Upboard Lab03

tags: `Upobard` `Embedded`

I. Introduction

A. Interrupt

- 外部中斷(External Interrupt)
  CPU外的周邊元件所引起的(I/O Complete Interrupt, I/O Device error)
- 內部中斷(Internal Interrupt)
  不合法的用法所引起的，CPU 本身所引發。(Debug、Divide-by-zero、overflow)
- 軟體中斷(Software interrupt)
  由CPU執行一個軟體中斷指令產生的中斷要求

B. Three issues

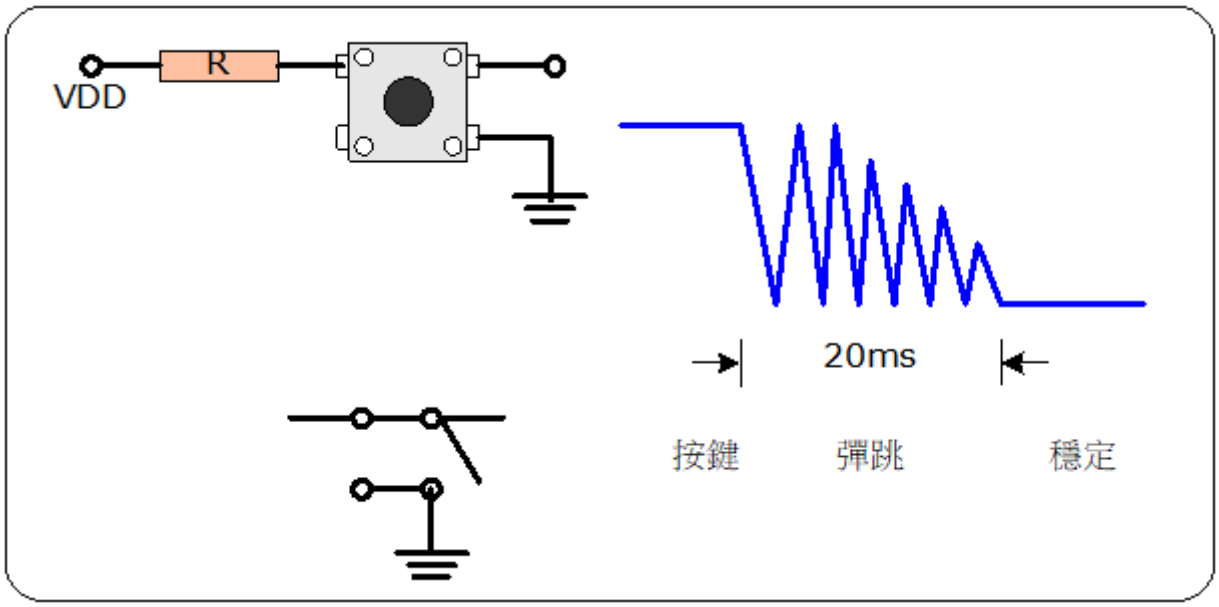
- Switch: 做為觸發中斷信號
- Timer: 改變led的亮滅頻率

- Interrupt: 了解中斷機制
- 本實驗之範例程式與練習題將實際操作與使用這些主題。
- 並了解Interrupt 與 polling的差異。

C. Switch

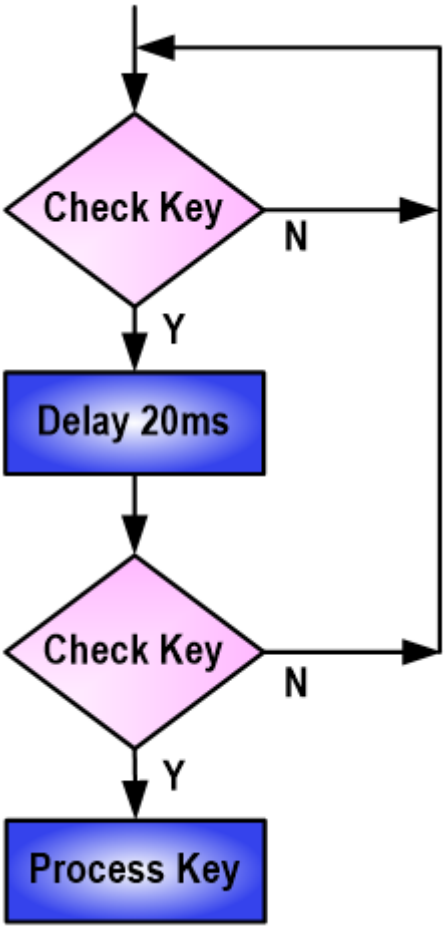
- 本實驗為使用 switch 按壓觸發外部中斷訊號，但 switch 會有機械彈跳訊號，造成判定上的問題，這是按鈕開關機構上的問題因此我們必須進行 debounce 來排除這個問題。

