微算機系統實習
MICROPROCESSOR SYSTEMS LAB.
SPRING, 2019

INSTRUCTOR： YEN-LIN CHEN(陳彥霖), PH.D.

PROFESSOR

DEPT. COMPUTER SCIENCE AND INFORMATION ENGINEERING

NATIONAL TAIPEI UNIVERSITY OF TECHNOLOGY

LECTURE 2 – 微算機系統實驗平台- GENERAL PURPOSE INPUT OUTPUT (GPIO)

OUTLINE

- GPIO簡介
- 嵌入式平台 TK1 的 GPIO 腳位介紹
- 透過 Ubuntu 操作 TK1 上的 GPIO
- 以 C++ 透過 Ubuntu Kernel 操作 TK1 上的 GPIO
- TK1 GPIO 相關參考資料

GPIO簡介

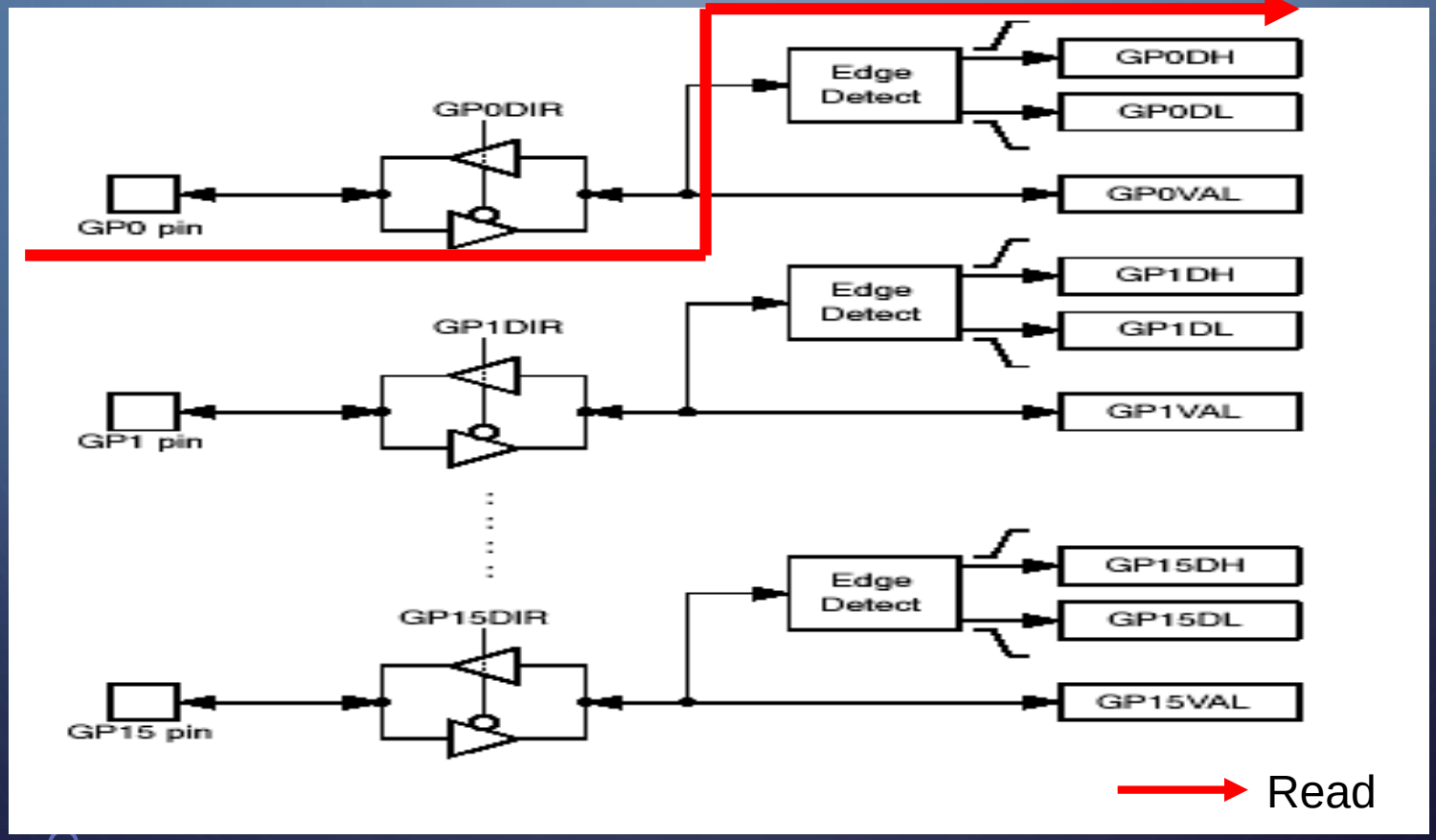
General Purpose Input Output (通用型之輸入輸出接口,GPIO)，在嵌入式系統中，常常有很多結構比較簡單的外部設備或電路，對於這些設備或電路來說，有的需要嵌入式平台的CPU為它提供控制，有的則需要當作輸入信號。且許多這樣的設備/電路只要求一個位元，也就是說，只要有開與關兩種狀態就夠了，比如LED的亮與滅。在實現這些控制時，使用傳統的串列埠或並列埠都不合適。所以在嵌入式平台微算機上一般都會提供GPIO。有無GPIO接口也是嵌入式平台微處理器區別於微型處理器的重要特徵之一。

GPIO簡介

- GPIO可作為輸入與輸出
- 使用輸出功能(out)時
  - 可分別輸入High(1)或Low(0)
- 使用輸入功能(in)時
  - 可方便地讀出外部輸入的高低電壓
- 所有的配置均可通過暫存器的配置快速實現

