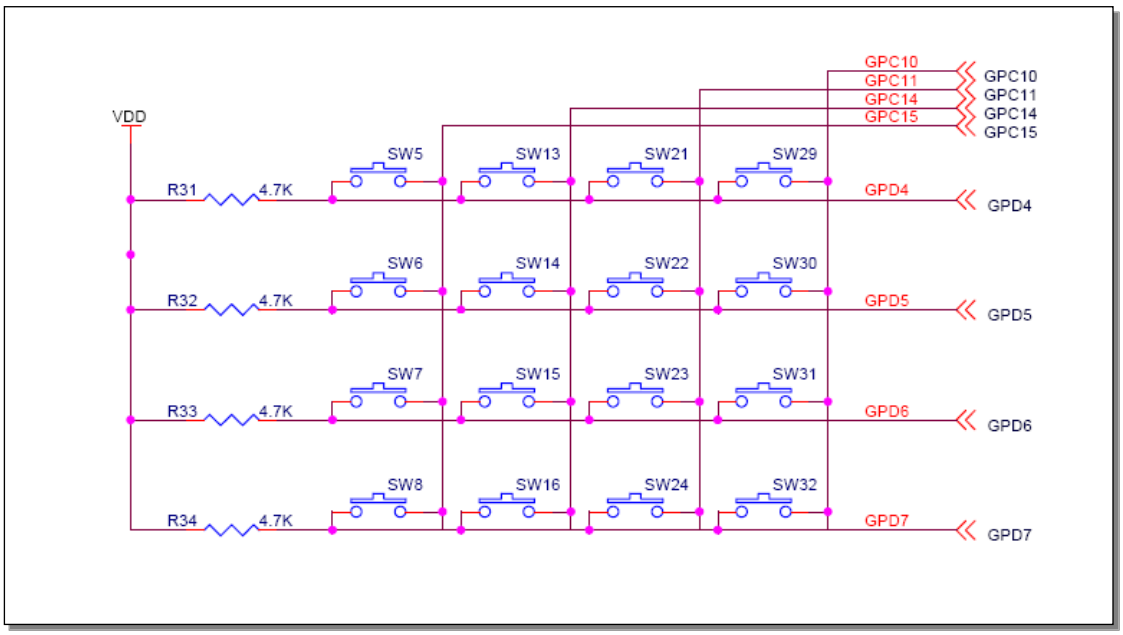
二、 矩陣鍵盤原理

如果我們有 100 組輸入埠，即可用上述之電路來設計鍵盤，但是，實際上來說是有困難的，因為我們不太可能設計一個具有 100 組的輸入埠的設備，除了浪費資源外，空間也是一個很大的考量，當然以後擴充也都是問題，所以矩陣式的鍵盤就是由此需要而產生，基本原理，從下圖開始說明：



- 上圖顯示之 GPC10、11、14、15 為輸出設備，處理器透過此四組輸出埠，控制輸入埠 GPD4 讀入 SW5、SW13、SW21 及 SW29 按鍵之狀態。
- GPD4 為輸入埠，當處理器利用所對應按鍵之控制埠(GPC10、11、14、15)，輸出低電壓後，即可讀入 GPD4 埠之內容，這些輸出埠我們可以稱之為按鍵開始之控制訊號，當為低電壓時，可讀入所對應按鍵之訊號。

或許讀者會覺得奇怪，為甚麼要多加輸出訊號，在控制埠(GPC10、11、14、15)處直接接低電位不是一樣嗎?以電路來說是相同的，但是我們所要做的是再加三組一樣的電路，做成 4X4 的矩陣按鍵，如下圖所示：



- 由上圖來看，按鍵可分為以下四組：
  - GPD4，讀入 SW5、SW13、SW21 及 SW29 按鍵之狀態。
  - GPD5，讀入 SW6、SW14、SW22 及 SW30 按鍵之狀態。
  - GPD6，讀入 SW7、SW15、SW23 及 SW31 按鍵之狀態。
  - GPD7，讀入 SW8、SW16、SW24 及 SW32 按鍵之狀態。
- GPD4、5、6、7 讀入時代表四個按鍵的那一個按鍵，決定於 GP10、11、14 及 15 之輸出狀態，以下說明之：
  - GPC15 為低電位時，讀入 GPD4、5、6、7 分別代表 SW5、6、7、8 按鍵