- 由 Hi 到 Lo 或 Lo 到 Hi 間之區域為不穩定區



- 防彈跳策略：
    - 首先偵測是否有按鍵
    - 若測得按鍵則啟動延時 20ms
    - 延時結束後再偵測按鍵
    - 若同樣測得按鍵表示為有效按鍵

- 否則視為干擾或無效按鍵



D. Kernel Timer

Kernel timer：

- 本實驗使用timer來改變led亮滅的頻率，要在kernel中寫Timer的機制，要對一些專有名詞有初步的了解，分別是HZ, Tick, Jiffies
- HZ：Linux kernel 每隔固定週期會發出timer interrupt，HZ就是用來定義每一秒有幾次 timer interrupts。例如:500 HZ 就是CPU每一秒會發出500個interrupt的意思
- Tick：Tick是HZ的倒數，意即timer interrupt每發生一次中斷所花的時間。如HZ為500時，tick為2毫秒 (millisecond)
- Jiffies：Jiffies為Linux核心變數，它被用來紀錄系統自開機以來，已經過多少的tick每發生一次timer interrupt，Jiffies變數會被加一

E. Lab1的polling與Lab3的 Interrupt 比較

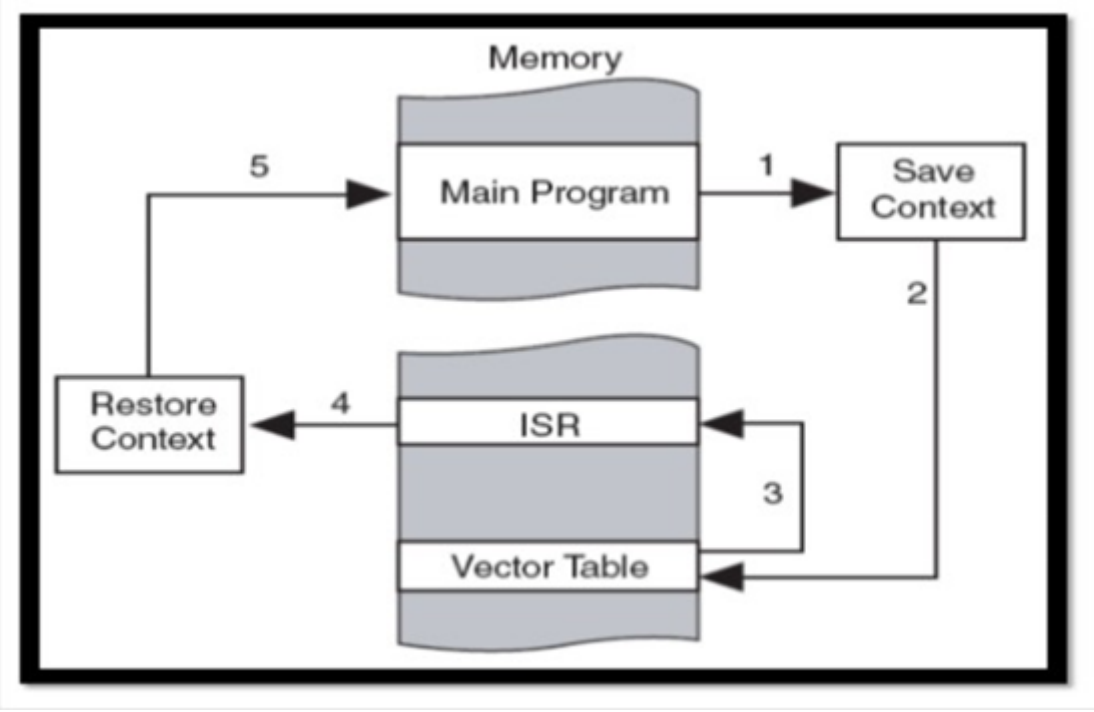
- Polling (檔案名稱: led_matrix)
  點矩陣每到9時讀取按鈕的狀態，還沒到9時按按鈕led不會亮，因此時沒讀取按鈕狀態。點矩陣到9時若有按下按鈕led為亮，沒按按鈕led為滅。當led為亮時放開按鈕燈不會馬上熄滅，直到點矩陣到9時，再次讀取按鈕狀態，沒按按鈕燈會熄滅。(不即時)

