Lab GPIO:嵌入式硬體平台輸出入實驗

課程：嵌入式系統與軟體工程
開發學校：台大電機系
王勝德 教授

Outline

- 1. 實驗器材
- 2. 實驗所需軟體
- 3. 簡介
- 4. 背景知識
- 5. 延伸討論

實驗器材

- PC x 2
  - Requirement: PC or Notebook computer
  - 一台安裝WindowsXP，另一台安裝Linux (CentOS).
- DMA-2440 開發平台支援WINCE 和Linux 嵌入作業系統。
- 製造商: 長高科技股份有限公司 DMATEK Co.,Ltd
  TEL: (04)2317-8130
  FAX: (04)2315-7426
  台中市西屯區文心路三段155-1號6樓
  http://www.dmatek.com.tw/tn/index.asp

實驗所需軟體

- Toolchain 3.4.1 安裝在 Linux 環境之下(交叉編譯用)
- Winscp安裝在Windows環境之下(File transfer between Linux and Windows)
- keypad.c , led.c ,led.h , Makefile 包含在 src files 資料夾內
- kernel source (linux-2.6.14.7-2440) 在光碟內可取得,建議安裝在 /root/linux-2.6.14.7-2440

簡介

- DMA-2440 有8個按鍵 (SW1...SW8) and 4個LEDs (LED1...LED4)。
- 驅動程式的源碼有 keypad.c led.c led.h and a Makefile 和相對應的 keypad.ko and led.ko 在此實驗會用到。
- 當某一個鍵被按下，相對應的IRQ會被觸發而導至IRQ service routine去開或關相對應的LEDs。

背景知識

- 1. Linux Interrupt Handling
  - 1. Interrupt 概觀
  - 2. 一般interrupt 的處理流程 (以shared IRQ為例)
- 2. Keypad Driver Design
- 3. Interrupt Service Routines
- 4. Recognizing buttons
- 5. Testing on DMA-2440 board

Linux Interrupt Handling(1/8)

- 1. Interrupt 概觀
- Interrupt lines是相當珍貴而且有限的資源，特別是在只有15或16條interrupt lines的x86機器上。Linux kernel內部會有一個”interrupt register table”，記載registry of interrupt lines和interrupt handler間的對應。
- 要使用IRQ (interrupt request) 的模組之前，必須先向kernel 登記特定的IRQ，並且在使用完畢後釋放該IRQ。
- request_irq()、free_irq() declared in < include/linux/sched.h >，可分別用來註冊及釋放IRQ，此外 request_irq() 也做安裝interrupt handler的工作。

```
int request_irq(
        unsigned int irq,
        void (*handler)(int, void *, struct pt_regs *),
        unsigned long flags,
        const char *dev_name,
        void *dev_id
);

void free_irq(unsigned int irq, void *dev_id);
```

Linux Interrupt Handling(2/8)

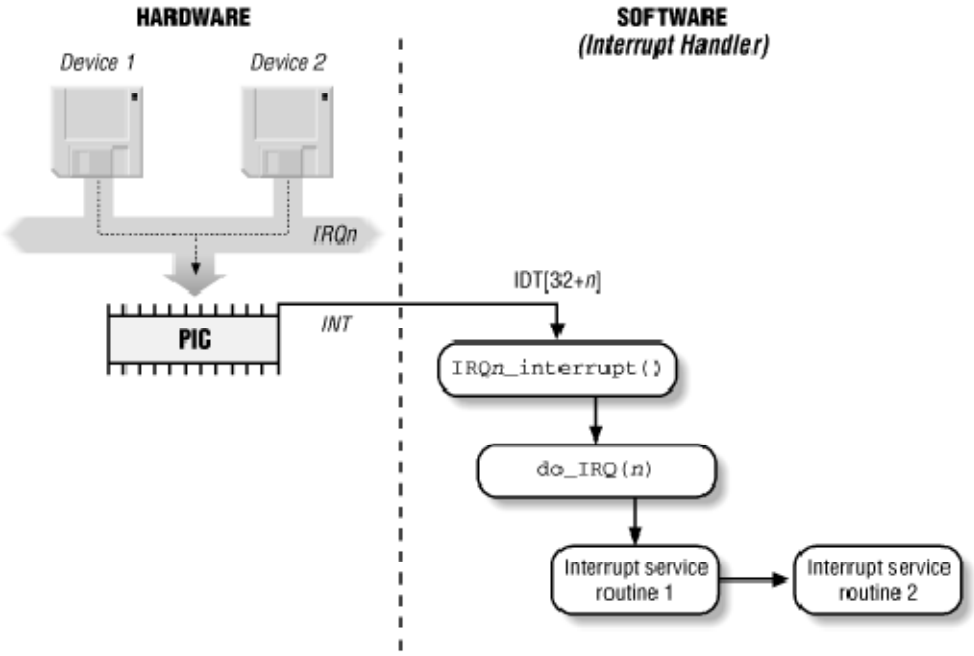
- 在Linux底下，/proc/interrupts 記錄目前IRQ使用的情況，e.g.

```
           CPU0
  0:     242755          XT-PIC  timer
  1:        484          XT-PIC  i8042
  2:          0          XT-PIC  cascade
  9:        126          XT-PIC  acpi, uhci_hcd, yenta, Intel 82801DB-ICH4
 10:          2          XT-PIC  uhci_hcd, ehci_hcd, yenta
 11:          5          XT-PIC  uhci_hcd, eth0
 12:         67          XT-PIC  i8042
 14:       2452          XT-PIC  ide0
 15:          2          XT-PIC  ohci1394
NMI:          0
LOC:     242702
ERR:          0
MIS:          0
```

