




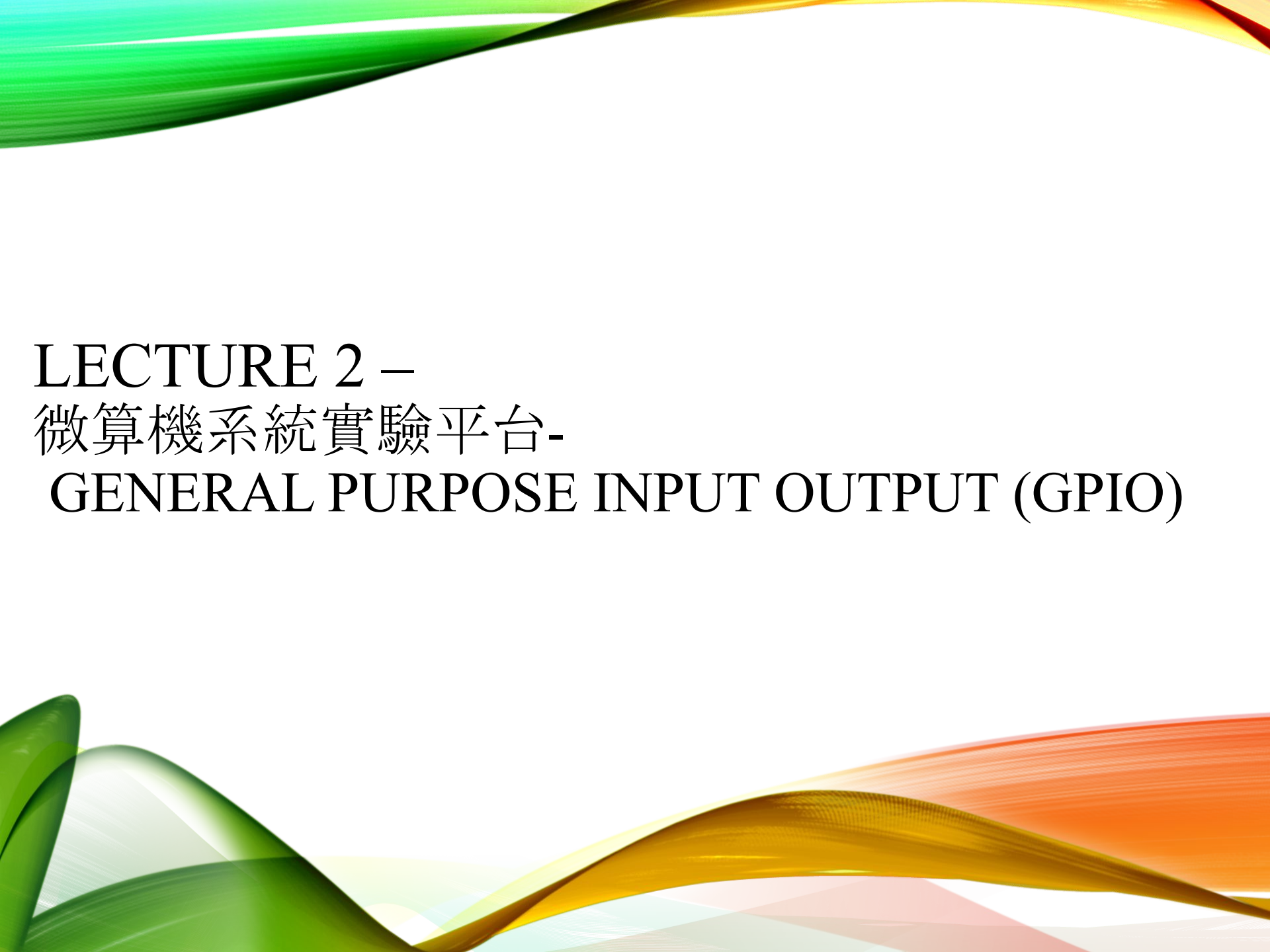
微算機系統實習 MICROPROCESSOR SYSTEMS LAB. SPRING, 2021

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 TX2 的 GPIO 腳位介紹
- 透過 Ubuntu 操作 TX2 上的 GPIO
- 以 C++ 透過 Ubuntu Kernel 操作 TX2 上的 GPIO
- TX2 GPIO 相關參考資料

GPIO簡介

General Purpose Input Output (通用型輸入輸出連接埠,GPIO), 在嵌入式系統中, 有很多結構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, 有的則只作為輸入信號。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連接埠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。