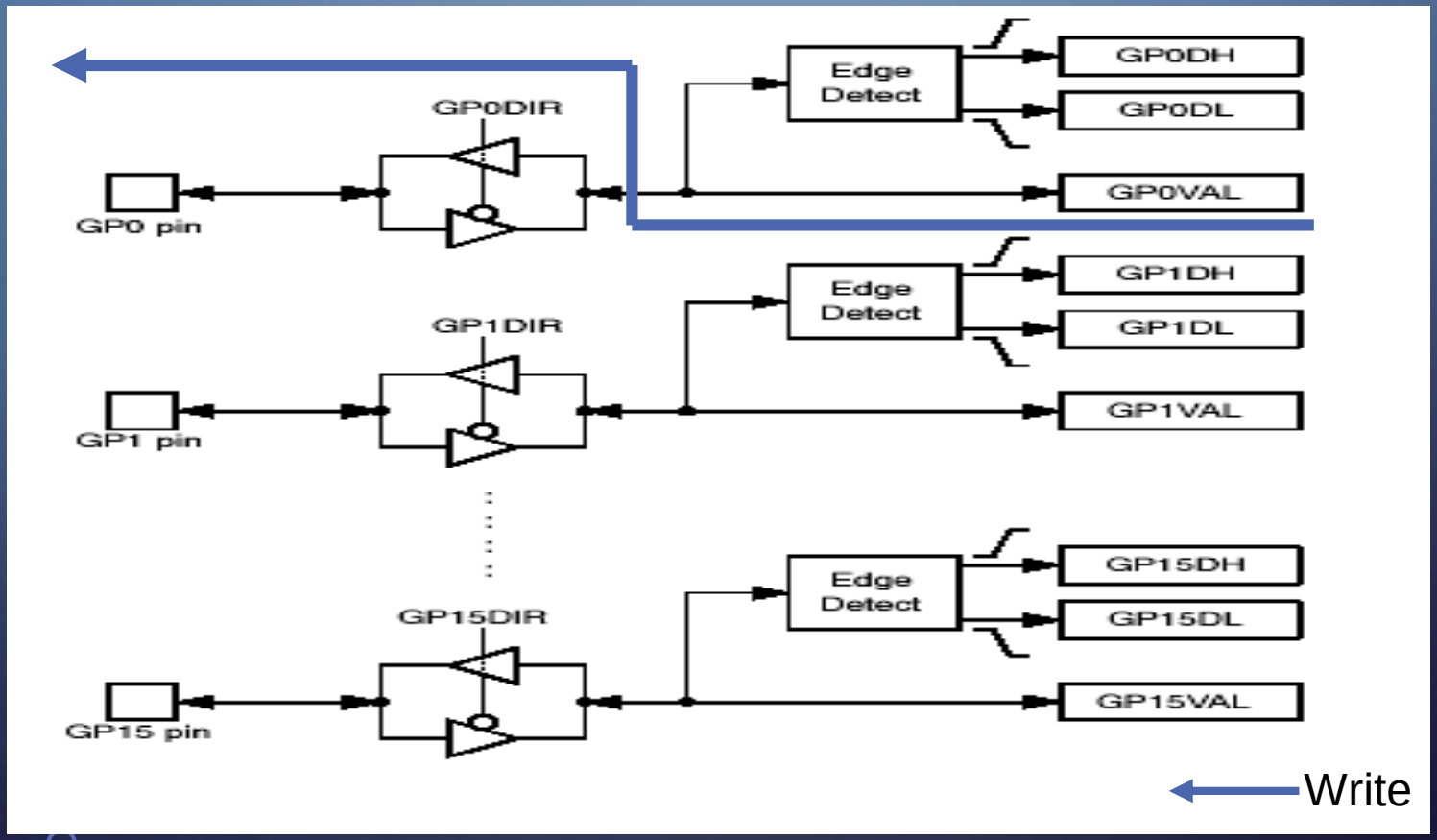
GPIO簡介

- 功能性GPIO之圖例(Read)



GPIO簡介

- 功能性GPIO之圖例(Write)



GPIO簡介

- GPIO訊號至少有兩種暫存器
  - 通用IO控制暫存器
  - 通用IO數據暫存器
- 控制暫存器用來控制GPIO通訊埠的信號方向，用來決定當前通訊埠處於輸出狀態(out)還是輸入狀態(in)
- 數據暫存器中的每一位元都對應著一個GPIO接腳的狀態。
  - 當接口為輸出狀態時，可以通過數據暫存器的寫入來控制接口輸出電壓高低狀態
  - 當接口為輸入狀態時，可以通過數據暫存器讀入當前接口的電壓高低狀態