- 一般常見的裝置(e.x. serial、parallel) 都有其固定且可能的IRQ數值。
- 其他裝置也可藉由試誤法(try & error)，經由kernel 輔助，或是驅動程式自行擁有一段程式去測試可用的IRQ。一般而言，系統內interrupt的數量(NR_IRQS) 定義於 <asm-arm/irq.h >、<fixup_irq() ; defined in asm/arch/irq.h>

Linux Interrupt Handling(3/8)

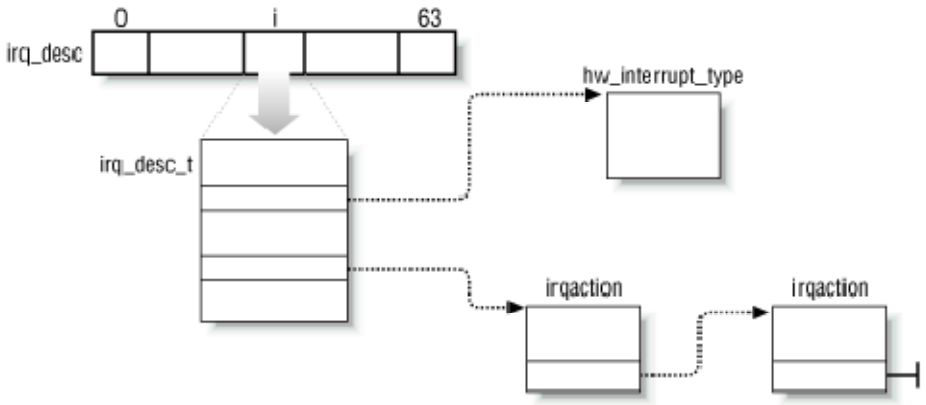
- 2. 一般interrupt 的處理流程 (以shared IRQ為例)
- Interrupt handling



- The two main IRQ descriptors : irq_desc_t & irq_action

Linux Interrupt Handling(4/8)

- IRQ descriptors



Linux Interrupt Handling(5/8)

- The irqaction descriptor
- An irq _desc array includes NR_IRQS irq _desc_t descriptors, which include the following fields:
  - Status
    A set of flags describing the IRQ line status.
  - Handler
    Points to the hw_interrupt_type descriptor that identifies the PIC circuit servicing the IRQ line.
  - Action
    Identifies the interrupt service routines to be invoked when the IRQ occurs. The field points to the first element of the list of irqaction descriptors associated with the IRQ. The irqaction descriptor is described briefly later in the chapter.

Linux Interrupt Handling(6/8)

- The irqaction descriptor
- As described earlier, multiple devices can share a single IRQ. Therefore, the kernel maintains irqaction descriptors, each of which refers to a specific hardware device and a specific interrupt. The descriptor includes the following fields.
  - Handler
    Points to the interrupt service routine for an I/O device. This is the key field that allows many devices to share the same IRQ.
  - Flags
    Describes the relationships between IRQ line and I/O device in a set of flags:

Linux Interrupt Handling(7/8)