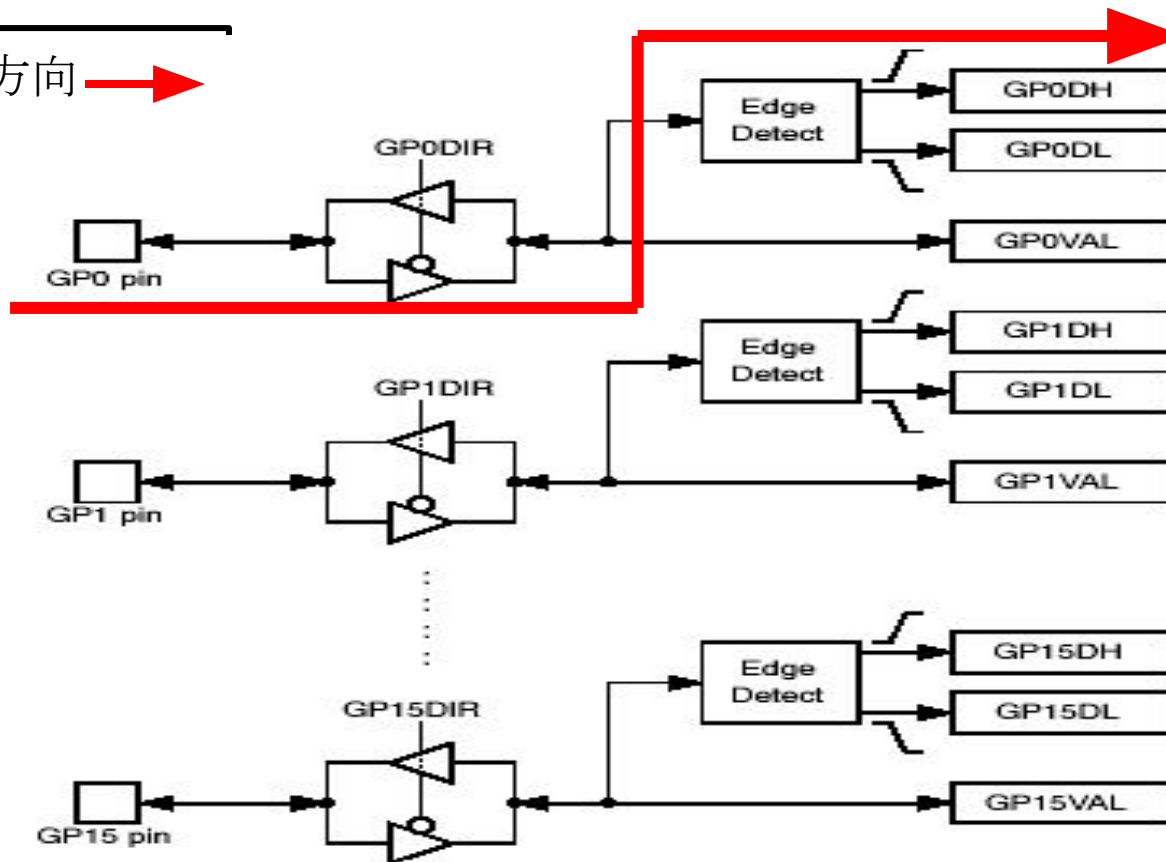
GPIO簡介

- 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

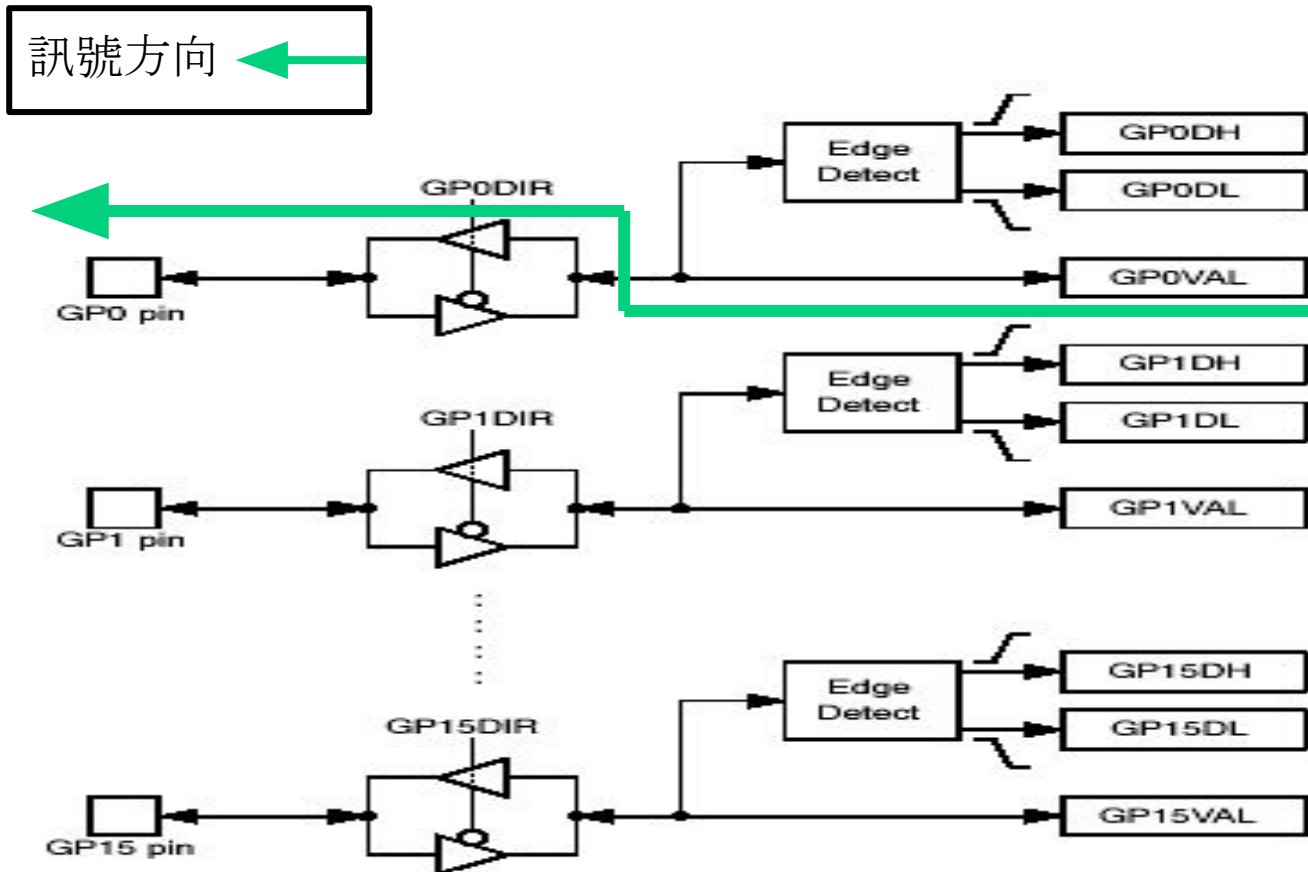
- 功能性GPIO之圖例(Read)

訊號方向 →



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(GPnDIR)
 - 通用IO位元暫存器
- 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連接埠的狀態。
 - 當連接埠為輸出(output)狀態時，可以通過位元暫存器的寫入來控制接口輸出電壓高低狀態
 - 當連接埠為輸入(input)狀態時，可以通過數據暫存器讀入當前連接埠的電壓高低狀態