- Interrupt (檔案名稱: led_intmat)
  可以發現按下按鈕後燈馬上亮，放開按鈕燈馬上熄滅，因為 interrupt 機制為當一個I/O 設備需要服務時，會發出 interrupt 來通知 cpu。所以當按下按鈕時馬上得知有按下按鈕，燈馬上亮。不需像 polling 須等到點矩陣顯示為9時才讀取按鈕狀態，interrupt 方法(較即時)。

F. Interrupt 相關術語

- 中斷請求(IRQ) : 外部硬體發出一個電脈衝訊號，要求CPU暫停手邊的工作，去執行其他工作。
- 中斷服務程序(ISR) : 外部中斷訊號發生後，CPU會去執行的程序。
- 中斷號 : 當外部設備發出中斷請求訊號時，這個訊號會有特定的標誌，讓CPU知道是哪個設備發出中斷請求。
- 中斷向量 : 中斷服務程序的指令碼的起始位置
- 中斷向量表 : 紀錄每一個中斷號對應執行的中斷向量

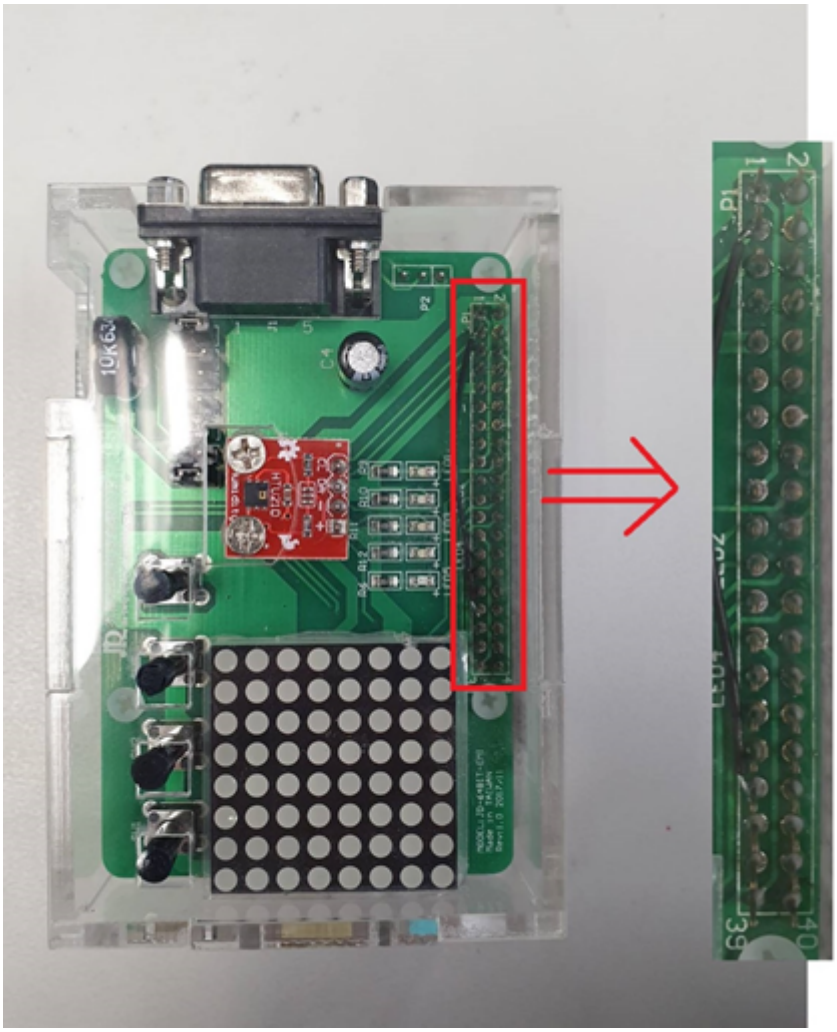
G. Interrupt 流程



1. 暫停目前 process 執行並保存此 process 當時執行狀況。
2. 根據 Interrupt ID 查尋 Interrupt vector 並取得 ISR (Interrupt Service Routine) 起始位址。
3. ISR執行
4. 執行完成，恢復先前 process 執行狀況
5. 回到原先中斷前的執行

H. Up-Board腳位對應圖

如圖所示，標示為GPIO的針腳可以拿來做為輸入、輸出使用



UP pinout	Pin	Pin	UP pinout
3.3V	1	2	5V
GPIO0	3	4	5V
GPIO1	5	6	Ground
GPIO2	7	8	GPIO15
Ground	9	10	GPIO16
GPIO3	11	12	GPIO17
GPIO4	13	14	Ground
GPIO5	15	16	GPIO18
3.3V	17	18	GPIO19
GPIO6	19	20	Ground
GPIO7	21	22	GPIO20
GPIO8	23	24	GPIO21
Ground	25	26	GPIO22
GPIO9	27	28	GPIO23
GPIO10	29	30	Ground
GPIO11	31	32	GPIO24
GPIO12	33	34	Ground
GPIO13	35	36	GPIO25
GPIO14	37	38	GPIO26
Ground	39	40	GPIO27

II. Demonstration

- 程式使用相關函式說明
  - **int gpio_request(unsigned gpio, const char *label)**;