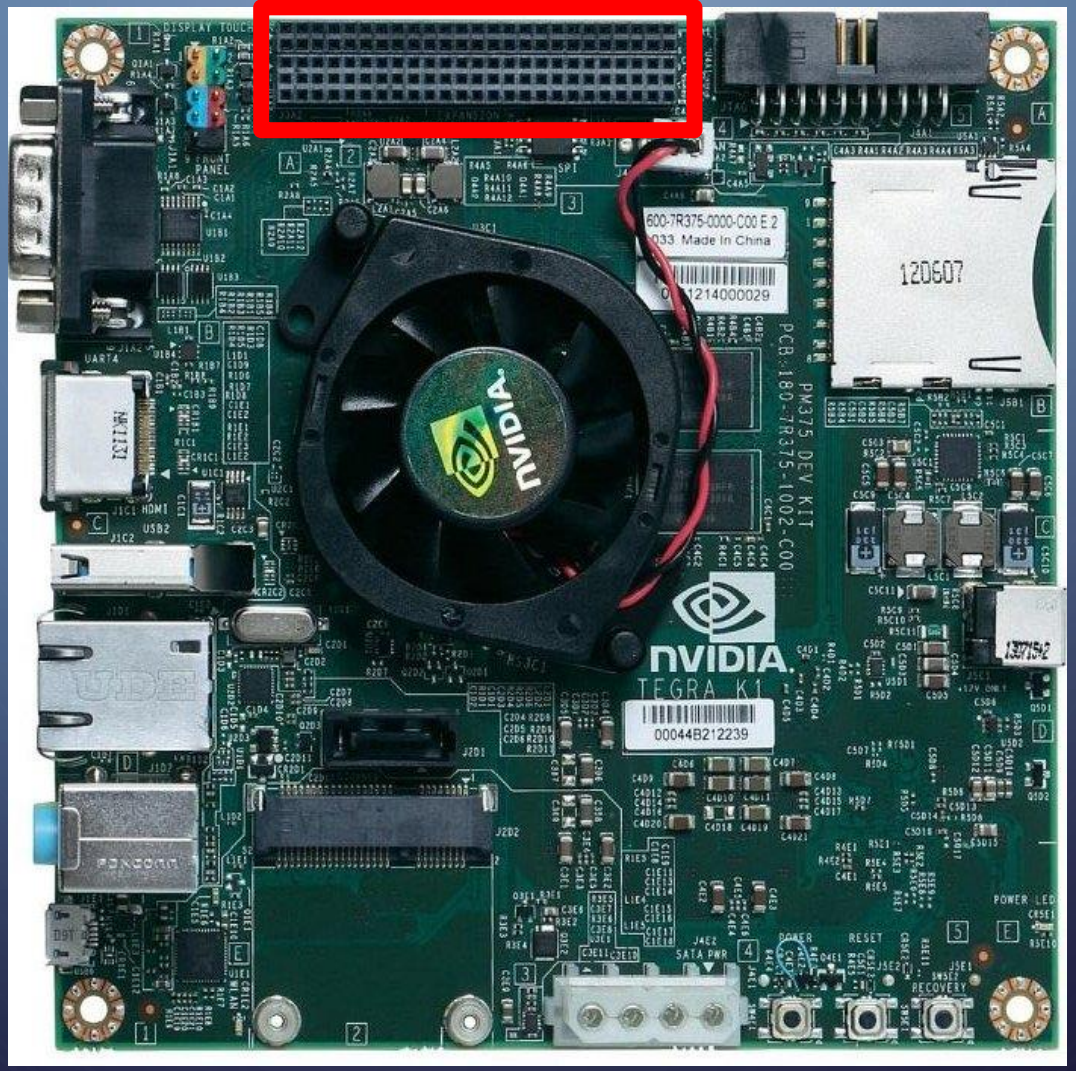
GPIO簡介

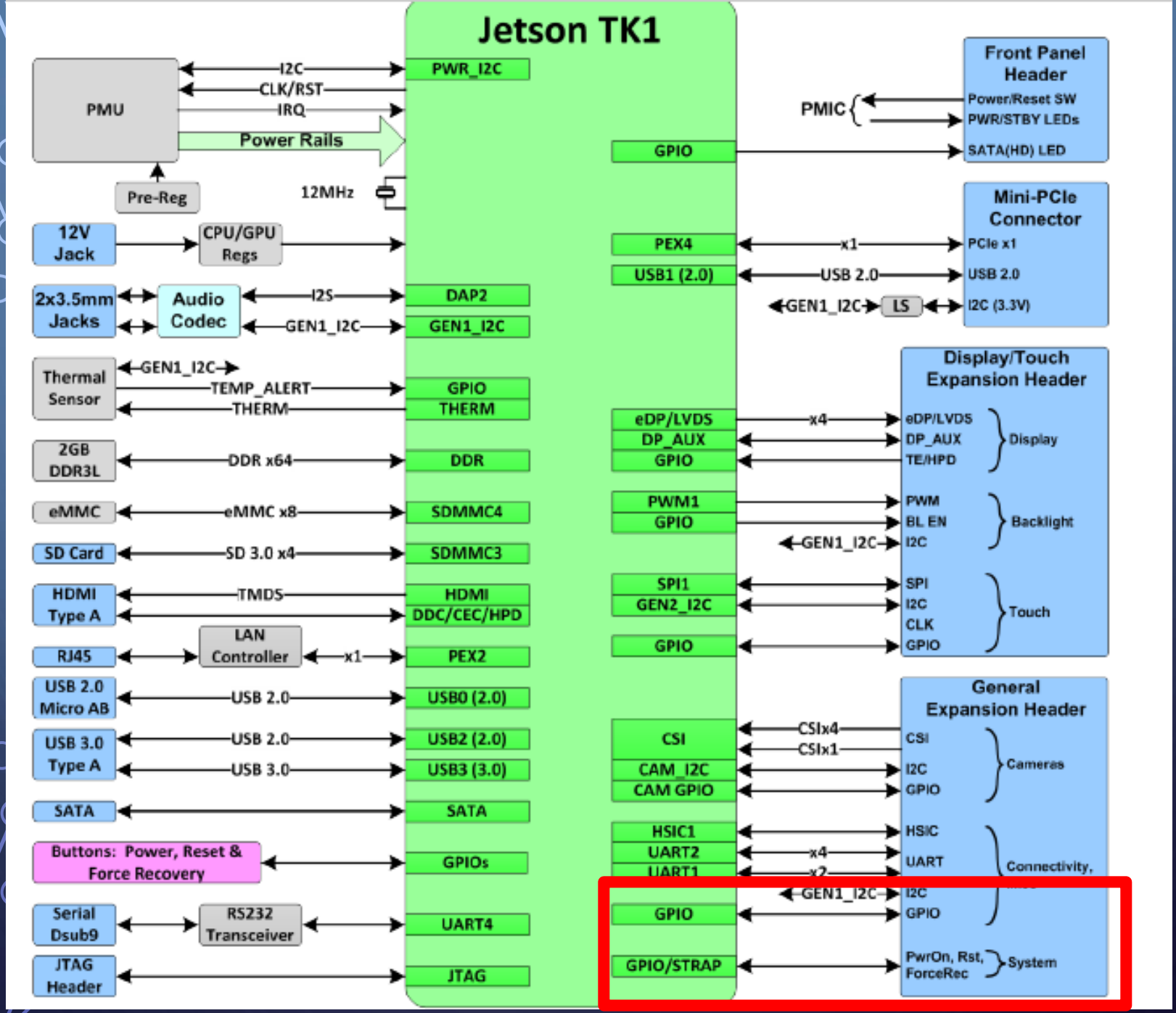
- GPIO的使用非常廣泛。用戶可以通過GPIO通訊埠與硬體進行數據交換、控制硬體(如LED、蜂鳴器、鏡頭等)工作、讀取硬體的工作狀態信號(如中斷信號)等。
- 一些傳輸協議相對簡單的低速匯流排(如I²C、SPI匯流排等)，也可以通過軟件控制一組GPIO通訊埠，從而模擬匯流排傳輸協議的通信。
- GPIO通訊埠是嵌入式平台中應用最廣泛和最靈活的接口之一。

GPIO簡介

- 部份GPIO亦能夠在使用其他用途
  - 通常較為價廉的解決方案，即是利用特定的控制器與特定的pin腳來滿足不同的需求與用途
- 例子
  - 記憶體介面
  - I²C, SPI 介面
  - UART 信號
  - Timer 輸出
  - LCD 信號
  - 外部中斷