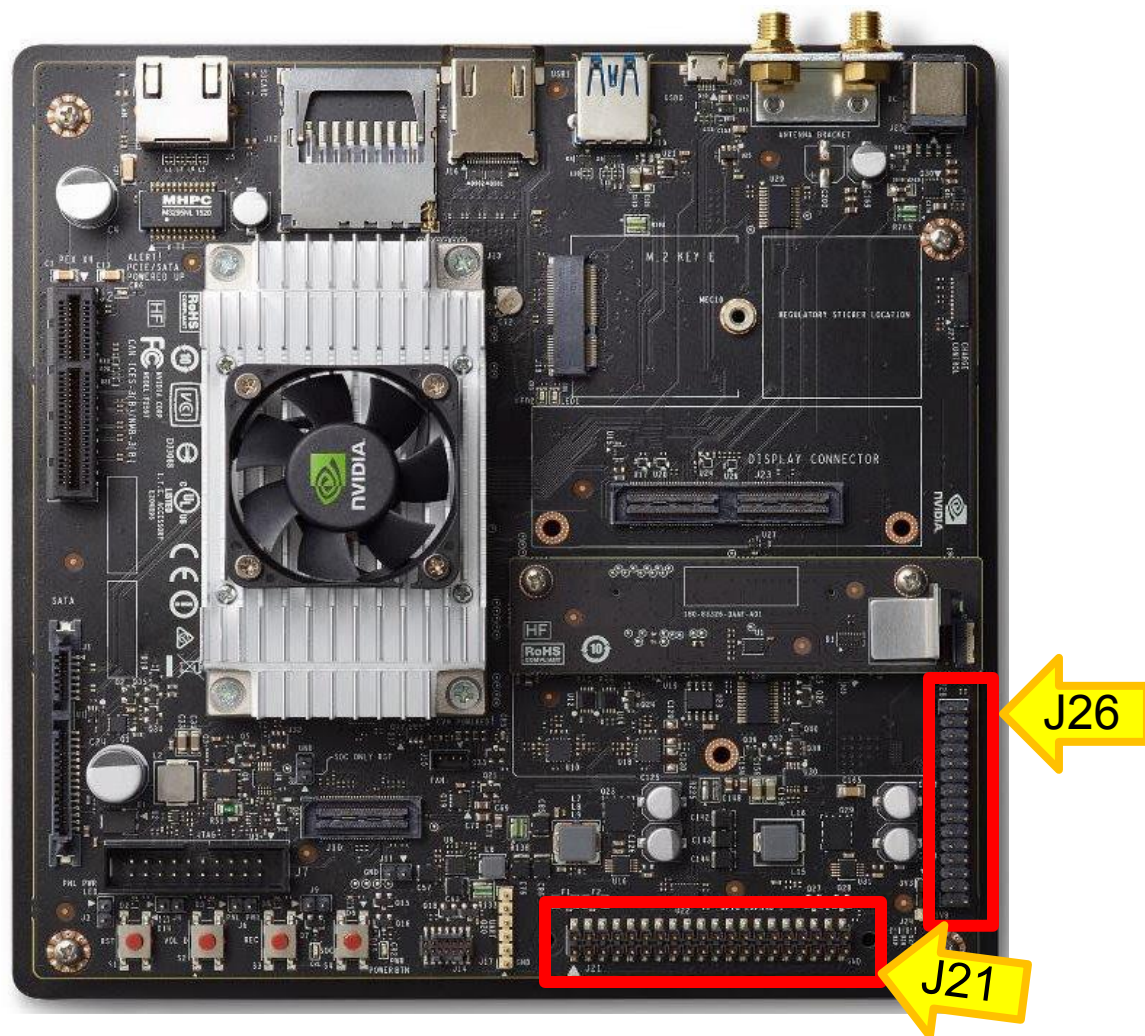
GPIO簡介

- GPIO的使用非常廣泛。使用者可以通過GPIO通訊埠與硬體進行資料交換、控制硬體(如LED、蜂鳴器、鏡頭等)工作、讀取硬體的工作狀態訊號(如中斷訊號)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訊。
- GPIO通訊埠是嵌入式平台中應用最廣泛和最靈活的連接埠之一。