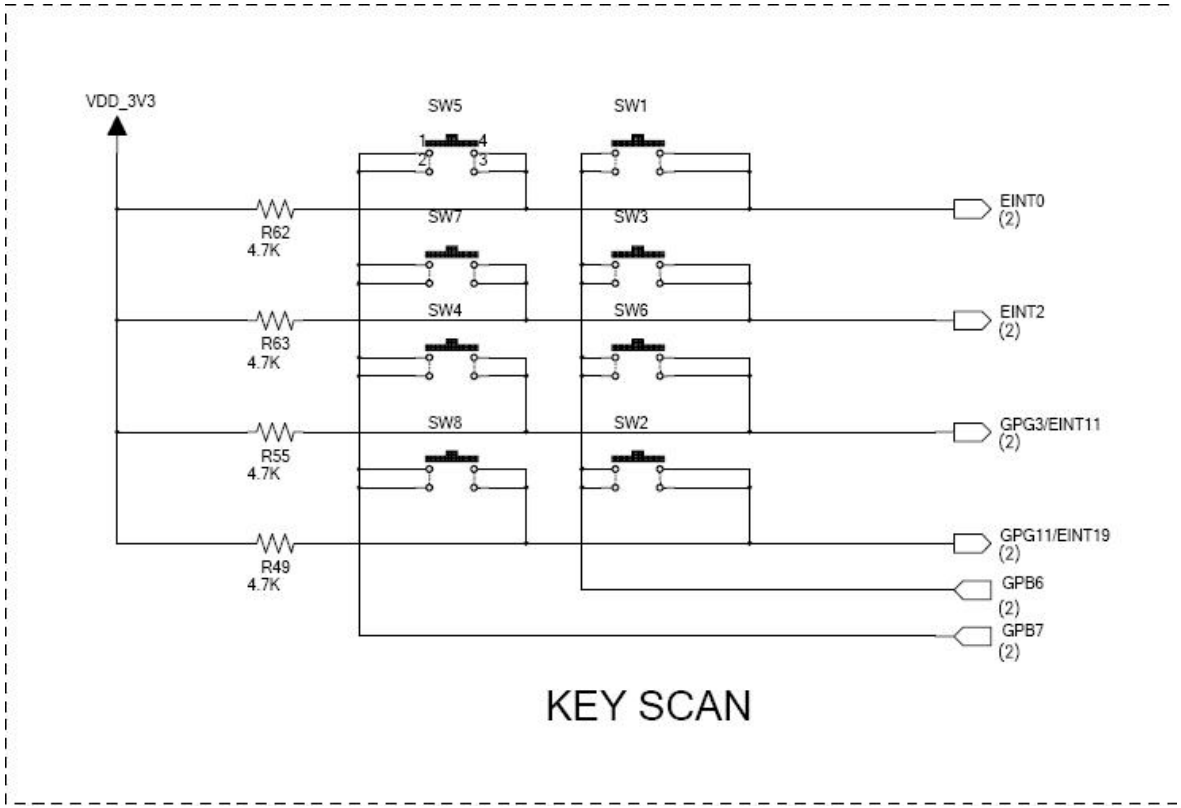
- SA_INTERRUPT
  The handler must execute with interrupts disabled.
- SA_SHIRQ
  The device permits its IRQ line to be shared with other devices.
- SA_SAMPLE_RANDOM
  The device may be considered as a source of events occurring randomly; it can thus be used by the kernel random number generator. (Users can access this feature by taking random numbers from the */dev/random* and */dev/urandom* device files.)
- SA_PROBE
  The kernel is using the IRQ line while performing a hardware device probe.

Linux Interrupt Handling(8/8)

- Name
  Names of the I/O device (shown when listing the serviced IRQs by reading the */proc/interrupts* file).
- dev_id
  The major and minor numbers that identify the I/O device.
- Next
  Points to the next element of a list of irqaction descriptors. The elements in the list refer to hardware devices that share the same IRQ.

Keypad Driver Design(1/8)

- Keypad 線路圖



Keypad Driver Design(2/8)

- 這在線路圖裡有4個外部中斷信號源(EINT0 ,EINT2,EINT11, EINT19)，初始電位為高電位VDD_3V3。我們可以將GPIO PIN，GPB6和GBP7 設定為低電位使得當按鍵被按下時，產生高電位轉低電位的電位轉態，以作為觸發中斷的來源。
- 由線路號我們知道SW1 和SW5有相同的中斷信號EINT0，所以我們必需要分辨當按鍵被按下時，有低電位產生，是來自SW1還是SW5，這正是KEY SCAN的重點所在。

```
348     // Enable interrupt
349     set_irq_type(IRQ_EINT0, IRQT_FALLING);
350     ret = request_irq(IRQ_EINT0, isr_kbd_eint0, SA_INTERRUPT, DEVICE_NAME, kbd);
351     if (ret)
352     {
353         printk( " Interrupt init=%x!!!!\n",ret);
354         return ret;
355     }
```

- 以 EINT0為例，在keypad.c的kbd_open()函式中，我們可以看到EINT0為設計為負緣觸發(IRQT_FALLING IRQ)的方式。

Keypad Driver Design(3/8)

- 初始化：
- 我們需對EINTs和GBP6,GBP7的電位作初始化，以符合我們設計的需求。所以我們需要參考相關的資料來設定相關的暫存器的值。

PORT F CONTROL REGISTERS (GPFCON, GPFDAT)

If GPF0–GPF7 will be used for wake-up signals at power down mode, the ports will be set in interrupt mode.