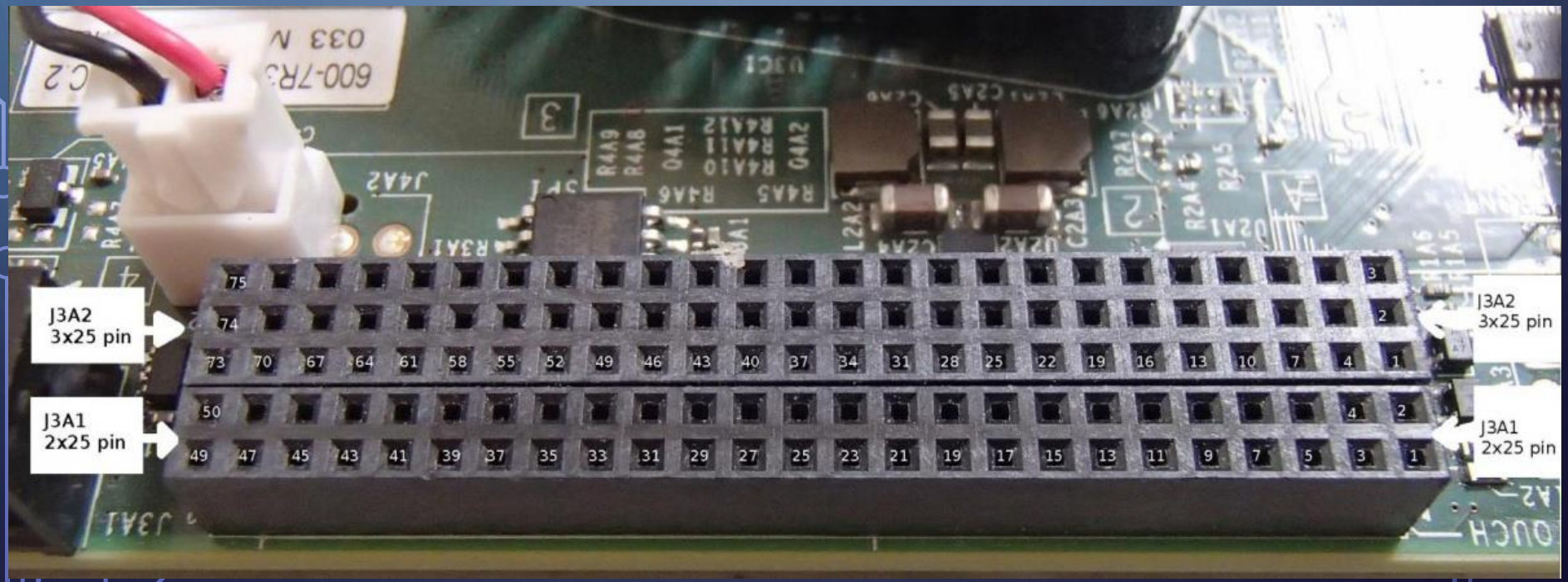
TK1 的 GPIO 腳位介紹




Jetson TK1
PMU
I2C
CLK/RST
IRQ
Power Rails
Pre-Reg
12MHz
PWR_I2C
12V Jack
CPU/GPU Regs
2x3.5mm Jacks
Audio Codec
I2S
GEN1_I2C
DAP2
GEN1_I2C
Thermal Sensor
GEN1_I2C
TEMP_ALERT
THERM
GPIO
THERM
2GB DDR3L
DDR x64
DDR
eMMC
eMMC x8
SDMMC4
SD Card
SD 3.0 x4
SDMMC3
HDMI Type A
TMDS
HDMI
DDC/CEC/HPD
RJ45
LAN Controller
x1
PEX2
USB 2.0 Micro AB
USB 2.0
USB0 (2.0)
USB 3.0 Type A
USB 2.0
USB2 (2.0)
USB 3.0
USB3 (3.0)
SATA
SATA
Buttons: Power, Reset & Force Recovery
GPIOs
Serial Dsub9
RS232 Transceiver
UART4
JTAG Header
JTAG
GPIO
PEX4
x1
USB1 (2.0)
USB 2.0
GEN1_I2C
LS
eDP/LVDS
x4
DP_AUX
GPIO
PWM1
GPIO
GEN1_I2C
SPI1
GEN2_I2C
GPIO
CSI
CSIx4
CSIx1
CAM_I2C
CAM GPIO
HSIC1
UART2
x4
UART1
x2
GEN1_I2C
GPIO
GPIO/STRAP
Front Panel Header
PMIC
Power/Reset SW
PWR/STBY LEDs
SATA(HD) LED
Mini-PCIe Connector
PCIe x1
USB 2.0
I2C (3.3V)
Display/Touch Expansion Header
eDP/LVDS
DP_AUX
TE/HPD
Display
PWM
BL EN
I2C
Backlight
SPI
I2C
CLK
GPIO
Touch
General Expansion Header
CSI
I2C
GPIO
Cameras
HSIC
UART
Connectivity,
I2C
GPIO
PwrOn, Rst, ForceRec
System

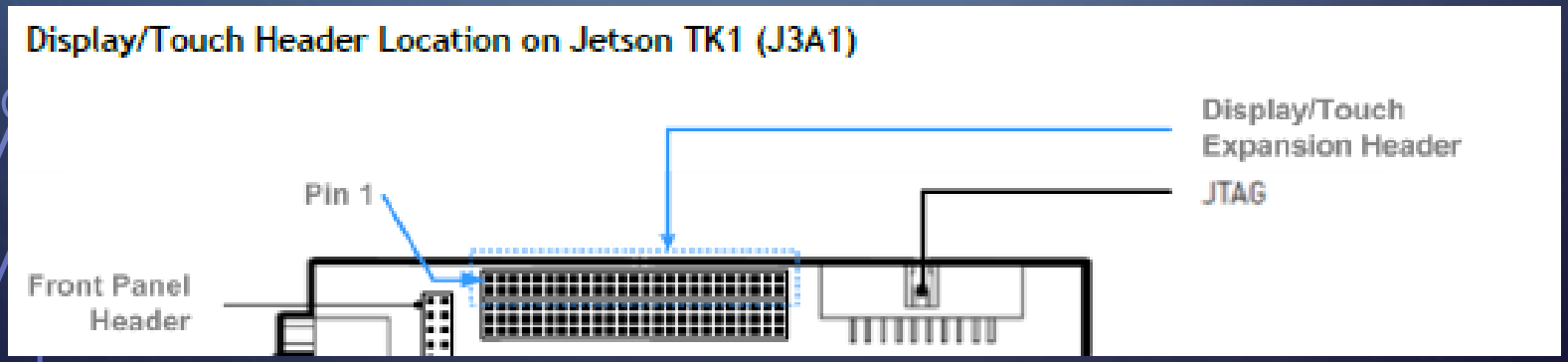
TK1 的 GPIO 腳位介紹

- TK1上的GPIO主要分為兩大區塊
  - J3A1(靠近外側兩排一組)
  - J3A2(靠近內側三排一組)



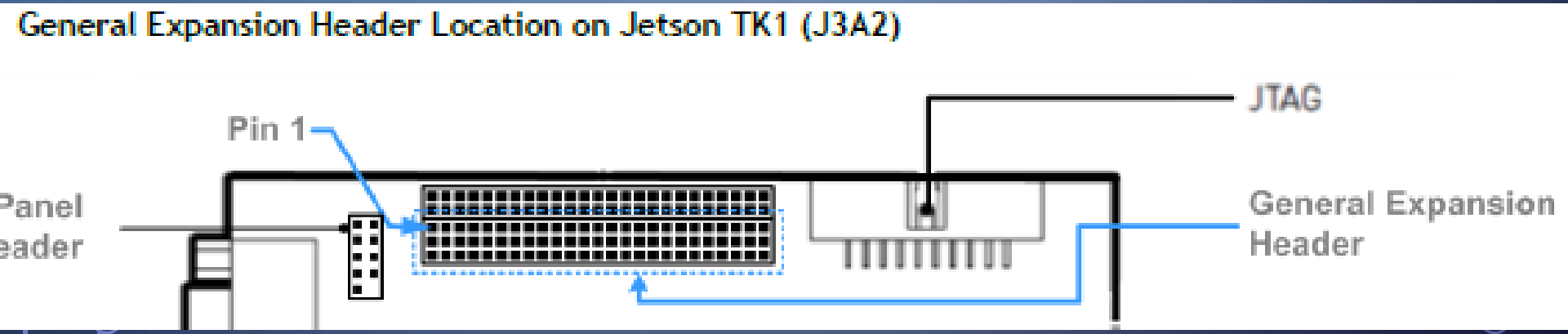
TK1 的 GPIO 腳位介紹

- TK1上的GPIO並非所有的GPIO腳位都讓使用者自定in/out與控制使用
- TK1只提共八個腳位可以讓使用者自行定義，在J3A1 一個、J3A2七個
- J3A1兩排GPIO其實主要是設計給觸控螢幕使用



