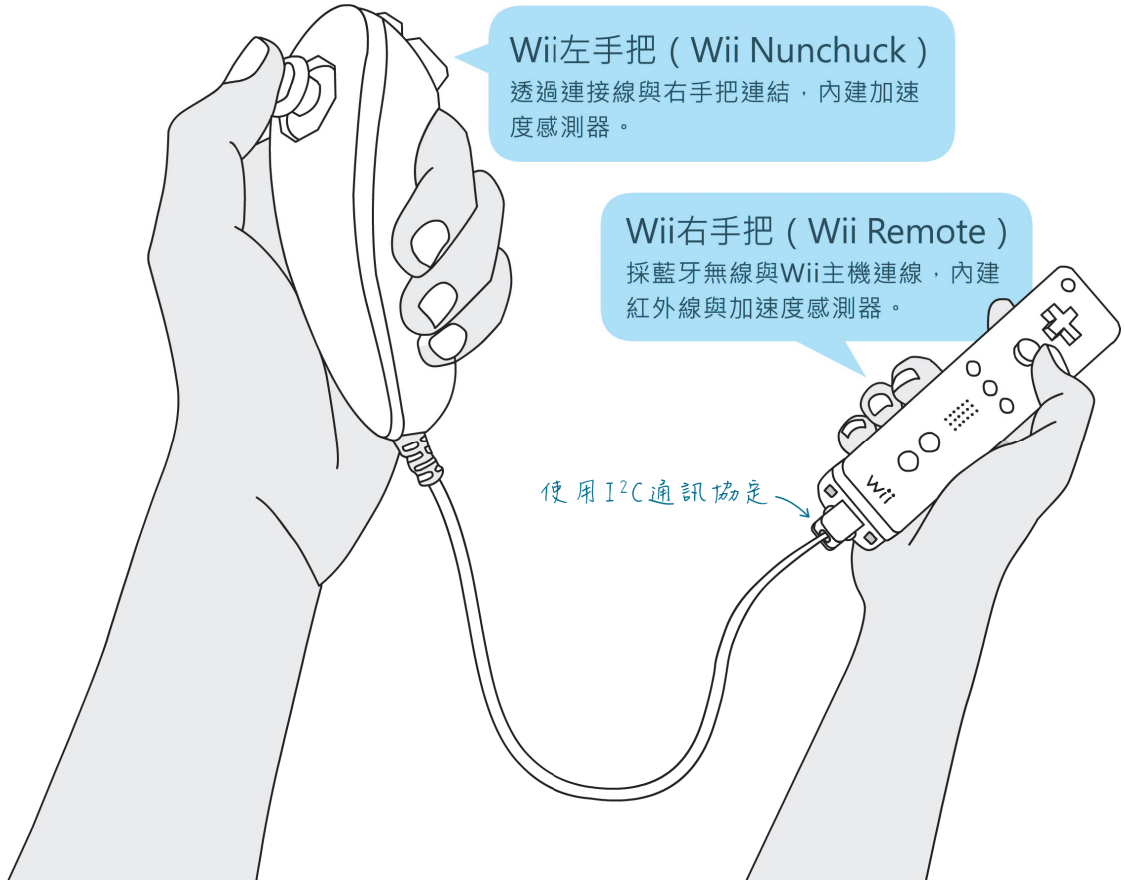
11001

# 使用 Wii 搖桿控制機械手臂

# F-1 認識 Wii 左手把的通訊介面：$I^2C$

任天堂的 Wii 遊戲機，除了帶來嶄新的體感玩法，從電子玩家的觀點來看，它還具備一項前所未見的特色：**Wii 是第一台廣泛採用電子業界標準的遊戲機。**在 Wii 之前的遊戲機，都具有各種專屬的介面，舉凡控制器的接頭、記憶卡的形式、遊戲卡匣或光碟的格式、連接顯示器的接線轉換裝置，甚至電源線的接頭，都是特殊規格。

Wii 遊戲機採用標準的 **SD 記憶卡**、無線控制器（右手把）和主機之間，採用標準的**藍牙介面**連線。右手把和左手把（原名為 Nunchuk）採用有線連結，它們之間的通訊協定（亦即，溝通方式）也是採用業界的 **$I^2C$ 標準**。