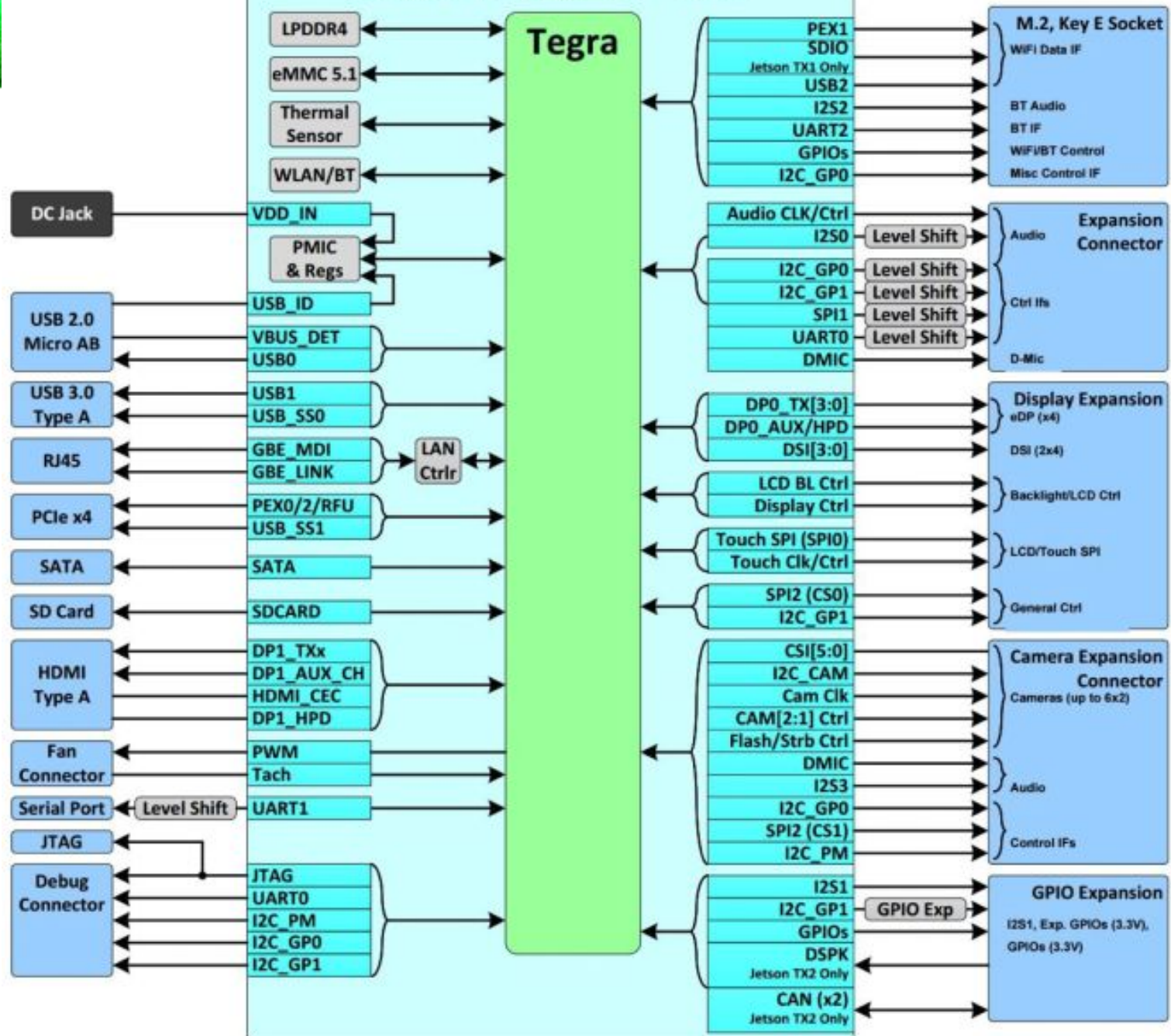
GPIO簡介

- 部份GPIO亦能夠在使用其他用途
 - 通常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

TX2 的 GPIO 腳位介紹

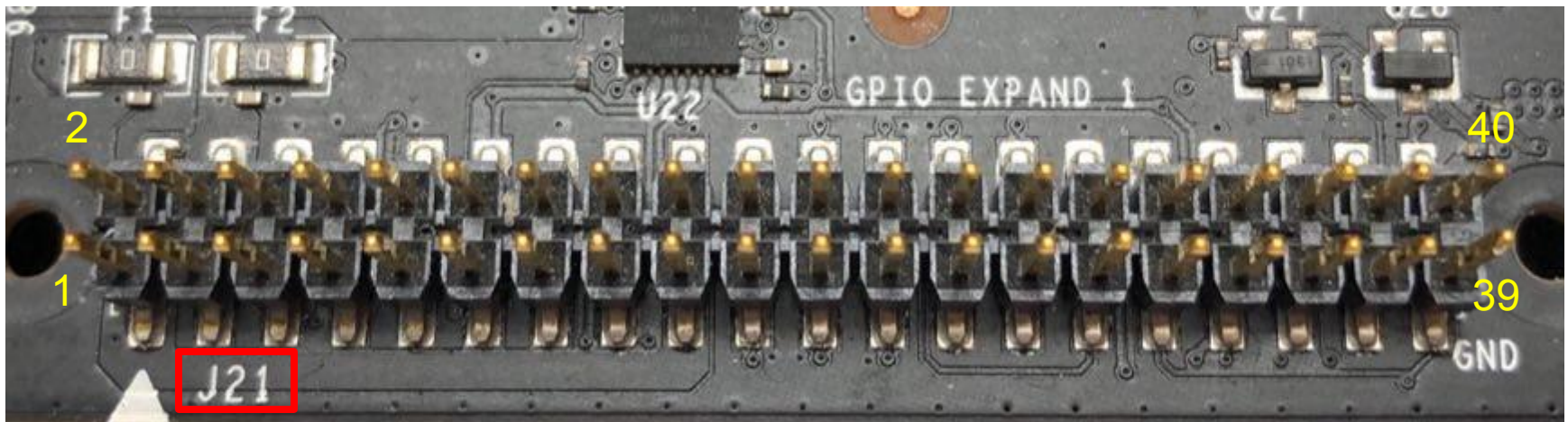


Jetson TX1 or TX2



TX2 GPIO HEADER PINOUT介紹

- TX2上的GPIO主要分為兩大區塊
 - J21 (實驗主要用J21的腳位)
 - J26



TX2 的 GPIO 腳位介紹(J21)

Jetson TX2 J21 Header					
Sysfs GPIO	Connector Label	Pin	Pin	Connector Label	Sysfs GPIO
	3.3 VDC Power	1	2	5.0 VDC Power	
	SDA1 General I2C Data 3.3V, I2C Bus 1	3	4	5.0 VDC Power	
	SCL1 General I2C Clock 3.3V, I2C Bus 1	5	6	GND	
gpio396	GPIO_GCLK Audio Master Clock (1.8/3.3V)	7	8	TXD0 UART #0 Transmit	
	GND	9	10	RXD0 UART #0 Receive	
gpio466	GPIO_GEN0 UART #0 Request to Send	11	12	GPIO_GEN1 Audio I2S #0 Clock	gpio392
gpio397	GPIO_GEN2 Audio Code Interrupt	13	14	GND	
gpio255	GPIO_GEN3 From GPIO Expander (P17)	15	16	GPIO_GEN4 Unused	gpio296
	3.3 VDC Power	17	18	GPIO_GEN5 Modem Wake AP GPIO	gpio481
gpio429	SPI_MOSI SPI #1 Master Out/Slave In	19	20	GND	

gpio428	SPI1_MISO SPI #1 Master In/Slave Out	21	22	GPIO_GEN6 From GPIO Expander (P16)	gpio254
gpio427	SPI_SCLK SPI #1 Shift Clock	23	24	SPI_CE0_N SPI Chip Select #0	gpio430
	GND	25	26	SPI_CE1_N SPI #1 Chip Select #1	
	ID_SD General I2C #1 Data (3.3V), I2C Bus 0	27	28	ID_SC General I2C #1 Clock (3.3V), I2C Bus 0	
gpio398	GPIO5 Audio Reset (1.8/3.3V)	29	30	GND	
gpio298	GPIO6 Motion Interrupt (3.3V)	31	32	GPIO12 Unused	gpio297
gpio389	GPIO13 AP Wake Bt GPIO	33	34	GND	
gpio395	GPIO19 AUDIO I2S #0 Left/Right Clock	35	36	GPIO16 UART #0 Clear to Send	gpio467
gpio388	GPIO26 (3.3V)	37	38	GPIO20 Audio I2S #0 Data in	gpio394
	GND	39	40	GPIO21 Audio I2S #0 Data in	gpio393

TX2 GPIO EXPANSION HEADER PIN DESCRIPTIONS(J21)