之狀態。

- GPC14 為低電位時，讀入 GPD4、5、6、7 分別代表 SW13、14、15、16 按鍵之狀態。
- GPC11 為低電位時，讀入 GPD4、5、6、7 分別代表 SW21、22、23、24 按鍵之狀態。
- GPC10 為低電位時，讀入 GPD4、5、6、7 分別代表 SW29、30、31、32 按鍵之狀態。

- 當讀入之狀態為低電位時，代表所對應之按鍵被按下，高電位則未按下。
- GPC10、GPC11、GPC14 及 GPC15 不可同時輸出為低電位，如果同時為低電位時，所讀入的狀態將無法判斷是那一組按鍵。
  - 若 GPC10、GPC11 同時為低電位時，當讀入 GPD4 時，無法分辨其狀態是反應 SW21 或 SW29 按鍵之狀態。
- 纂寫程式的原理為，依照順序分別將輸出埠 GPC10、GPC11、GPC14 及 GPC15 的狀態設定為低電位，當輸出埠為低電位時，讀取輸入埠 GPC4、GPD5、GPD6 及 GPD7 之狀態，此時狀態所代表之意義為所對應之按鍵狀態。

三、 矩陣鍵盤實例

S3C44B0X 處理器的輸出入埠為一多用途之輸出入接腳，這些接腳可當成輸入、輸出或是位址等接腳使用，這些接腳透過輸出入埠暫存器決定其用途，有關輸出入埠暫存器之設定及其詳細欄位內容，請參照 S3C44B0X 的使用手冊，以下分別列出初始化的步驟：

- 將 GPC10、11、14、15 接腳設定為輸出，內容如下：

```
#define MATRIX_CONTROL_PORT_INIT    CSR_WRITE(rPCONC,((CSR_READ(rPCONC) & 0x0f0fffff) | 0x5f5fffff))
```

暫存器 PCONC 之 Bit[21:20]、[23:22]、[29:28]、[30:31]為 01 分別設定 GPC10、11、14、15 為輸出接腳

- 將 GPD4、5、6、7 接腳設定為輸入，內容如下：

#define MATRIX_READ_PORT_INIT CSR_WRITE(rPCOND,(CSR_READ(rPCOND) & 0xff))

暫存器 PCOND 之 Bit[9:8]、[11:10]、[13:12]、[15:14]為 00
分別設定 GPD4、5、6、7 為輸入接腳

- 將 GPC10、11、14、15 接腳之輸出狀態設定為高電位，內容如下：

#define MATRIX_CONTROL0_H CSR_WRITE(rPDATC,(CSR_READ(rPDATC) | 0x400))

暫存器 PDATC 之 Bit[10]為 1 代表 GPC10 為輸出狀態為高
電位

#define MATRIX_CONTROL1_H CSR_WRITE(rPDATC,(CSR_READ(rPDATC) | 0x800))

暫存器 PDATC 之 Bit[11]為 1 代表 GPC11 為輸出狀態為高
電位

#define MATRIX_CONTROL2_H CSR_WRITE(rPDATC,(CSR_READ(rPDATC) | 0x4000))

暫存器 PDATC 之 Bit[14]為 1 代表 GPC14 為輸出狀態為高
電位

#define MATRIX_CONTROL3_H CSR_WRITE(rPDATC,(CSR_READ(rPDATC) | 0x8000))

暫存器 PDATC 之 Bit[15]為 1 代表 GPC15 為輸出狀態為高電位

- 讀取鍵盤狀態函式---- ***mKey_ReadRowData***

```
void mKey_ReadRowData(int signum)
{
    unsigned int status;
    MatrixKeyStatus = 0x00;
    MATRIX_CONTROL0_L;
    status = (MATRIX_KEY_READ << 12) & 0xf000;
    MatrixKeyStatus |= status;
    MATRIX_CONTROL0_H;

    MATRIX_CONTROL1_L;
    status = (MATRIX_KEY_READ << 8) & 0xf00;
    MatrixKeyStatus |= status;
    MATRIX_CONTROL1_H;

    MATRIX_CONTROL2_L;
    status = (MATRIX_KEY_READ << 4) & 0xf0;
    MatrixKeyStatus |= status;
    MATRIX_CONTROL2_H;

    MATRIX_CONTROL3_L;
    status = MATRIX_KEY_READ;
    MatrixKeyStatus |= status;
    MATRIX_CONTROL3_H;
.....
}
```