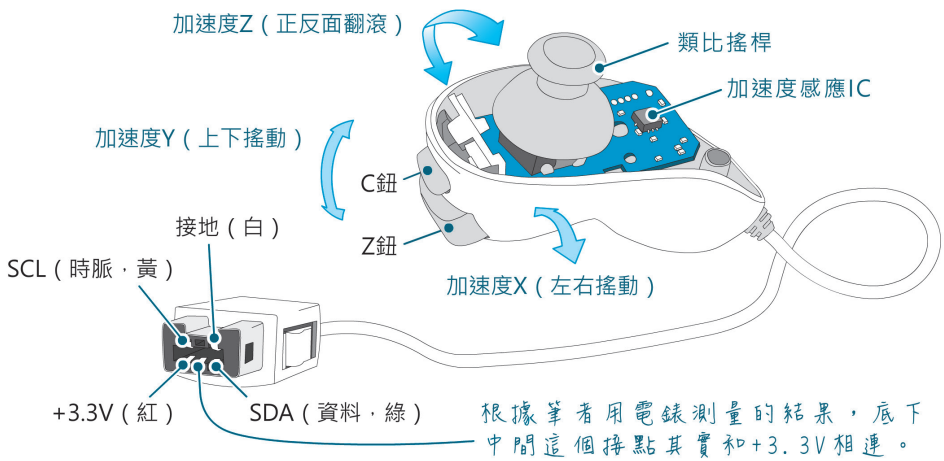
# 動手做 F-1 讀取 Wii 左手把的搖桿、按鈕與加速度計值

**實驗說明：**Wii 的控制器分成左、右兩個手把，都各自內建一個加速度感測器。本實驗將連結左手把與 Arduino，並讀取手把上的類比搖桿、按鈕與加速度狀態值。

**實驗材料：**

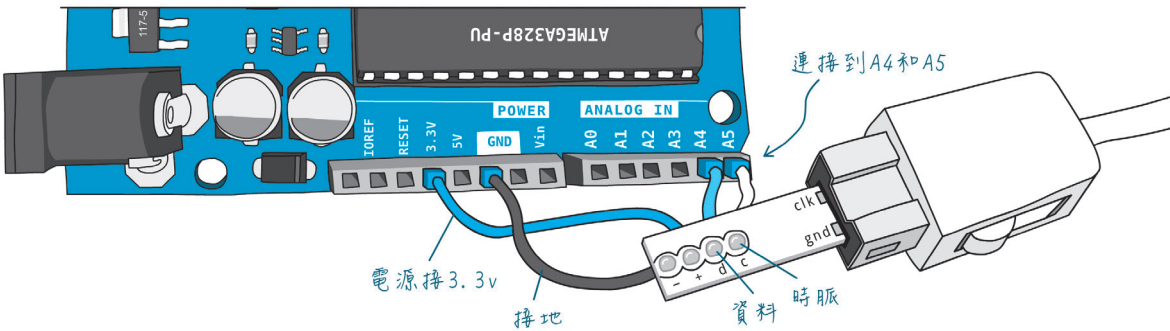
| | |
|---|---|
| Wii 左手把 | 一支 |
| Arduino 與左手把的轉接器 | 一塊 |

左手把具有一個**類比搖桿**、**兩個按鈕**以及**加速度感測器**，它和右手把之間的連線，採用標準的 $I^2C$ 規範，因此，左手把可以輕易地和 Arduino 整合在一起。