Pin #	Signal Name	Associated Jetson Module Pin Name	Usage/Description	Type/Direction	Signal Voltage Level at Header	GPIO Max Drive (I _{OL} /I _{OH}) or Power Pin Current Capability	Notes
1	VDD_3V3_SYS	—	Main 3.3V Supply	Power	—	1A	1
2	VDD_5V0_IO_SYS	—	Main 5.0V Supply	Power	—	1A	1
3	I2C_GPO_SDA_3V3_LVL	I2C_GPO_DAT	General I2C #0 Data	Bidir/OD	3.3V	1mA / -1mA	2
4	VDD_5V0_IO_SYS	—	Main 5.0V Supply	Power	—	1A	1
5	I2C_GPO_SCL_3V3_LVL	I2C_GPO_CLK	General I2C #0 Clock	Bidir/OD	3.3V	1mA / -1mA	2
6	GND	—	Ground	Ground	—	—	—
7	AUDIO_I2S_MCLK_3V3	AUDIO_MCLK	Audio Master Clock	Bidir	1.8/3.3V	20uA / -20uA	3
8	UART1_TXD_HDR_3V3	UART0_TX	UART #0 Transmit	Output	3.3V	24mA / -24mA	4
9	GND	—	Ground	Ground	—	—	—
10	UART1_RXD_HDR_3V3	UART0_RX	UART #0 Receive	Input	3.3V	—	4
11	UART1_RTS_HDR_3V3	UART0_RTS#	UART #0 Request to Send	Output	3.3V	24mA / -24mA	4
12	AUDIO_I2S_SRCLK_3V3	I2S0_SCLK	Audio I2S #0 Clock	Bidir	1.8/3.3V	20uA / -20uA	3
13	AUDIO_CDC_IRQ_LVL	GPIO20_AUD_INT	Audio Codec Interrupt	Bidir	1.8/3.3V	20uA / -20uA	3
14	GND	—	Ground	Ground	—	—	—
15	GPIO_EXP_P17_3V3	—	From GPIO Expander (P17)	Bidir	3.3V	25mA / -10mA	5
16	AO_DMIC_IN_DAT_LVL	AO_DMIC_IN_DAT	Digital Mic Input	Input	1.8/3.3V	20uA / -20uA	3, 8
17	VDD_3V3_SYS	—	Main 3.3V Supply	Power	—	1A	1
18	MDM_WAKE_AP_LVL	GPIO16_MDM_WAKE_AP	Modem Wake AP GPIO	Input	1.8/3.3V	20uA / -20uA	3, 8
19	SPI1_MOSI_3V3	SPI1_MOSI	SPI #1 Master Out/Slave In	Bidir	1.8/3.3V	20uA / -20uA	3
20	GND	—	Ground	Ground	—	—	—
21	SPI1_MISO_3V3	SPI1_MISO	SPI #1 Master In/Slave Out	Bidir	1.8/3.3V	20uA / -20uA	3
22	GPIO_EXP_P16_3V3	—	From GPIO Expander (P16)	Bidir	3.3V	25mA / -10mA	5
23	SPI1_SCK_3V3	SPI1_CLK	SPI #1 Shift Clock	Bidir	1.8/3.3V	20uA / -20uA	3
24	SPI1_CS0_3V3	SPI1_CS0#	SPI #1 Chip Select #0	Bidir	1.8/3.3V	20uA / -20uA	3
25	GND	—	Ground	Ground	—	—	—
26	SPI1_CS1_3V3	SPI1_CS1#	SPI #1 Chip Select #1	Bidir	1.8/3.3V	20uA / -20uA	3
27	I2C_GP1_DAT_3V3	I2C_GP1_DAT	General I2C #1 Data	Bidir/OD	3.3V	1mA / -1mA	6
28	I2C_GP1_CLK_3V3	I2C_GP1_CLK	General I2C #1 Clock	Bidir/OD	3.3V	1mA / -1mA	6
29	AUD_RST_LVL	GPIO19_AUD_RST	Audio Reset	Output	1.8/3.3V	20uA / -20uA	3, 8
30	GND	—	Ground	Ground	—	—	—
31	MOTION_INT_AP_L_LVL	GPIO9_MOTION_INT	Motion Interrupt	Input/OD	3.3V	1mA / -1mA	2, 8
32	AO_DMIC_IN_CLK_LVL	AO_DMIC_IN_CLK	Digital Mic Clock	Output	1.8/3.3V	20uA / -20uA	3, 8
33	AP_WAKE_BT_3V3	GPIO11_AP_WAKE_BT	AP Wake Bt GPIO	Bidir	1.8/3.3V	20uA / -20uA	3, 8
34	GND	—	Ground	Ground	—	—	—
35	AUDIO_I2S_SFSYNC_3V3	I2S0_LRCLK	AUDIO I2S #0 Left/Right Clock	Bidir	1.8/3.3V	20uA / -20uA	3
36	UART1_CTS_HDR_3V3	UART0_CTS#	UART #0 Clear to Send	Input	3.3V	—	4
37	SAR_TOUT_LVL	GPIO8_ALS_PROX_INT	Accelerometer/Proximity Interrupt	Output/OD	3.3V	1mA / -1mA	2, 8
38	AUDIO_I2S_SIN_3V3	I2S0_SDIN	Audio I2S #0 Data In	Input	1.8/3.3V	20uA / -20uA	3, 8
39	GND	—	Ground	Ground	—	—	—
40	AUDIO_I2S_SOUT_3V3	I2S0_SDOOUT	Audio I2S #0 Data Out	Output	1.8/3.3V	20uA / -20uA	3, 8

Legend

Ground	Power	Not available on Jetson TX1	Not available on Jetson TX2	Reserved	Unassigned on carrier board
--------	-------	-----------------------------	-----------------------------	----------	-----------------------------

TX2 GPIO EXPANSION HEADER PIN DESCRIPTIONS(J26)