TK1 的 GPIO 腳位介紹

- J3A2三排GPIO設計給一般通用型擴增接口，但實際只有7個接腳可以由使用者自行定義控制



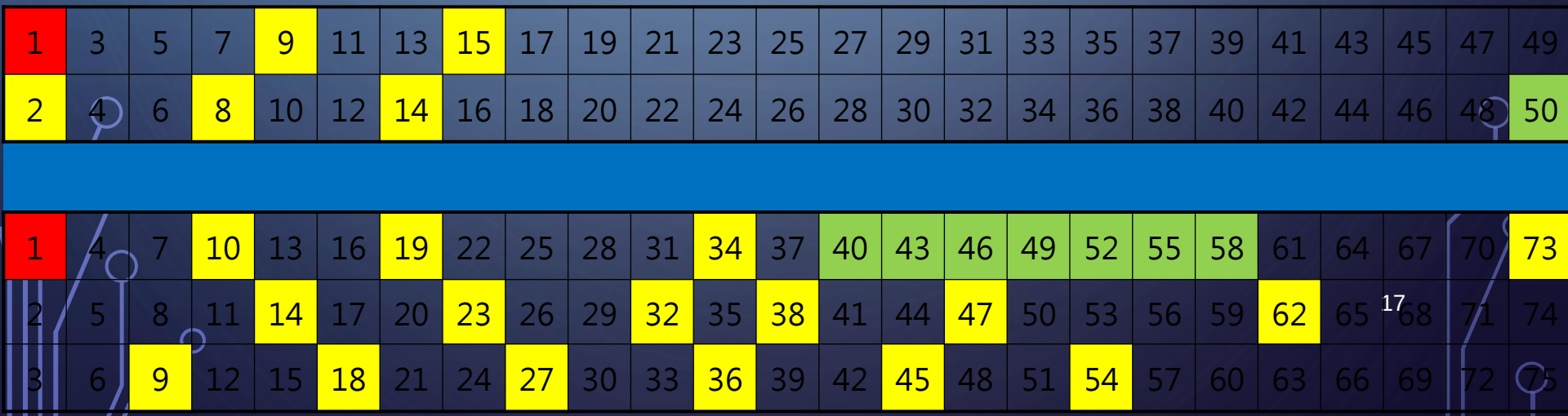
TK1 的 GPIO 腳位介紹

- 可自訂的接角列表
- sysfs filename很重要，Ubuntu對應的gpio編號是依照這個

Port	sysfs filename	Physical pin	Notes
GPIO_PU0	gpio160	Pin 40 on J3A2	
GPIO_PU1	gpio161	Pin 43 on J3A2	
GPIO_PU2	gpio162	Pin 46 on J3A2	(Disabled by default)
GPIO_PU3	gpio163	Pin 49 on J3A2	
GPIO_PU4	gpio164	Pin 52 on J3A2	
GPIO_PU5	gpio165	Pin 55 on J3A2	
GPIO_PU6	gpio166	Pin 58 on J3A2	
GPIO_PH1	gpio57	Pin 50 on J3A1	

TK1 的 GPIO 腳位介紹

- 接腳位置編號(上面兩排為J3A1、下面兩排為J3A2)
- 綠色為可自定義腳位(High: 1.8V)
- 黃色為接地(GND)
- 紅色為固定VCC腳位(5V)
- J3A1與J3A2請勿跨接



TK1 的 GPIO 腳位介紹

- 詳細GPIO接腳說明可以參考
  - 網頁: http://elinux.org/Jetson/GPIO
  - TK1開發手冊(之後會另外提供於課程網站)

UBUNTU 操作 TK1 上的 GPIO

- 在Ubuntu上已於Kernel層將GPIO的操作包裝過了，使用者無法使用ioctl這類的方式直接存取GPIO
- 需要透過/sys/class/gpio/下的export先啟用指定的GPIO Pin (Ubuntu只認識sysfs filename的編號)
- 啟用的sysfs filename會產生一個資料夾，編輯裡面的文件可以對對應的GPIO Pin產生相對應的控制

UBUNTU 操作 TK1 上的 GPIO

1. 操作前必須為Root才能取得控制權

```
$ sudo su
```

2. 進入/sys/class/gpio
   - # cd /sys/class/gpio/
3. 使用export啟用一個GPIO Pin，這裡示範pin 58 (sysfs filename: gpio166)
   - 指令: echo (sysfs filename編號) > export

```
root@tegra-ubuntu:/sys/class/gpio# echo 166 > export
```

UBUNTU 操作 TK1 上的 GPIO

3. 查看/sys/class/gpio/的目錄下是否有剛剛啟用的sysfs filename對應資料夾(範例為gpio166，對應pin腳為58)

```
root@tegra-ubuntu:/sys/class/gpio# ls
export   gpio143   gpio166   gpiochip0_ gpiochip1016   unexport
```

4. 進入剛剛產生出來的sysfs filename對應資料夾，並且查看其資料夾下的內容

```
root@tegra-ubuntu:/sys/class/gpio# cd gpio166
root@tegra-ubuntu:/sys/class/gpio/gpio166# ls
active_low  device  direction  edge  power  subsystem  uevent  value
```

UBUNTU 操作 TK1 上的 GPIO

- 主要會使用到direction與value
  - direction:用來設定in/out
  - value: 用來設定1/0
- 設定請用 echo (內容 ex: out) > (檔案 ex: direction)

```
root@tegra-ubuntu:/sys/class/gpio# cd gpio166
root@tegra-ubuntu:/sys/class/gpio/gpio166# ls
active_low  device  direction  edge  power  subsystem  uevent  value
```

UBUNTU 操作 TK1 上的 GPIO

5. 步驟五:將剛剛啟用的sysfs filename(範例為gpio166)資料夾下的direction設定為out，並且用cat查看direction內容是否已經為out

```
root@tegra-ubuntu:/sys/class/gpio/gpio166# echo out > direction
root@tegra-ubuntu:/sys/class/gpio/gpio166# cat direction
out
```