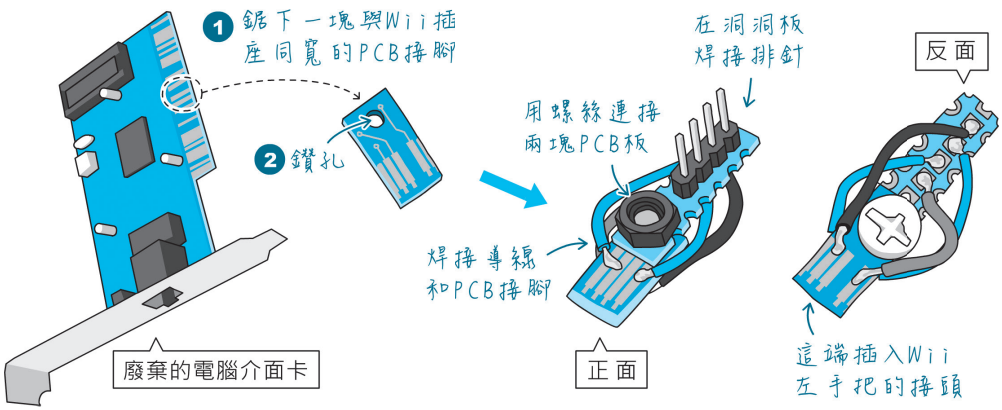



上圖紅、黃、白、綠指的是接頭內部的接線顏色，讀者可以忽略。

**實驗電路：**Wii 左手把的接頭是特殊規格，除了把它拆掉重新焊接之外，市面上有販售轉接 PCB 板，一端接 Wii 左手把，另一邊可插入 Arduino 板。左手把裡面已經內建上拉電阻，因此不需要再接其他電阻。



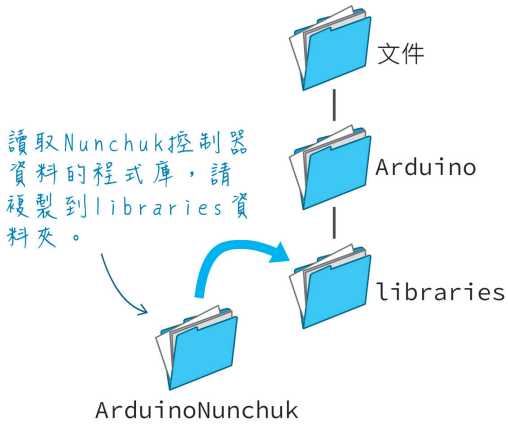

筆者的轉接板是從廢棄的桌上型電腦介面卡鋸下一小塊銅箔接點部分（裁切的寬度與 Wii 左手把的插孔相同），再焊接四條導線連接 PCB 板正反兩面，四個角落的接點：




多數人都直接把轉接板插入 Arduino 的 A2~A5 接腳，A2 連接到手把的接地，A3 連接到手把的正電源輸入，然後把 **A2 設定成低電位（LOW），A3 設定成高電位（HIGH）**，這樣就相當於從 Arduino 板子提供 5V 電壓給左手把。

然而，**Wii 左手把的電源是 3.3V**，雖然未聽說因為接 5V 而損毀，但為了保障它的壽命，建議讀者還是接 3.3V 比較好。

**實驗程式：**在 Arduino 上讀取 Wii 左手把的各項數值最簡單的方法，是採用 Gabriel Bianconi (http://www.gabrielbianconi.com/) 開發的 ArduinoNunchuk 程式庫。請先把該程式庫（收錄在範例檔中，也可以從原作者的網站下載）複製到 Arduino 的 **libraries** 根目錄：




重新開啟 Arduino 程式開發工具，選擇『**檔案/範例/ArduinoNunchuk/ArduinoNunchukDemo**』，即可開啟範例程式碼（註：範例程式的序列埠連線採用 19200bps 速率，筆者將它改成 9600）：

```
#include <Wire.h>
#include <ArduinoNunchuk.h>

ArduinoNunchuk nunchuk = ArduinoNunchuk();

void setup() {
  Serial.begin(9600);
  nunchuk.init();
}

void loop() {
  nunchuk.update();

  Serial.print(nunchuk.analogX, DEC);
  Serial.print(' ');
  Serial.print(nunchuk.analogY, DEC);
  Serial.print(' ');
  Serial.print(nunchuk.accelX, DEC);
  Serial.print(' ');
  Serial.print(nunchuk.accelY, DEC);
  Serial.print(' ');
  Serial.print(nunchuk.accelZ, DEC);
  Serial.print(' ');
  Serial.print(nunchuk.zButton, DEC);
  Serial.print(' ');
  Serial.println(nunchuk.cButton, DEC);
}
```