Register	Address	R/W	Description	Reset Value
GPFCON	0x56000050	R/W	Configures the pins of port F	0x0
GPFDAT	0x56000054	R/W	The data register for port F	Undef.
GPFUP	0x56000058	R/W	Pull-up disable register for port F	0x000
Reserved	0x5600005c	–	–	–

GPFCON	Bit	Description
GPF7	[15:14]	00 = Input 01 = Output 10 = EINT[7] 11 = Reserved
GPF6	[13:12]	00 = Input 01 = Output 10 = EINT[6] 11 = Reserved
GPF5	[11:10]	00 = Input 01 = Output 10 = EINT[5] 11 = Reserved
GPF4	[9:8]	00 = Input 01 = Output 10 = EINT[4] 11 = Reserved
GPF3	[7:6]	00 = Input 01 = Output 10 = EINT[3] 11 = Reserved
GPF2	[5:4]	00 = Input 01 = Output 10 = EINT2] 11 = Reserved
GPF1	[3:2]	00 = Input 01 = Output 10 = EINT[1] 11 = Reserved
GPF0	[1:0]	00 = Input 01 = Output 10 = EINT[0] 11 = Reserved

Keypad Driver Design(4/8)

- 根據GPIO controller的規格書的Port F控制暫存器，我們將GPF0設定為EINT[0] 而且GPF2為EINT[2]形式。

- 在G ports裡，我們將GPG3設定為EINT[11]而GPF11為EINT[19]形式。

Keypad Driver Design(5/8)

PORT G CONTROL REGISTERS (GPGCON, GPGDAT)

If GPG0–GPG7 will be used for wake-up signals at Sleep mode, the ports will be set in interrupt mode.

Register	Address	R/W	Description	Reset Value
GPGCON	0x56000060	R/W	Configures the pins of port G	0x0
GPGDAT	0x56000064	R/W	The data register for port G	Undef.
GPGUP	0x56000068	R/W	Pull-up disable register for port G	0xfc00

GPGCON	Bit	Description	
GPG15*	[31:30]	00 = Input 10 = EINT[23]	01 = Output 11 = Reserved
GPG14*	[29:28]	00 = Input 10 = EINT[22]	01 = Output 11 = Reserved
GPG13*	[27:26]	00 = Input 10 = EINT[21]	01 = Output 11 = Reserved
GPG12	[25:24]	00 = Input 10 = EINT[20]	01 = Output 11 = Reserved
GPG11	[23:22]	00 = Input 10 = EINT[19]	01 = Output 11 = TCLK[1]
GPG10	[21:20]	00 = Input 10 = EINT[18]	01 = Output 11 = nCTS1
GPG9	[19:18]	00 = Input 10 = EINT[17]	01 = Output 11 = nRTS1
GPG8	[17:16]	00 = Input 10 = EINT[16]	01 = Output 11 = Reserved
GPG7	[15:14]	00 = Input 10 = EINT[15]	01 = Output 11 = SPICLK1
GPG6	[13:12]	00 = Input 10 = EINT[14]	01 = Output 11 = SPIMOSI1
GPG5	[11:10]	00 = Input 10 = EINT[13]	01 = Output 11 = SPIMISO1
GPG4	[9:8]	00 = Input 10 = EINT[12]	01 = Output 11 = LCD_PWRDN
GPG3	[7:6]	00 = Input 10 = EINT[11]	01 = Output 11 = nSS1
GPG2	[5:4]	00 = Input 10 = EINT[10]	01 = Output 11 = nSS0
GPG1	[3:2]	00 = Input 10 = EINT[9]	01 = Output 11 = Reserved
GPG0	[1:0]	00 = Input 10 = EINT[8]	01 = Output 11 = Reserved

Keypad Driver Design(6/8)

- **步驟1：**
- 在keypad.c裡的kdc_init(void)函式內加入以下的程式碼。

```
493         gpfcon = __raw_readl(S3C2410_GPFCON);
494         gpfcon &= 0x0<<0;
495         gpfcon &= 0x0<<1;
496         gpfcon &= 0x0<<4;
497         gpfcon &= 0x0<<5;
498         gpfcon |= 0x1<<1;
499         gpfcon |= 0x1<<5;
500         __raw_writel(gpfcon,S3C2410_GPFCON);
501
502         gpfcon = __raw_readl(S3C2410_GPGCON);
503         gpfcon &= 0x0<<6;
504         gpfcon &= 0x0<<7;
505         gpfcon &= 0x0<<22;
506         gpfcon &= 0x0<<23;
507         gpfcon |= 0x1<<7;
508         gpfcon |= 0x1<<23;
509         __raw_writel(gpfcon,S3C2410_GPGCON);
510
```

- 我們需將GPB6和GPB7也作相關的設定，根據Port B的表格，我們設定GBP6和GPB7為輸出信號的形式，並且將其電位設定為低電位。

Keypad Driver Design(7/8)

PORT B CONTROL REGISTERS (GPBCON, GPBDAT, GPBUP)

Register	Address	R/W	Description	Reset Value
GPBCON	0x56000010	R/W	Configures the pins of port B	0x0
GPBDAT	0x56000014	R/W	The data register for port B	Undef.
GPBUP	0x56000018	R/W	Pull-up disable register for port B	0x0
Reserved	0x5600001c			