Pin #	Signal Name	Associated Jetson Module Pin Name	Usage/Description	Type/Direction	Signal Voltage Level at Header	GPIO Max Drive (IOL/IOH) or Power Pin Current Capability	Notes
1	CAN_WAKE	CAN_WAKE	CAN Wake	Output	3.3V	1mA / -1mA	2, 4
2	VDD_3V3_SYS	–	Main 3.3V Supply	Power	–	1A	1
3	CAN0_STBY	–	Unused	Unused	–	–	–
4	VDD_1V8	–	Main 1.8V Supply	Power	–	1A	1
5	CAN0_RX	CAN0_RX	CAN #0 Receive	Output	3.3V	1mA / -1mA	2, 4
6	AP2MDM_READY	GPIO15_AP2MDM_READY	AP to Modem Ready GPIO	Bidir	–	1mA / -1mA	2, 4
7	CAN0_TX	CAN0_TX	CAN #0 Transmit	Input	3.3V	1mA / -1mA	2, 4
8	VDD_5V0_IO_SYS	–	Main 5.0V Supply	Power	–	1A	1
9	CAN0_ERR	CAN0_ERR	CAN #0 Error	Output	3.3V	1mA / -1mA	2, 4
10	GND	–	Ground	Ground	–	–	–
11	GND	–	Ground	Ground	–	–	–
12	I2C_GP2_CLK	I2C_GP2_CLK	General I2C #2 Clock	Bidir/OD	1.8V	1mA / -1mA	2
13	CAN1_STBY	CAN1_STBY	CAN #1 Standby	Input	3.3V	1mA / -1mA	2, 4
14	I2C_GP2_DAT	I2C_GP2_DAT	General I2C #2 Data	Bidir/OD	1.8V	1mA / -1mA	2
15	CAN1_RX	CAN1_RX	CAN #1 Receive	Output	3.3V	1mA / -1mA	2, 4
16	WDT_TIME_OUT_L	WDT_TIME_OUT#	Watchdog Timer Output	Output	–	1mA / -1mA	2, 4
17	CAN1_TX	CAN1_TX	CAN #1 Transmit	Input	3.3V	1mA / -1mA	2, 4
18	I2C_GP3_CLK	I2C_GP3_CLK	General I2C #3 Clock	Bidir/OD	1.8V	1mA / -1mA	2
19	CAN1_ERR	CAN1_ERR	CAN #1 Error	Output	3.3V	1mA / -1mA	2, 4
20	I2C_GP3_DAT	I2C_GP3_DAT	General I2C #3 Data	Bidir/OD	1.8V	1mA / -1mA	2
21	GND	–	Ground	Ground	–	–	–
22	SLEEP	SLEEP#	Sleep Indicator	Output	1.8V	1mA / -1mA	2, 4
23	I2S1_CLK	I2S1_CLK	I2S #1 Clock	Bidir	1.8V	1mA / -1mA	2
24	I2S1_SDOUT	I2S1_SDOUT	I2S #1 Data Out	Bidir	1.8V	1mA / -1mA	2
25	I2S1_SDIN	I2S1_SDIN	I2S #1 Data In	Input	1.8V	1mA / -1mA	2, 4
26	I2S1_LRCLK	I2S1_LRCLK	I2S #1 Left/Right Clock	Bidir	1.8V	1mA / -1mA	2
27	DSPK_OUT_CLK	DSPK_OUT_CLK	Digital Speaker Out Clock	Output	1.8V	1mA	2, 4
28	GND	–	Ground	Ground	–	–	–
29	DSPK_OUT_DAT	DSPK_OUT_DAT	Digital Speaker Out Data	Output	1.8V	1mA	2, 4
30	GNSS_PSS	Reserved	–	–	–	–	–

Legend

Ground	Power	Not available on Jetson TX1	Not available on Jetson TX2	Reserved/Not Available
--------	-------	-----------------------------	-----------------------------	------------------------

TX2 的 GPIO 腳位介紹

- TX2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
- TX2只提供18個腳位(J21)可以讓使用者自行定義

TX2 的 GPIO 腳位介紹

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

sysfs filename	Physical pin
gpio396	Pin 7 on J21
gpio466	Pin 11 on J21
gpio392	Pin 12 on J21
gpio397	Pin 13 on J21
gpio255	Pin 15 on J21
gpio481	Pin 18 on J21
gpio429	Pin 19 on J21
gpio428	Pin 21 on J21

TX2 的 GPIO 腳位介紹

sysfs filename	Physical pin
gpio254	Pin 22 on J21
gpio427	Pin 23 on J21
gpio398	Pin 29 on J21
gpio298	Pin 31 on J21
gpio297	Pin 32 on J21
gpio389	Pin 33 on J21
gpio395	Pin 35 on J21
gpio388	Pin 37 on J21
gpio394	Pin 38 on J21
gpio393	Pin 40 on J21

TX2 的 GPIO 腳位介紹

- 接腳位置編號
- 綠色為可自定義腳位(High: 1.8V/3.3V)
- 黃色為接地(GND)

2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40
1	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31	33	35	37	39

TX2 的 GPIO 腳位介紹

- 詳細GPIO接腳說明可以參考

- 網頁:

<https://www.jetsonhacks.com/nvidia-jetson-tx2-j21-header-pinout/>

- TX2開發手冊

<https://forums.developer.nvidia.com/uploads/short-url/QhIPn8rdiD1Cp4w7ueh28eQpHR.pdf>

UBUNTU 操作 TX2 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 (input/output control)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 TX2 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pin15 (sysfs filename: gpio255)

- 指令: `echo (sysfs filename編號) > export`

```
root@tegra-ubuntu:/sys/class/gpio# echo 255 >export
```

UBUNTU 操作 TX2 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gpio255, 對應pin腳為15)

```
root@tegra-ubuntu:/sys/class/gpio# echo 255 >export
root@tegra-ubuntu:/sys/class/gpio# ls
export      gpiochip216  gpiochip240  gpiochip320
gpio255     gpiochip224  gpiochip256  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 gpio255
root@tegra-ubuntu:/sys/class/gpio/gpio255# ls
active_low  device  direction  power  subsystem  uevent  value
```

UBUNTU 操作 TX2 上的GPIO

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

UBUNTU 操作 TX2 上的 GPIO

5. 步驟五:將剛剛啟用的sysfs filename(範例為gpio255)資料夾下的direction設定為out, 並且用cat查看direction檔案內容是否已經為out

```
root@tegra-ubuntu:/sys/class/gpio/gpio255# echo out > direction
root@tegra-ubuntu:/sys/class/gpio/gpio255# cat direction
out
```