引用ArduinoNunchuk程式庫

宣告一個名叫"nunchuk"的ArduinoNunchuk物件

初始化Wii左手把

接收並更新手把資料

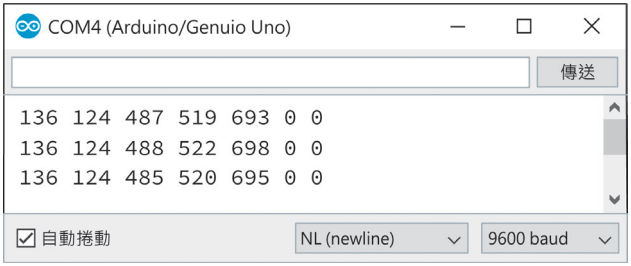
代表輸出成十進位，可省略不寫。

輸出類比搖桿X, Y值

輸出加速度X, Y, Z方向值

輸出Z鈕與C鈕值

**實驗結果：**接好 Wii 左手把，然後編譯並上傳程式碼，再開啟**序列埠監控視窗**，將能看見 Wii 左手把傳回的感測器數值。




如果讀者想要進一步了解和 Wii 左手把詳細溝通的過程，請參閱下文「解析 Wii 左手把的 $I^2C$ 通訊流程」一節。

## Wii 左手把的感測器數值範圍

從顯示在**序列埠監控視窗**的感測器數值可以發現，如果把類比搖桿推到最左邊，X 值不是 0，而是 30 左右；把它推到最右邊，X 傳回值也不是 1024。Wii 左手把的類比搖桿傳回值如下：

- **搖桿 X 軸值：**30（最左邊）～225（最右邊）
- **搖桿 Y 軸值：**29（最底部）～223（最上方）

同樣地，加速度感測器的有效值範圍是 0~1024。但**只有在劇烈晃動或者甩動控制器時，才會出現接近 0 或 1024 的數值**，平時操作時的加速度感測器的傳回值如下：

- **X 方向的加速度值：**約 300（向左邊傾斜）～740（向右邊傾斜）。
- **Y 方向的加速度值：**約 280（手把前頭朝上或往後傾斜）～720（朝下或往前傾斜）。
- **Z 方向的加速度值：**約 320（翻轉到背面）～760（翻轉到正面）。

# 動手做 F-2 使用 Wii 左手把控制機械手臂

**實驗說明：**延續「動手做 F-1」的 Wii 左手把接線，我們可以用 Wii 左手把上的類比搖桿，取代「動手做 13-2」的類比搖桿來控制機械手臂。

**實驗程式：**首先撰寫使用左手把搖桿控制機械手臂的程式碼：

```
#include <Wire.h>
#include <ArduinoNunchuk.h>
#include <Servo.h>     // 引用伺服馬達程式庫

ArduinoNunchuk nunchuk = ArduinoNunchuk();
Servo servoX, servoY; // 宣告伺服馬達程式物件


int posX, posY;        // 暫存伺服馬達角度的變數

void setup() {
  nunchuk.init();      // 初始化 Wii 左手把
  servoX.attach(8);    // 伺服碼馬達接在數位 8 和 9 腳
  servoY.attach(9);
}

void loop()
{
  nunchuk.update();    // 更新左手把資料

  // 將左手把搖桿輸入值，對應成伺服馬達的 0~179 度
  posX = map(nunchuk.analogX, 30, 225, 0, 179);
  posY = map(nunchuk.analogY, 29, 223, 0, 179);
  servoX.write(posX); // 設定伺服馬達的旋轉角度
  servoY.write(posY);
  delay(15);
}
```

不過既然接上 Wii 左手把，沒用到加速度功能實在可惜。本單元將把上一節的操作模式改成：平時用搖桿操作，**按著左手把的 Z 鈕不放時，改由加速度偵測器控制伺服馬達。**