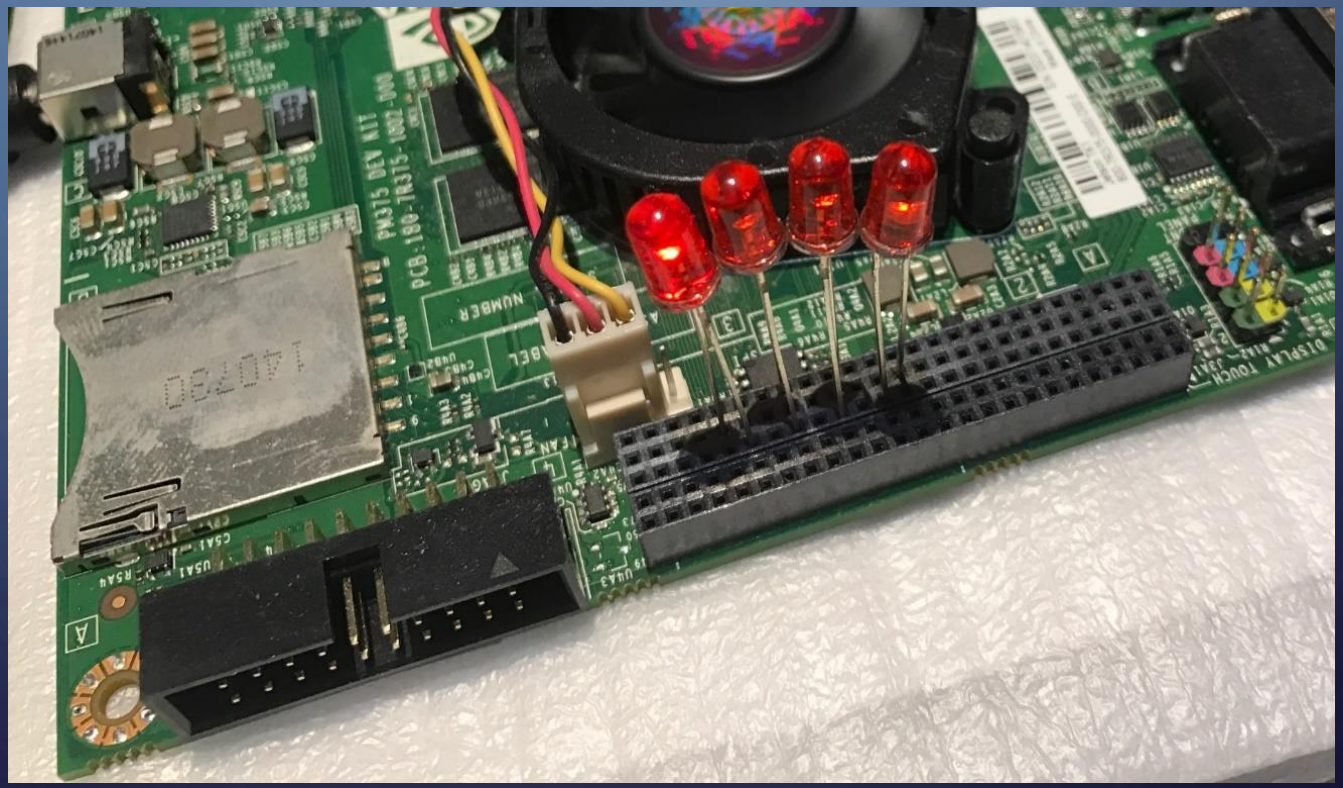
UBUNTU 操作 TK1 上的 GPIO

6. 將剛剛啟用的sysfs filename (範例為gpio166)資料夾下的value設定為1，並且用cat查看value內容是否已經為1

```
root@tegra-ubuntu:/sys/class/gpio/gpio166# echo 1 > value
root@tegra-ubuntu:/sys/class/gpio/gpio166# cat value
1
```

UBUNTU 操作 TK1 上的 GPIO

- 若已完成上面的操作步驟，則sysfs filename(範例為gpio166)對應的Pin腳(範例為Pin 58)狀態為1，輸出為1.8v，若有接LED可以看到該腳位的LED是亮的



UBUNTU 操作 TK1 上的 GPIO

7. 若要停用剛剛啟用的GPIO需要使用/sys/class/gpio/目錄下的unexport
   - 指令: echo (sysfs filename編號) > unexport

```
root@tegra-ubuntu:/sys/class/gpio# ls
export  gpio143  gpio166  gpiochip0  gpiochip1016  unexport
root@tegra-ubuntu:/sys/class/gpio# echo 166 > unexport
root@tegra-ubuntu:/sys/class/gpio# ls
export  gpio143  gpiochip0  gpiochip1016  unexport
```

以 C++ 透過 UBUNTU 操作 TK1 上的 GPIO

- 因為Ubuntu已於Kernel層將GPIO的操作封裝過了，使用者只能從上面範例提到的/sys/class/gpio/底下的介面來使用GPIO，所以在C/C++上我們不能使用ioctl的方式直接控制硬體
- 要以C/C++控制GPIO必須透過開檔/關檔/寫檔/讀檔的方式存取/sys/class/gpio/底下的介面來操作GPIO
- 以下為使用 export unexport direction value 的 C++ function範例code(僅供參考實際情形請按照使用環境需求改寫)

以 C++ 透過 UBUNTU 操作 TK1 上的 GPIO

- export

```
/****************************************************************
 * gpio_export 啟用GPIO
 ****************************************************************/
int gpio_export(unsigned int gpio) //傳入gpio的編號
{
    int fd, len;
    char buf[64];

    fd = open("/sys/class/gpio/export", O_WRONLY); //對/sys/class/gpio/export此程式做開檔讀寫
    if (fd < 0) {
        perror("gpio/export"); //開檔失敗
        return fd;
    }

    len = snprintf(buf, sizeof(buf), "%d", gpio); //將應啟用的gpio編號放入buf變數中
    write(fd, buf, len); //寫入export(啟用此gpio)
    close(fd); //關檔

    return 0;
}
```

* snprintf函式為格式化字串內容使用，在後方會再深入說明！

以 C++ 透過 UBUNTU 操作 TK1 上的 GPIO

- unexport

```
/****************************************************************
 * gpio_unexport 關閉GPIO
 ****************************************************************/
int gpio_unexport(unsigned int gpio) //傳入gpio的編號
{
    int fd, len;
    char buf[64];

    fd = open("/sys/class/gpio/unexport", O_WRONLY); //對/sys/class/gpio/unexport此程式做開檔讀寫
    if (fd < 0) {
        perror("gpio/export"); //開檔失敗
        return fd;
    }

    len = snprintf(buf, sizeof(buf), "%d", gpio); //將應要關閉的gpio編號放入buf變數中
    write(fd, buf, len); //寫入unexport(關閉此gpio)
    close(fd); //關檔
    return 0;
}
```