    gpio_request 可以檢查GPIO number是否超出範圍或<0 及指定的GPIO是否正在使用。有時候別的地方也控制著同一個 GPIO 可能會發生程式預期外的結果。
  - **int gpio_to_irq(unsigned gpio)**
    把GPIO的PIN值轉換為相應的IRQ值，回傳值通常做為request_irq()的第一個參數。
  - **int gpio_direction_output(unsigned gpio, int value)**
    設置gpio的引腳為輸出。int value可設置gpio的pin為高電位(1)或低電位 (0)。
  - **ktime_get()**
    用於得到當前時間，時間單位為Nanosecond
  - **irq_handler_t**
    當IRQ發生時設置被呼叫的函式，負責處理（回應）實體的硬體中斷
  - int request_irq(unsigned int irq, irq_handler_t handler, unsigned long irqflags, const char *devname, void *dev_id)
    - int request_irq的參數說明:
      - unsigned int irq: 為要註冊中斷服務函數的中斷號。

        - irq_handler_t handler: 為要註冊的中斷服務函數。
        - unsigned long irqflags: 觸發中斷的參數,比如邊緣觸發。
        - const char *devname: 中斷程序的名字,使用
          cat /proc/interrupts 可以查看中斷程序名字
        - void *dev_id: 傳入中斷處理程序的參數,註冊共享中斷時不能為NULL,因為卸載時需要這個做參數,避免卸載其它中斷服務函數。此實驗非註冊共享中斷，所以設NULL。

- 檔案說明
    - Linux ko檔相關說明
      檔案副檔名 ko 是一種 Linux Kernel Module File。
    - insmod: 掛載 module
      ex: lab3-1-on_off.ko 檔要載入 kernel中，可輸入
      sudo insmod lab3-1-on_off.ko
    - rmmod: 卸載module
      ex: lab3-1-on_off.ko 檔要卸載，可輸入 sudo rmmod lab3-1-on_off.ko