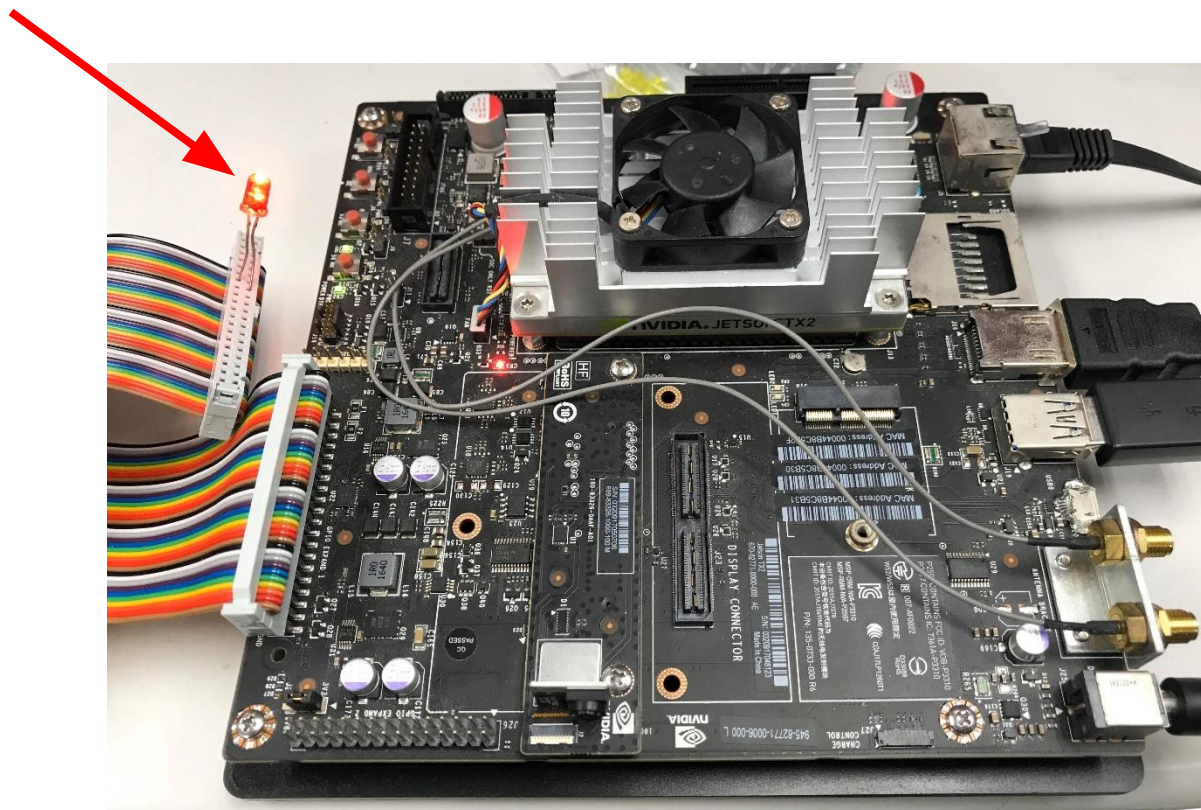
UBUNTU 操作 TX2 上的 GPIO

6. 將剛剛啟用的sysfs filename (範例為gpio255)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/gpio255# echo 1 > value
root@tegra-ubuntu:/sys/class/gpio/gpio255# cat value
1
```


UBUNTU 操作 TX2 上的 GPIO

- 若已完成上面的操作步驟，則sysfs filename(範例為gpio255)對應的Pin腳(範例為Pin 15)狀態為1，若有接LED可以看到該腳位的LED是亮的



UBUNTU 操作 TX2 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      gpiochip216  gpiochip240  gpiochip320
gpio255     gpiochip224  gpiochip256  unexport
root@tegra-ubuntu:/sys/class/gpio# echo 255 > unexport
root@tegra-ubuntu:/sys/class/gpio# ls
export      gpiochip224  gpiochip256  unexport
gpiochip216  gpiochip240  gpiochip320
```


以 C++透過 UBUNTU 操作 TX2 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/sys/class/gpio/GPIO必須透過開檔/關檔/寫檔/讀檔的方式存取底下的介面來操作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 TX2 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 TX2 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 TX2 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 TX2 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 TX2 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 TX2 上的 GPIO

- 範例主程式

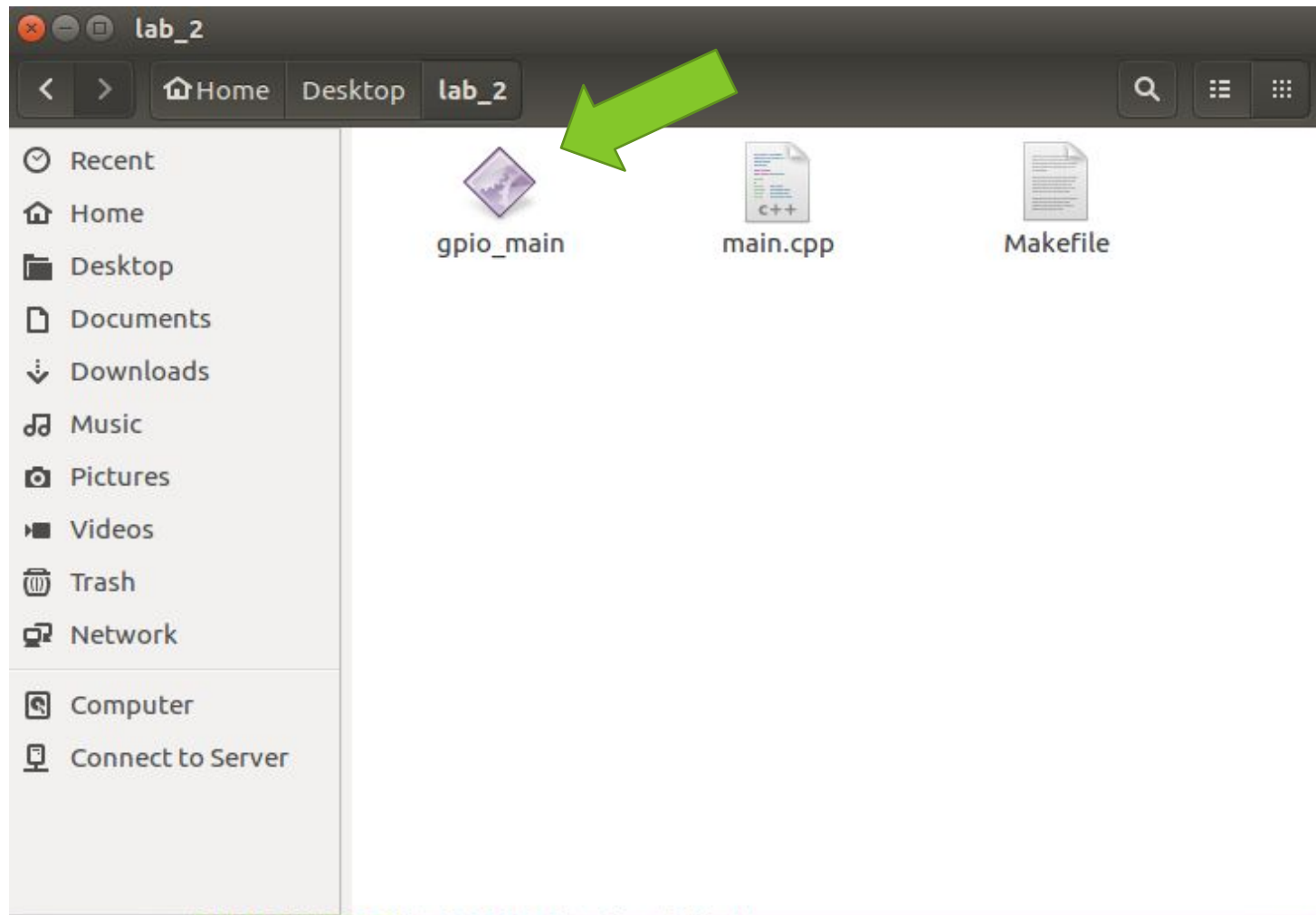
```
// 使用範例
int main(int argc, char *argv[])
{
    int input;
    std::cout << "輸入1為啟用gpio255並設定狀態為1\n輸入2為將gpio255設定狀態為0並停用gpio255\n>>>";
    std::cin >> input;
    if(input == 1)
    {
        // 啟用GPIO255
        gpio_export(255);
        // 將此gpio255設定為輸出用途out
        gpio_set_dir(255, "out");
        // 將此gpio255設定狀態為1
        gpio_set_value(255, 1);
    }
    else if(input == 2)
    {
        // 將此gpio255設定狀態為0
        gpio_set_value(255, 0);
        // 停用gpio255
        gpio_unexport(255);
    }
    return 0;
}
```