PBCON	Bit	Description
GPB10	[21:20]	00 = Input 01 = Output 10 = nXDREQ0 11 = reserved
GPB9	[19:18]	00 = Input 01 = Output 10 = nXDACK0 11 = reserved
GPB8	[17:16]	00 = Input 01 = Output 10 = nXDREQ1 11 = Reserved
GPB7	[15:14]	00 = Input 01 = Output 10 = nXDACK1 11 = Reserved
GPB6	[13:12]	00 = Input 01 = Output 10 = nXBREQ 11 = reserved
GPB5	[11:10]	00 = Input 01 = Output 10 = nXBACK 11 = reserved
GPB4	[9:8]	00 = Input 01 = Output 10 = TCLK [0] 11 = reserved
GPB3	[7:6]	00 = Input 01 = Output 10 = TOUT3 11 = reserved
GPB2	[5:4]	00 = Input 01 = Output 10 = TOUT2 11 = reserved]
GPB1	[3:2]	00 = Input 01 = Output 10 = TOUT1 11 = reserved
GPB0	[1:0]	00 = Input 01 = Output 10 = TOUT0 11 = reserved

GPBDAT	Bit	Description
GPB[10:0]	[10:0]	When the port is configured as input port, the corresponding bit is the pin state. When the port is configured as output port, the pin state is the same as the corresponding bit. When the port is configured as functional pin, the undefined value will be read.

GPBUP	Bit	Description
GPB[10:0]	[10:0]	0: The pull up function attached to the corresponding port pin is enabled. 1: The pull up function is disabled.

Keypad Driver Design(8/8)

- **步驟2：**
- 在keypad.c 裡的kdc_init(void)函式內加入以下的程式碼，

```
513         gpfcon = __raw_readl(S3C2410_GPBCON);
514         gpfcon &= 0x0<<12;
515         gpfcon &= 0x0<<13;
516         gpfcon &= 0x0<<14;
517         gpfcon &= 0x0<<15;
518         gpfcon |= 0x1<<12;
519         gpfcon |= 0x1<<14;
520         __raw_writel(gpfcon,S3C2410_GPBCON);
521
522         gpfcon = __raw_readl(S3C2410_GPBDAT);
523         gpfcon &= 0x0<<6;
524         gpfcon &= 0x0<<7;
525         __raw_writel(gpfcon,S3C2410_GPBDAT);
526
527
528
529         gpfcon = __raw_readl(S3C2410_GPBUP);
530         gpfcon &= 0x0<<6;
531         gpfcon &= 0x0<<7;
532         __raw_writel(gpfcon,S3C2410_GPBUP);
533
```

Interrupt Service Routines(1/3)

- 初始化之後，註冊ISRs，概述一下ISRs需要做哪些事情：
  - a.設定 EINT 為輸入形式，以便之後的讀取使用。
  - b.呼叫副程式test_row (KBD_DEV *kbd,int col)，其中參數col = 0 for EINT0 ,col=1 for EINT2，以此類推。
  - c.副程式test_row將辨認哪個按鍵被按下，且將相對應的LED燈點亮或關閉。
  - d.執行test_row後，將EINT設定回EINT形式且回傳IRQ_HANDLED。

Interrupt Service Routines(2/3)

- **步驟3：**
- 在keypad.c裡的isr_kdb_eint0函式內加入以下的程式碼

```
220 static irqreturn_t isr_kbd_eint0(int irq, void *dev_id, struct pt_regs *reg)
221 {
222
223         KBD_DEV * kbd = (KBD_DEV *) dev_id;
224         int mask;
225         int gpfcon,gpfdat;
226         int i=1;
227
228         spin_lock_irq(&(kbd->lock));
229         gpfcon = __raw_readl(S3C2410_GPFCON);
230         gpfcon &= ~(0x3);
231         gpfcon |= 0x0;
232         __raw_writel(gpfcon,S3C2410_GPFCON);
233         gpfdat = __raw_readl(S3C2410_GPFDAT);
234         printk("#######irqreturn_t isr_kbd_eint0#########\n\r");
235
236         test_row(kbd,0);
237
238         gpfcon = __raw_readl(S3C2410_GPFCON);
239         gpfcon &= ~(0x3);
240         gpfcon |= 0x2;
241         __raw_writel(gpfcon,S3C2410_GPFCON);
242
243         spin_unlock_irq(&(kbd->lock));
244         return IRQ_HANDLED;
245
246 }
```