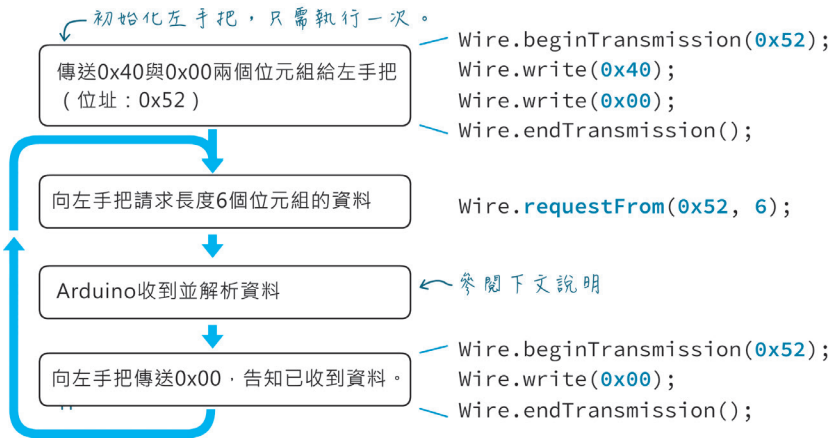
請將上一節的 loop() 區塊，修改成：

```
void loop() {
  nunchuk.update();

  if (nunchuk.zButton) {    // 若 Z 鈕的值為 1...

    // 用加速度 X, Y 軸值設定伺服馬達的旋轉角度值
    posX = map(nunchuk.accelX, 300, 740, 0, 179);
    posY = map(nunchuk.accelY, 280, 720, 0, 179);
  } else {
    posX = map(nunchuk.analogX, 30, 225, 0, 179);
    posY = map(nunchuk.analogY, 29, 223, 0, 179);
  }

  servoX.write(posX); // 設定伺服馬達的旋轉角度
  servoY.write(posY);
  delay(15);
}
```

**實驗結果：**編譯並上傳程式碼即可使用左手把控制機械手臂。

## 解析 Wii 左手把的 I²C 通訊流程

若不使用現成的程式庫，想從頭自己採用 Wire.h 程式庫和 Wii 左手把的 I²C 協定奮戰，其實也不難。連結 Wii 左手把並讀取資料的流程如下：




向左手把索取資料時，它每次都會傳回 6 個位元組，裡面包含類比搖桿 X, Y 軸與加速度感測器等數值：

每次都傳回6個位元組資料

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | 類比搖桿X軸值 | | | | | | | |
| 1 | 類比搖桿Y軸值 | | | | | | | |
| 2 | 加速度X軸值（第2到9位元） | | | | | | | |
| 3 | 加速度Y軸值（第2到9位元） | | | | | | | |
| 4 | 加速度Z軸值（第2到9位元） | | | | | | | |
| 5 | 加速Z軸位元1 | 加速Z軸位元0 | 加速Y軸位元1 | 加速Y軸位元0 | 加速X軸位元1 | 加速X軸位元0 | C鈕 | Z鈕 |

跟第2位元組結合成10位元長的資料

加速度感測器的值長度為 10 位元，一個位元組放不下，前兩個位元放在編號 5 的位元組裡面，像上圖藍底的加速度 X 軸值一樣。此外，每個位元組資料都事先經過編碼，需要經過底下的運算式解碼，才能取得真正的數值：

實際資料值 = (讀取值 XOR 0x97) + 0x97

或者：

實際資料值 = (讀取值 XOR 0x17) + 0x17


XOR 代表**邏輯互斥**（或稱為**異或**）運算：若**兩個輸入值不同**，運算結果為**真（true）**。

其實，有另一種不需要解碼的方式，主要差別在於初始化 Wii 左手把的步驟。但不需解碼的程式設計並不會比較簡單，因此本文採用需要解碼的步驟做說明。上文採用的 ArduinoNunchuk 程式庫，其內部程式就是用不需解碼的步驟，有興趣的讀者可自行研究它的原始碼。

接收左手把傳入的 6 組資料，並且解碼的程式如下。先宣告一個儲存 6 個位元組資料的陣列，命名成 "buff"，並在收到資料時，逐一將它們解碼並存入 buff 陣列：