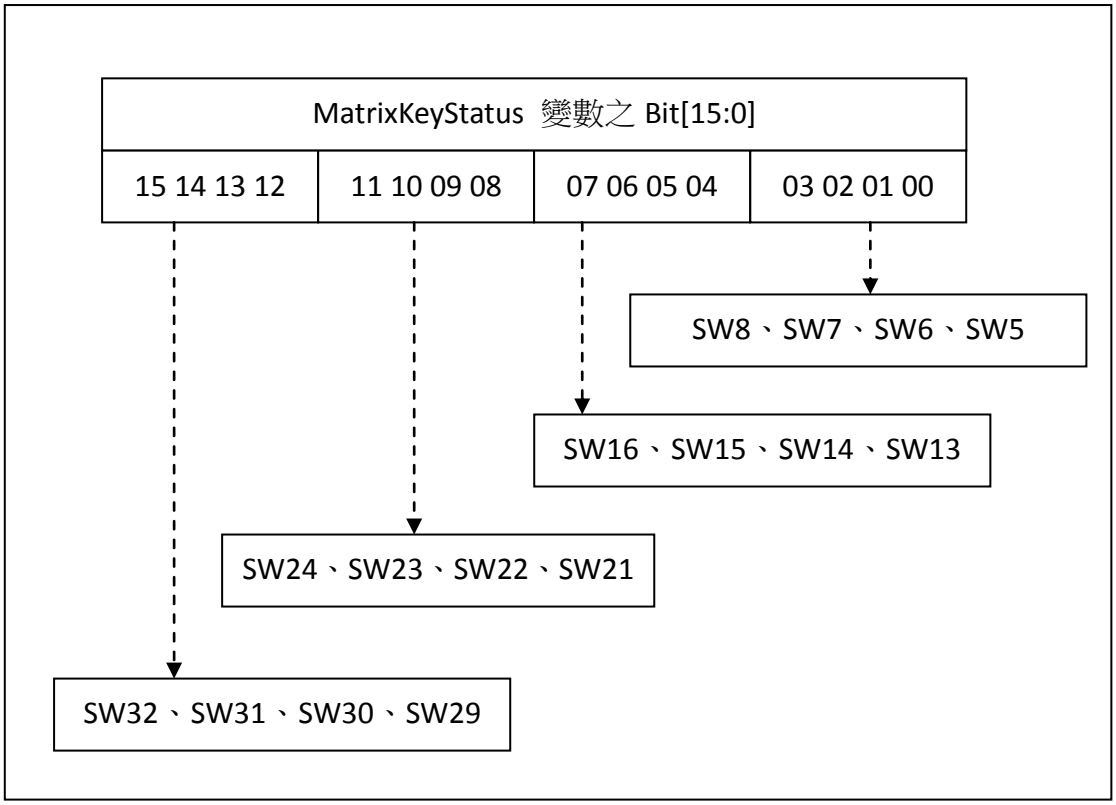
鍵盤讀取的方式為，依序將所對應之輸出埠之狀態設定為低電位，然後再讀入輸入埠之狀態，此狀態為所對應按鍵之狀態，讀取完後再恢復高電位，以第一組為例說明：

- 將 GPC10 接腳之輸出狀態，設定為低電位：

#define MATRIX_CONTROL0_L CSR_WRITE(rPDATC,(CSR_READ(rPDATC) & 0xfbff))

暫存器 PDATC 之 Bit[10]為 0 代表 GPC10 為輸出狀態為低電位

- 將四次所讀到的狀態存於 MatrixKeyStatus 變數中，代表不同按鍵之狀態：



最後一步，就是要想辦法定時呼叫函式 mKey_ReadRowData 掃描鍵盤狀態，那就是 sigaction 函式的使用。

四、 sigaction 函式的使用

sigaction()函式會依參數 signum 指定 SIGKILL 和 SIGSTOP 以外的信號編號來設置該信號的處理函數。

函式名稱： int sigaction(int signum,const struct sigaction *act ,struct sigaction *oldact)