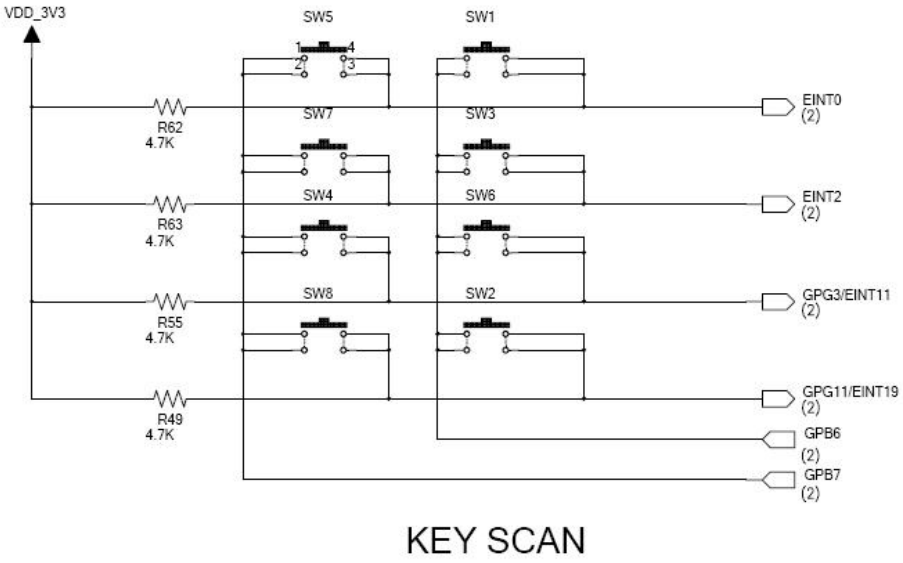
Interrupt Service Routines(3/3)

- 其中的涵意為：
- Step a: 第229 至 232行
- Step b and c:第236行
- Step d: 244行

- **請參考上面程式碼寫法，將EINT2,EINT11 EINT19初始化。**

Recognizing buttons(1/4)

- 稍早我們提及到KEY SCAN為辨認哪個按鍵被按下，現在我們來討論它的機制為何。
- 我們再觀察一下線路圖，以EINT0為例

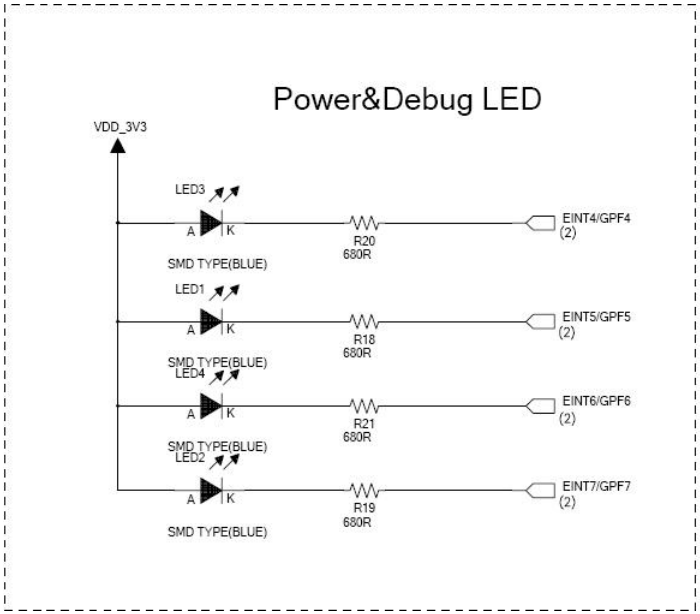


Recognizing buttons(2/4)

- 當EINT0拉至低電位將導致寫進入ISR，而去呼叫副程式test_row 。此時， EINT0的電壓值為零因為SW1或SW5被按下了(GPB6和GPB7初始化為低電位)。以下方式為辨認哪個按鍵被按下了：
  - a.首先將GPB6 設為1 (high voltage).
  - b.再去讀EINT0的電位值
  - c.再將GPB6設回0 (ground)
- 在step a，由於GPB6 被設為1，假如是SW1被按下則EINT0得到的電位值就會是1。倘若是SW5被按下則EINT0得到的電位值會是0。同理，以此類推，我們將可以辨認出8個按鍵的事件。

Recognizing buttons(3/4)

- 為什麼這樣的機制可以工作，這是因為software function call的回應時間為瞬間，在我們按下按鍵的那一瞬間，就執行完了。所以在按鍵被下時，我們可以借由將GPB6設為1，再去讀EINT0去判別之。
- LEDs的開/關



- 這是LED的線路圖，若LED要亮燈則另一端需為低電位，

Recognizing buttons(4/4)

- 步驟4：
- 在keypad.c 的test_row 函式內加入以下的程式碼：(此為LED1的例子，請將其他的LED和對應的SW，仿以下的方式，加入其程式碼)