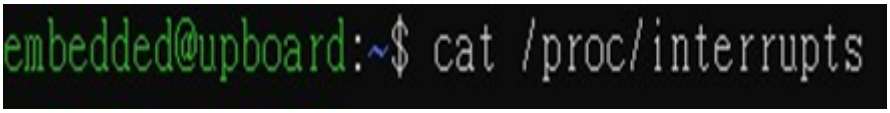
III. (lab3-1-on_off.c)程式建置的流程

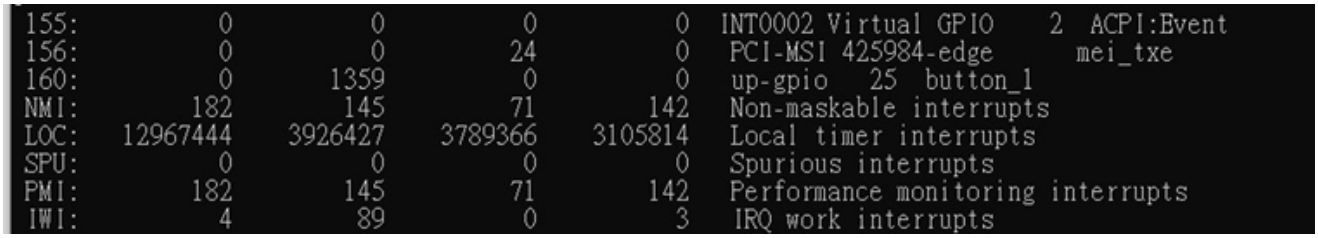
- 初始化開發環境(初始化過程只需1次):
    - sh setup.sh (http://setup.sh)
    - scp -r em_up:/lib/modules/5.4.0-1-generic/ ~
    - sudo mv ~/5.4.0-1-generic/ /lib/modules

- 編譯程式
    - lab3-1-on_off.c、 lab3-2-blink_freq.c 分別在二個不同的目錄下，目錄名稱lab3-1、lab3-2 。目錄路徑皆在src/ 資料夾下
    - 進入lab3-1 目錄裡面有 lab3-1-on_off.c 程式,執行make後會產生lab3-1-on_off.ko
    - 再把 lab3-1-on_off.ko 檔由 em_dev 環境移至 em_up 環境
        - scp lab3-1-on_off.ko em_up:
    - ssh 進入upBoard 環境
        - ssh em_up

- 進入upBoard環境後
    - 執行 lab3-1-on_off.ko 檔載入程序
    - sudo insmod lab3-1-on_off.ko
    - 載入程序後請參閱投影片第27頁實驗結果確認符合實驗要求

- 若要移除 lab3-1-on_off.ko 輸入
  - sudo rmmod lab3-1-on_off.ko
- 實驗結果確認
  1. 可在upBoard環境下終端輸入cat /proc/interrupts確定ISR已被註冊。此實驗定義IRQ名稱為 button_1。
  2. 實驗結果: 按下SW1按鍵可看見LED燈的狀態會跟著改變 (亮→熄) (熄→亮)





IV. 範例程式(二)

程式執行後LED1以每0.1秒閃爍的頻率亮滅，按下UpBoard上的SW1按鍵觸發外部中斷，使程式執行ISR，ISR的內容為變更LED1燈狀態由0.1秒的閃爍頻率變為1秒的閃爍頻率。

A. 程式使用相關函式說明

- **timer_setup(struct timer_list *timer, callback, (unsigned int) flags)**:
  用於第一次使用timer做好準備。callback: 當計時器結束時被呼叫的函式
- **mod_timer(struct timer_list *timer, unsigned long expires)**:
  修改計時器的結束時間
- **gpio_request_one(24, GPIOF_INIT_LOW, “ gpio_24”)**:
  宣告gpio號24的pin為輸出用，並設置為低電位
- **gpio_request_one(25, GPIOF_IN, “gpio_25”)**:
  宣告gpio號25的pin為輸入用
- **int gpio_direction_output(unsigned gpio, int value)**:
  設置gpio的引腳為輸出。int value可設置gpio的pin為高電位(1)或低電位 (0)
- **static irqreturn_t myinterrupt(int irq, void dev_id)*
  中斷服務程序，輸入參數有兩個，分別是中斷號irq和dev_id。
- **msecs_to_jiffies(timeout_ms)**:
  Jiffies為Linux核心變數，它被用來紀錄系統自開機以來，已經過多少的 tick。每發生一次 timer interrupt，Jiffies變數會被加一。jiffies + msecs_to_jiffies(timeout_ms)為定義delay 多少毫秒。