以 C++ 透過 UBUNTU 操作 TK1 上的 GPIO

- 設定direction

```
/****************************************************************
 * gpio_set_dir 設定GPIO輸入或輸出
 ****************************************************************/
int gpio_set_dir(unsigned int gpio, string dirStatus) //傳入參數分別為gpio編號與欲改為out或in
{
    int fd;
    char buf[64];

    //使用snprintf將字串組合
    snprintf(buf, sizeof(buf),"/sys/class/gpio/gpio%d/direction", gpio);

    fd = open(buf, O_WRONLY);//對direction此程式做開檔讀寫
    if (fd < 0) {
        perror("gpio/direction"); //開檔失敗
        return fd;
    }

    if (dirStatus == "out")
        write(fd, "out", 4); //如果傳入的為out
    else
        write(fd, "in", 3); //如果傳入的不是為out(代表要為in)

    close(fd); //關檔
    return 0;
}
```

以 C++ 透過 UBUNTU 操作 TK1 上的 GPIO

- 設定value

```
/****************************************************************
 * gpio_set_value 設定gpio的值為1或0
 ****************************************************************/
int gpio_set_value(unsigned int gpio, int value) //傳入參數分別為gpio編號與欲改為狀態為1或0
{
    int fd;
    char buf[64];

    //使用snprintf將字串組合
    snprintf(buf, sizeof(buf),"/sys/class/gpio/gpio%d/value", gpio);

    fd = open(buf, O_WRONLY);//對value此程式做開檔讀寫
    if (fd < 0) {
        perror("gpio/set-value"); //開檔失敗
        return fd;
    }

    if (value == 0)
        write(fd, "0", 2); //如果傳入的狀態為0
    else
        write(fd, "1", 2); //如果傳入的狀態不是為0(代表為1)

    close(fd); //關檔
    return 0;
}
```

以 C++ 透過 UBUNTU 操作 TK1 上的 GPIO

- 可能需要include的library(僅供參考)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <fcntl.h>
#include <iostream>

using namespace std;
```

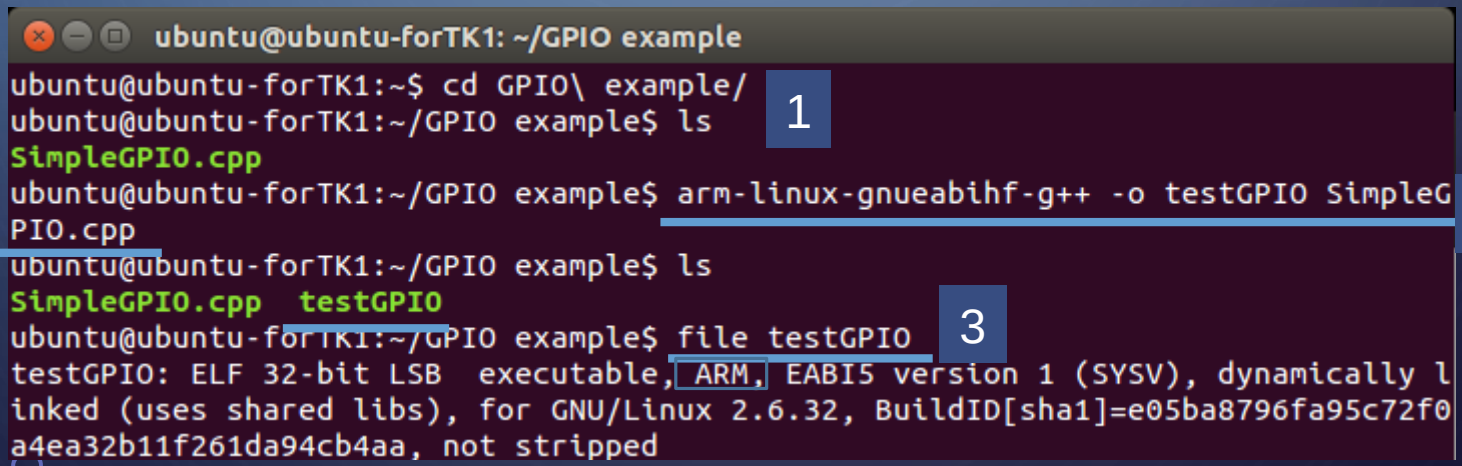
以 C++ 透過 UBUNTU 操作 TK1 上的 GPIO

- 範例主程式

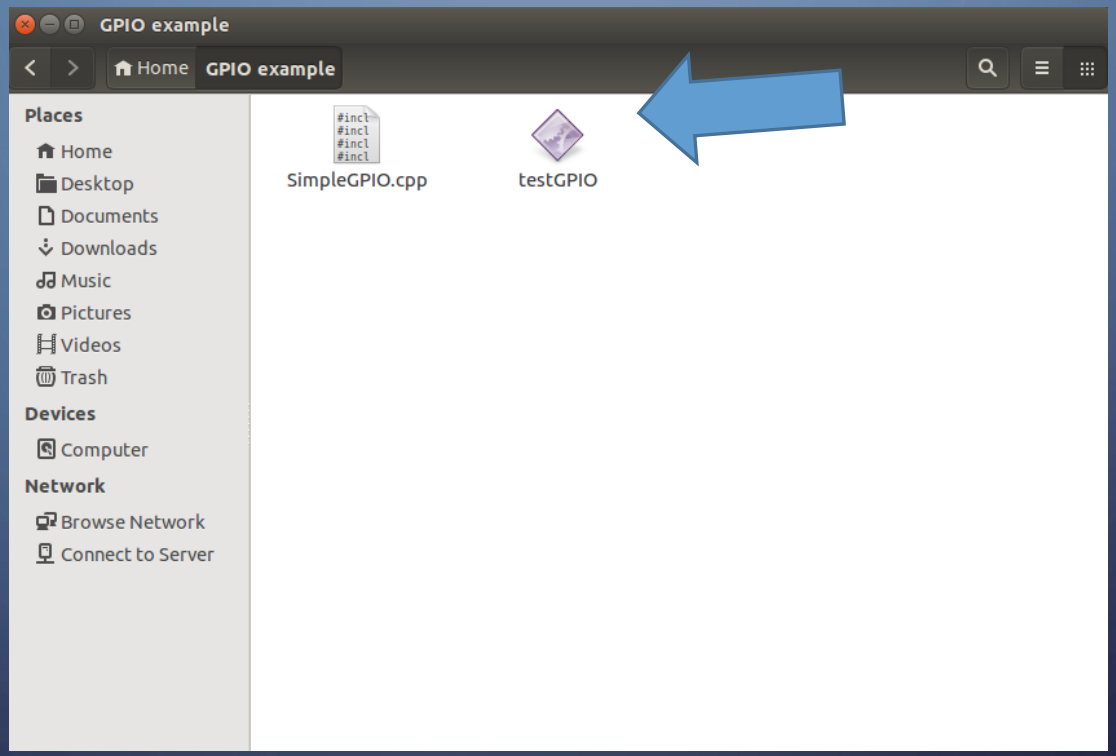
```
//使用範例
int main()
{
    int input;
    cout << "輸入1為啟用gpio166並設定狀態為1\n輸入2為將gpio166設定狀態為0並停用gpio166\n>>>";
    cin >> input;