以 C++透過 UBUNTU 操作 TX2 上的 GPIO

- 跨平台編譯並傳到TX2上執行
 - 我們需要使用 **aarch64-linux-gnu-g++** 對此程式編譯成可以在TX2上執行的程式
 1. 使用 **aarch64-linux-gnu-g++-o** <輸出的執行檔名稱> <cpp檔案名稱>
 2. 可以用 **file** 查看編譯出來的執行檔是否為arm平台使用的
 - **file** <執行檔的名稱>

```
nvidia@ubuntu:~/Desktop$ cd lab_2/
nvidia@ubuntu:~/Desktop/lab_2$ aarch64-linux-gnu-g++ -o gpio_main main.cpp 1
nvidia@ubuntu:~/Desktop/lab_2$ ls
gpio_main main.cpp Makefile
nvidia@ubuntu:~/Desktop/lab_2$ file gpio_main 2
gpio_main: ELF 64-bit LSB executable, ARM aarch64, version 1 (SYSV), dynamically
linked, interpreter /lib/ld-, for GNU/Linux 3.7.0, BuildID[sha1]=f0e31de68abc47
4520e9ce80048d36bf8a048cd2, not stripped
```

以 C++ 透過 UBUNTU 操作 TX2 上的 GPIO



以 C++透過 UBUNTU 操作 TX2 上的 GPIO

- 跨平台編譯完成後可以透過前一章節所教的sftp將此執行檔(範例為:gpio_main)傳送到TX2嵌入式平台上執行
- 透過ssh進入TX2後移動到指定目錄找到剛剛傳上去的程式, 要執行該程式要先確認是否有執行權限(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

```
nvidia@tegra-ubuntu:~/lab_2$ ls -l
total 20
-rw-rw-r-- 1 nvidia nvidia 16472 Feb 26 13:17 gpio_main
nvidia@tegra-ubuntu:~/lab_2$ chmod +x gpio_main
nvidia@tegra-ubuntu:~/lab_2$ ls -l
total 20
-rwxrwxr-x 1 nvidia nvidia 16472 Feb 26 13:17 gpio_main
```

沒權限

增加權限

有權限

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

```
nvidia@tegra-ubuntu:~/lab_2$ ./gpio_main
輸入1為啟用gpio255並設定狀態為1
輸入2為將gpio255設定狀態為0並停用gpio255
>>>
```

以 C++透過 UBUNTU 操作 TX2 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 ./gpio_main**
 - 方法2(懶人版, 小心使用):使用**sudo su**將權限改為**root**即可直接執行

```
nvidia@tegra-ubuntu:~/lab_2$ sudo ./gpio_main
[sudo] password for nvidia:
輸入1為啟用gpio255並設定狀態為1
輸入2為將gpio255設定狀態為0並停用gpio255
>>>1
nvidia@tegra-ubuntu:~/lab_2$ ls /sys/class/gpio/
export  gpio389  gpiochip216  gpiochip240  gpiochip320
gpio255  gpio398  gpiochip224  gpiochip256  unexport
nvidia@tegra-ubuntu:~/lab_2$ cat /sys/class/gpio/gpio255/value
1
```

```
nvidia@tegra-ubuntu:~/lab_2$ sudo ./gpio_main
輸入1為啟用gpio255並設定狀態為1
輸入2為將gpio255設定狀態為0並停用gpio255
>>>2
nvidia@tegra-ubuntu:~/lab_2$ ls /sys/class/gpio/
export  gpio398  gpiochip224  gpiochip256  unexport
gpio389  gpiochip216  gpiochip240  gpiochip320
```

成功停用
gpio255

TX2 GPIO 相關參考資料

- `snprintf`
 - `snprintf()` 的功能類似 `printf()/fprintf()/sprintf()`，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-sprintf>
 - <http://www.cplusplus.com/reference/cstdio/sprintf/>
 - https://wirelessr.gitbooks.io/working-life/content/sprintf_miao_wu_qiong.html

TX2 GPIO 相關參考資料

- 以下是TX2 GPIO相關參考資料, 提供更進階的學習內容:
<https://www.jetsonhacks.com/nvidia-jetson-tx2-j21-header-pinout/>
- TX2開發手冊
<https://forums.developer.nvidia.com/uploads/short-url/QhIPn8rdiD1Cp4w7ueh28eQpHR.pdf>