SW1 : turn on LED1，SW2 :turn on LED2，
SW3 :turn on LED3，SW4 : turn on LED4
SW5 : turn off LED1，SW6 :turn off LED2，
SW7 :turn off LED3，SW8 :turn off LED4

```
114             gpfcon = __raw_readl(S3C2410_GPFDAT);//read EINT0
115             if(gpfcon&(0x1)) //SW1
116             {
117                 printk("SW 1 pushed! \n");
118                 gpfdat = __raw_readl(S3C2410_GPFDAT);
119                 gpfdat &=0<<5; //LED 1 ON
120                 printk("LED 1 : ON\n");
121                 __raw_writel(gpfdat,S3C2410_GPFDAT);
122                 for(i=0;i<60000;i++);
123                 for(i=0;i<60000;i++);
124             }
```

Testing on DMA-2440 board(1/6)

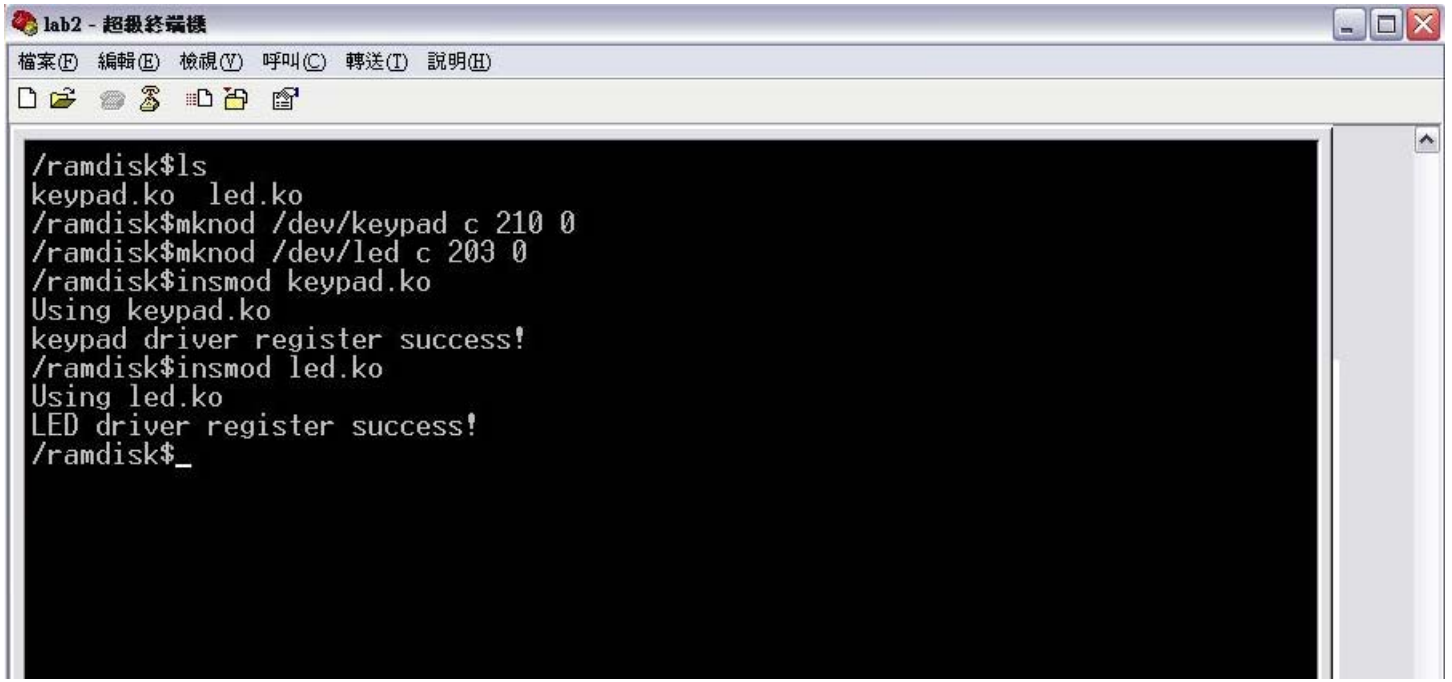
- 步驟5：
- 使用Makefile將keypad.c交叉編譯，

```
## s3c2440 kernel path
S3C2440_KERNEL_DIR = /root/linux-2.6.14.7-2440

ifneq ($(KERNELRELEASE),)
        obj-m := keypad.o led.o
else
        KERNELDIR ?= $(S3C2440_KERNEL_DIR)
        PWD := $(shell pwd)
default:
        $(MAKE) -C $(KERNELDIR) M=$(PWD) modules
clean:
        rm -rf *.o *.mod.* *.ko *.o.* .*.*.cmd .tmp_versions
endif
```

Testing on DMA-2440 board(2/6)