- **module_param(name,type,perm)**
  有三個參數，分別是變數名稱、變數的資料型別，以及其對應 sysfs檔案的權限。sysfs 是 Linux 所採用的一種虛擬檔案系統，其功能就是為了要傳遞 kernel module 的資訊。
- int request_irq(unsigned int irq, irq_handler_t handler, unsigned long irqflags, const char *devname, void *dev_id)
  int request_irq 的參數說明:
  - unsigned int irq: 為要註冊中斷服務函數的中斷號。
  - irq_handler_t handler: 為要註冊的中斷服務函數。
  - unsigned long irqflags: 觸發中斷的參數,比如邊緣觸發。
  - const char *devname: 中斷程序的名字,使用cat/proc/interrupts 可以查看中斷程序名字
  - void *dev_id: 傳入中斷處理程序的參數,註冊共享中斷時不能為NULL,因為卸載時需要這個做參數,避免卸載其它中斷服務函數。此實驗非註冊共享中斷，所以設NULL。

Timer setup

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/sched.h>//jiffies在此頭文件中定義
#include <linux/init.h>
#include <linux/timer.h>
struct timer_list mytimer;//定義一個定時器
void mytimer_ok(unsigned long arg)
{
            printk("Mytimer is ok\n");
            printk("receive data from timer: %d\n",arg);
   }

static int __init hello_init (void)
{
    printk("hello,world\n");
    init_timer(&mytimer); //初始化定時器
    mytimer.expires = jiffies+100;//設定超時時間，100代表1秒
    mytimer.data = 250; //傳遞給定時器超時函數的值
    mytimer.function = mytimer_ok;//設置定時器超時函數
    add_timer(&mytimer); //添加定時器，定時器開始生效
    return 0;
}

static void __exit hello_exit (void)

{
    del_timer(&mytimer);//卸載模塊時，刪除定時器
    printk("Hello module exit\n");
}

module_init(hello_init);
module_exit(hello_exit);
MODULE_AUTHOR("CXF");
MODULE_LICENSE("Dual BSD/GPL");
```

- 交叉編譯後，放到開發板上：

#insmod timer.ko

可以發現過一秒後定時器過期函數被執行了，打印出了信息，250也被正確傳遞了。

#rmmod timer.ko

我們也可以用lsmod | grep timer 來查看是否加載了timer驅動。

可以用dmesg | tail -20 查看驅動打印的信息

dmesg -c 清楚信息

- 進一步理解定時器：

在上面的定時器超時函數mytimer_ok(unsigned long arg)中，添加如下代碼：

mytimer.expires = jiffies+100;//設定超時時間，100代表1秒

mytimer.function = mytimer_ok;//設置定時器超時函數

add_timer(&mytimer); //添加定時器，定時器開始生效

交叉編譯後，放到開發板上

#insmod timer.o

發現每隔一秒，mytimer_ok函數就執行一次，這是因為每次定時器到期後，都

又重新給它設置了一個新的超時時間，並且新的超時函數指向自己，形成一個遞

歸，所以就會一直執行下去。

#rmmod timer

可以卸載模塊，當然打印也就結束了，注意因為定時器超時函數不停的打印信息

，導致輸入上面的命令時會被定時器超時函數不停的打印信息淹沒，不用管他，

耐心的把上面的命令輸完就可以成功卸載。

- 可以參考以下網址了解Linux timer運作
  https://b8807053.pixnet.net/blog/post/3611170 (https://b8807053.pixnet.net/blog/post/3611170)

SSH 連接不上

解決方法：

1. 選擇 View -> Command Palette.
2. 輸入 remote
3. 選擇 Remote-SSH: Kill VS Code Server on Host...