參數說明：

int signum 信號編號

struct sigaction *act 定義產生信號的動作結構

```
struct sigation{        void   (*sa_handler) (int);
                     sigset_t  sa_mask;
                        int    sa_flags;
                       void   (*sa_restorer) (void);
                   }
```

void (*sa_handler)(int) 信號處理函式，請參閱 signal.h

sigset_t sa_mask 設置在處理該信號時暫時將 sa_mask 指定的信號擱置

int sa_flags 用來設置信號處理的其他相關操作，可用 OR(|)來組合下列的數值

A_NOCLDSTOP	參數 signum 為 SIGCHLD 時，子行程暫停不會通知父行程
SA_ONESHOT 或 SA_RESETHAND	調用新的信號處理函數前，將此信號處理方式改為系統預設的方式
SA_RESTART	被信號中斷的系統調用會自行重新啟動
SA_NOMASK 或 SA_NODEFER	在信號處理未結束前不理會此信號的再次到來

void(*sa_restorer)(void)　預留，尚未使用

struct sigaction *oldact　不是 NULL 指標，則原來的信號處理方式會由此結構 sigaction 返回

回傳值：　執行成功則返回 0，如果有錯誤則返回-1

五、 settimer 函式的使用

程式中使用 alarm 來安排核心調用行程在指定秒數後發出一個 sigalrm 的信號；如果指定的參數 seconds 為 0，則不再發送 sigalrm 信號。後一次設定將取消前一次的設定。在使用時，alarm 只設定為發送一次信號，如果要多次發送，就要多次使用 alarm 調用；所以很多程式不再使用 alarm 調用，而是使用 setitimer 調用來設置計時器。

函式名稱：　int setitimer(int which, const struct itimerval *value, struct itimerval *ovalue);

參數說明：

int which　settimer 提供了三個計時器，它們各自有其獨有的計時域，當其中任何一個到達，就發送一個對應的信號給行程，並重新開始計時。三個計時器由參數 which 指定。

- timer_real　按實際時間計時，計時到達時發送 ***sigalrm*** 信號
- itimer_virtual　僅當行程執行時才進行計時。計時到達時發送 ***sigvtalrm*** 信號
- itimer_prof　行程執行時和系統為該行程執行動作時都計時。與 itimer_virtual 是一對，該計時器經常用來統計進程在使用者態和內核態花費的時間。計時到達將發送 ***sigprof*** 信號。

*value　用來指明計時器的時間，其結構及說明如下：

```
struct itimerval {
 struct timeval   it_interval;     /*下一次的取值*/
 struct timeval   it_value;        /*本次的設定值*/
 };
```

timeval 結構說明如下：

```
struct timeval {
```

```
        long    tv_sec;     /* 秒 */
                tv_usec;    /* 微秒，1 秒 = 1000000 微秒
                            */
    };
```

ovalue　參數 **ovalue** 如果不為空，則其中保留的是上次呼叫所設定的值。

傳回值　執行成功則返回 0，如果有錯誤則返回-1

計時器動作
- ✧ 計時器將 it_value 遞減到 0 時，產生一個信號，並將 it_value 的值設定為 it_interval 的值，然後重新開始計時，如此往復。
- ✧ 當 it_value 設定為 0 時，或者當它計時到期，而 it_interval 為 0 時停止。

六、 定時掃描鍵盤

最後我們利用 settimer 函式來設定定時掃描鍵盤的時間，並且透過 sigaction 函式來設定產生 SIGVTALRM 信號

- 設定掃描鍵盤時間：利用 settimer 函式設定每隔 100ms 掃描鍵盤，程式碼說明如下：

程式碼	說明
scantimer.it_value.tv_sec = 0; scantimer.it_value.tv_usec = 100000;	it_value 為現在的時間, 遞減至 0 後，將 it_interval 的內容存至 it_value，如此將會定期的產生信號
scantimer.it_interval.tv_sec = 0; scantimer.it_interval.tv_usec = 100000;	
setitimer (ITIMER_VIRTUAL, &scantimer, NULL);	利用 settimer 函式設定鍵盤掃描時間

- 設定產生信號：

程式碼	說明
sa.sa_handler = &mKey_ReadRowData; sigaction (SIGVTALRM, &sa, NULL);	利用 sigaction 函式，設定產生 SIGVTALRM 信號的服務函式為 mKey_ReadRowData

想要測試的讀者，可下載連結壓縮檔並於 ESD44B0_B 目標板上實作。

Victor 於加拿大