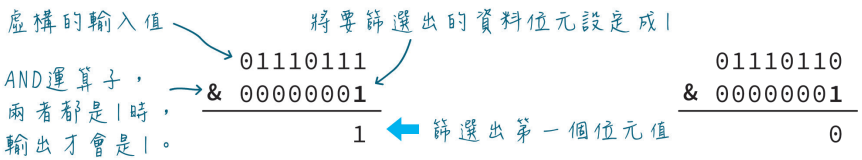
```
byte buff[6];
byte i = 0;
while(Wire.available()) {
  if(i < 6) {
    buff[i] = (Wire.read() ^ 0x17) + 0x17;
  }
  i++;
}
```

若覺得上面 8 行程式碼太囉嗦，可以用 for 迴圈改寫，功能一樣：

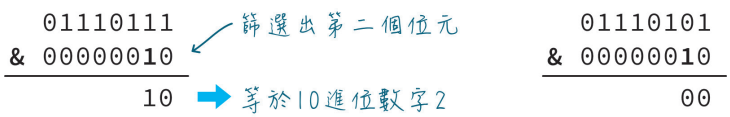
```
byte buff[6];
for (byte i = 0; Wire.available() && (i < 6); i++) {
  buff[i] = (Wire.read() ^ 0x17) + 0x17;
}
```

## 解析 Wii 左手把的資料值

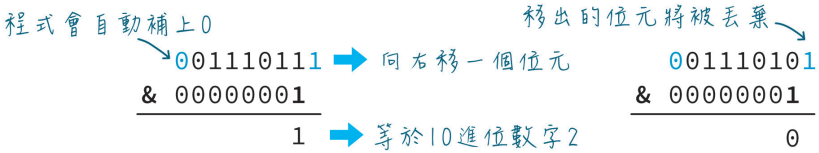
傳回的資料中，編號第 5 的位元組包含了不同意義的資料，假設我們要篩選出其中的位元 0 和 1（亦即，左手把的 Z 鈕和 C 鈕值），可以透過 **AND 運算子**達成：




同樣地，底下的 AND 運算將能篩選出位元 1 的值：




然而，篩選出的結果將是 10 進位的 "2" 或 0，如果要得到 "1" 或 0 的結果，請**先將輸入值往右移動一個位元**，再篩選數值：




左手把傳回的 Z 鈕和 C 鈕值，**0 代表「按下」，1 代表「放開」**。可是，按下是 1，放開是 0，比較符合程式的邏輯，如果要這麼做的話，將輸入值「取相反值」即可，實際的程式碼如下：

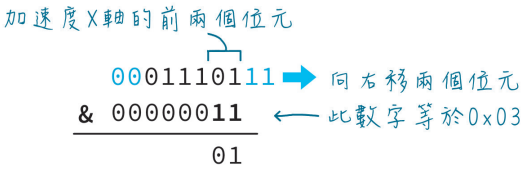
```
byte btnZ = ~buff[5] & 0x01;
byte btnC = (~buff[5] >> 1) & 0x01;
```

取相反值

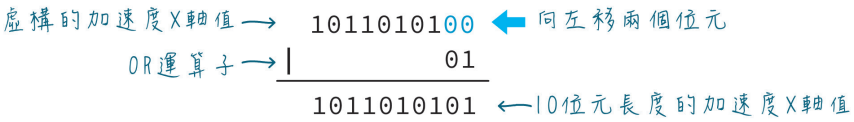
右移一個位元

如此，當 Z 鈕被按下時，btnZ 的值將是 1。

最後，我們需要重組加速度感測器 X, Y, Z 軸的 10 位元值，以 X 軸值為例，先篩選出編號第 5 的位元組裡的兩個位元：




接著把存放加速度 X 軸的 2~9 位元值，向左移兩個位元，再執行 **OR（或）運算**（亦即，只要任一邊為 1，運算結果就是 1）結合兩段數字，即可得到 10 位元長度的數值了：




實際的程式寫法：

```
int accX = buff[2] << 2 | ((buff[5] >> 2) & 0x03);
```

左移加速度X軸的2~9位元值　篩選出加速度X軸的前兩個位元

以此類推，合併加速度 Y 和 Z 軸資料的程式如下：

```
int accY = buff[3] << 2 | ((buff[5] >> 4) & 0x03);
int accZ = buff[4] << 2 | ((buff[5] >> 6) & 0x03);
```