    if (input == 1)
    {
        //啟用GPIO166
        gpio_export(166);
        //將此gpio166設定為輸出用途out
        gpio_set_dir(166,"out");
        //將此gpio166設定狀態為1
        gpio_set_value(166,1);
    }
    else if (input == 2)
    {
        //將此gpio166設定狀態為0
        gpio_set_value(166,0);
        //停用gpio166
        gpio_unexport(166);
    }
    return 0;
}
```

以 C++ 透過 UBUNTU 操作 TK1 上的 GPIO

- 跨平台編譯並傳到TK1上執行
  - 我們需要使用arm-linux-gnueabihf-g++對此程式編譯成可以在TK1上執行的程式
    1. 使用終端機將目錄移動到欲編譯的cpp檔位置
    2. 使用arm-linux-gnueabihf-g++ -o <輸出的執行檔名稱> <cpp檔案名稱>
    3. 可以用file查看編譯出來的執行檔是否為arm平台使用的

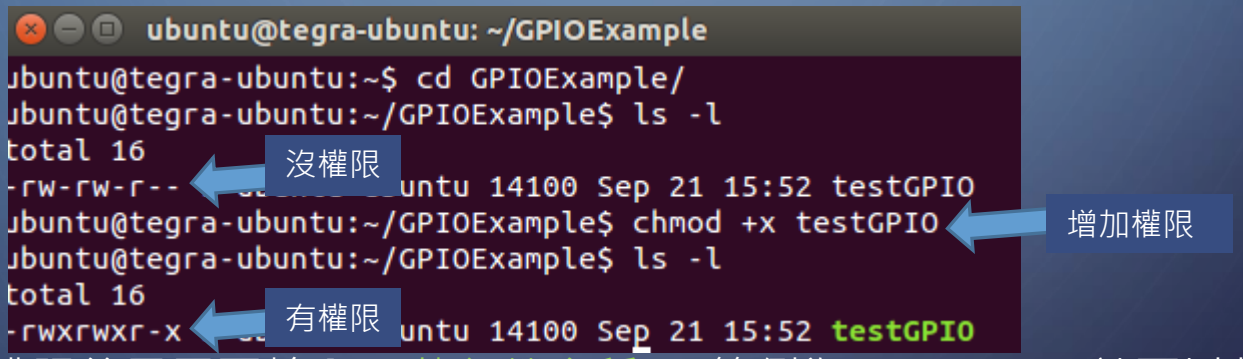


以 C++ 透過 UBUNTU 操作 TK1 上的 GPIO

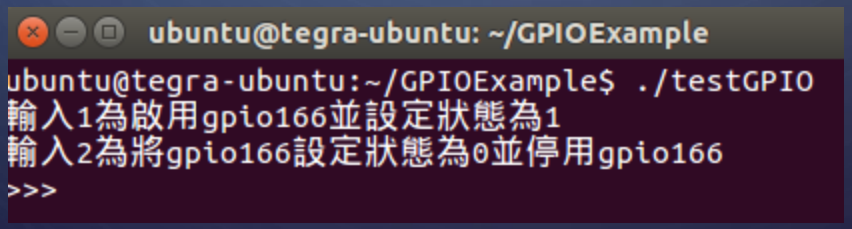


以 C++ 透過 UBUNTU 操作 TK1 上的 GPIO

- 跨平台編譯完成後可以透過前面所教的sftp將此執行檔(範例為:testGPIO)傳送到TK1嵌入式平台上執行
- 透過ssh進入TK1後移動到指定目錄找到剛剛船上去的程式，要執行該程式要先確認他有執行權限(x)，可以用 ls –l查看此執行檔是否有執行權限，若沒有可以使用chmod +x <執行檔名稱> 來增加此權限



- 有執行權限後只需要輸入./<執行檔名稱> (範例為:./testGPIO)就可以執行此程式了



以 C++ 透過 UBUNTU 操作 TK1 上的 GPIO

- 依照上面的方式執行此程式後，當我們輸入1會發現系統告訴我們Permission denied 原因是要控制GPIO必須要有root權限
- 解決方式
  - 方法1(推薦):使用sudo執行 ex: sudo ./testGPIO
  - 方法2(懶人版，但要小心):使用sudo su將自己改為root就可以直接執行

```
ubuntu@tegra-ubuntu:~/GPIOExample$ sudo ./testGPIO
[sudo] password for ubuntu:
輸入1為啟用gpio166並設定狀態為1
輸入2為將gpio166設定狀態為0並停用gpio166
>>>1
ubuntu@tegra-ubuntu:~/GPIOExample$ ls /sys/class/gpio/
export  gpio143  gpio166  gpiochip0  gpiochip1016  unexport
ubuntu@tegra-ubuntu:~/GPIOExample$ cat /sys/class/gpio/gpio166/value
1
```

```
ubuntu@tegra-ubuntu:~/GPIOExample$ sudo ./testGPIO
輸入1為啟用gpio166並設定狀態為1
輸入2為將gpio166設定狀態為0並停用gpio166
>>>2
ubuntu@tegra-ubuntu:~/GPIOExample$ ls /sys/class/gpio/
export  gpio143  gpiochip0  gpiochip1016  unexport
```

TK1 GPIO 相關參考資料

- snprintf
  - snprintf() 的功能是，就好像 printf()/fprintf()/sprintf() 一樣，給定一個 format specifier，以及額外的一些不定個數的參數，將會依序依指定的格式，填入 format specifier 裡面，以 % 開頭的欄位。printf() 和 fprintf() 會把結果，輸出到 STDOUT 或指定的 FILE stream，而 sprintf() 則是會把結果，填入第一個參數：一個 C-style 字串 buffer。然而，由於 sprintf() 在 protocol 設計上，沒有辦法讓實作 sprintf() 的程式庫，在被呼叫後，得知這個 buffer 的大小，因此，若結果比實際上的 buffer 還要長，就會造成 buffer overflow 的問題（所以我們範例中的buf變數宣告大小為64個字元大小）。因此，snprintf() 多出了第二個參數：buffer 的大小，以避免這個問題。
- 參考網站
  - http://blog.xuite.net/tzeng015/twblog/113272245-snprintf
  - http://www.cplusplus.com/reference/cstdio/snprintf/
  - https://wirelessr.gitbooks.io/working-life/content/snprintf_miao_wu_qiong.html

TK1 GPIO 相關參考資料

- 以下是TK1 GPIO相關參考資料，提供更進階的學習內容：
  - http://elinux.org/Jetson/GPIO
  - http://elinux.org/Jetson/Tutorials/GPIO
  - http://elinux.org/Jetson/Tutorials/Vision-controlled_GPIO
  - TK1 開發手冊（會放置於課程網站）