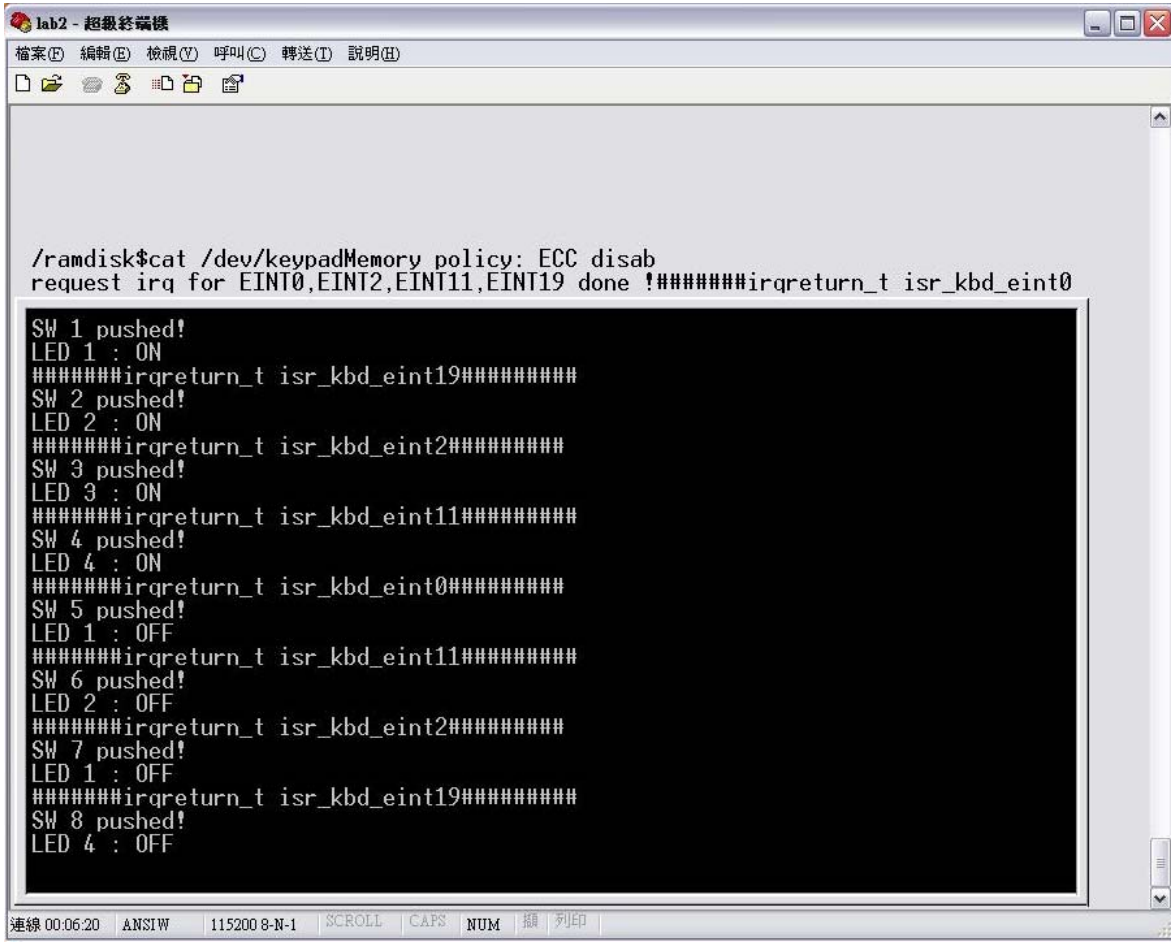
- *****其中S3C2440_KERNEL_DIR為核心目錄的路徑*****
  - a.將編譯好的 keypad.ko和led.ko下載到開發板上，利用chmod 指令更改權限，然後insmod且mknod，其中led major number 為203 而keypad major number 為210。



Testing on DMA-2440 board(3/6)

- b.執行指令cat /dev/keypad 測試驅動程式
- 在下圖，我們可以看到當我們按下按鍵，相對應的LED會被點亮，程式可以清楚的判斷出按鍵與LED的對應關係。

Testing on DMA-2440 board(4/6)



Testing on DMA-2440 board(5/6)

- SW1 : turn on LED1
- SW2 : turn on LED2
- SW3 : turn on LED3
- SW4 : turn on LED4
- SW5 : turn off LED1
- SW6 : turn off LED2
- SW7 : turn off LED3
- SW8 : turn off LED4

Testing on DMA-2440 board(6/6)

- 由下圖，我們可以看到在DMA-2440板子上按鍵和LED的行為。
- a.初始化

b.當SW5被按下,LED 1 關閉

- c.當SW5 SW6 SW7 SW8 都被按下,LED 1 -LED4 全部關閉

延伸討論

- 1.在註冊IRQ的時候,我們是在kbd_open()函式中去實作它。然而,我們也可以把IRQ的註冊寫在kbd_init()函式裡,如此一來當我們驅動裝置的時候(步驟5.a)就可以將IRQ註冊的步驟完成。試比較和思考其中的差別。
- 2.請嘗試其它種類按鍵與LED燈的組合。如:跑馬燈。