# MediaTek Filogic 130A 課程

作者：盧怡仁

(1) IO mux 實驗：

一般而言，MCU 的每根 IO 都具備多功能，除了當 GPIO (General Purpose I/O) 之外，還能透過 API 函數來設定腳位的功能，稱為 IO multiplexing。先從 IO 控制 LED 燈號閃爍和偵測按鈕是否按下，這算是開始使用 Filogic 的熱身實驗。

(2) 感測器實驗：

Sensors (照度感測器 **BH1750**、溫溼度感測器 **SHT30**)和一個 **LCM 1602** 顯示模組，串在 I2C 介面上, 讀取這兩個感測器的數值，並顯示到 LCM 螢幕上。另外，進一步學習以 PID (Proportional, Integral, Derivative)控制之概念，寫一個溫溼度和光照的邏輯程式，以控制 GPIO 繼電器的開與關。

(3) 數據收集實驗：

先建立 WiFi 連線，取得板子的 IP 位址。再寫一個連線程式到 NTP 伺服器，取得網路時間，記錄感測器的數據和時間戳。此外，將感測器的數值透過 HTTP Get/Post 上傳到 Thingspeak 網站 https://thingspeak.com/，再從雲端查看數據 Dashboard。

(4) 音訊介面實驗：

學習 SPI 通訊介面，在 ST7735 顯示模組上繪圖。此外，SPI 的另一擴充介面為 SD，先在 SD 卡裡存放幾首 .wav 的檔案，由 Filogic 130A 處理器撥放 .wav，板子上的按鈕做為 play / stop。同理，利用錄音方式，將聲音存成 .wav 格式的音檔。

(5)藍芽 BLE 實驗：製作無線藍芽遙控器

在手機端安裝一個 APP (BLE Scanner)，連線 Filogic 開發板的藍芽功能，然後從 APP 發送指令到板子上。

準備一個 RTL8722 模組(國產的 Ameba 開發板)，燒錄 BLE_client 範例程式，同樣連線到 Filogic 開發板的藍芽，並傳送指令控制 Filogic 板。

# 開發環境之準備工作

在學習聯發科 Filogic 130A 開發板的課程之前，我們要先在電腦上安裝開發的軟體套件，像是編譯程式、燒錄、上傳韌體…等的工具。聯發科將這塊開發板的 SDK 已經整合到 Arduino IDE 環境上，讓開發者更為方便地使用。此外，如果有興趣深入研究 SDK 的人，Filogic 130A 也可以在 Linux 的環境下透過命令列執行 makefile 來達成上述的動作，這就是最原始且最直接的開發方式，參考[1]文章。

## Arduino IDE

Arduino 開發板發行多年了，也是 DIY maker 很喜歡拿來開發的板子之一，主要是板子的周邊介面讓入門者簡易明白，加上 IDE (Integrated Develop Environment) 這套工具介面簡潔且非常容易上手，這些優點讓 DIY maker 能夠很快驗證某些電路或功能和應用。另外，在教育方面，由於免費的開發工具以及初學者能負擔的板子價格，所以在理工科系也會拿 Arduino 板子做為課程，設計各類嵌入式系統的應用。不過，在這兒並不打算介紹 Arduino 開發板如何使用，而是介紹 Arduino IDE 這套工具如何編譯「非」Arduino 板子，借用這套 IDE 來開發第三方的板子。

隨著 Arduino 這個生態系統發展多年後，使用者越來越多，在 DIY maker 的討論越來越火熱的情況下，吸引來自「非」Arduino 生態的廠商也希望用 IDE 工具來編譯自家的板子。既然 IDE 是編譯工具，能直接編譯其他「非」Arduino 廠商的晶片嗎？答案當然是不行了，因為每家晶片的核心不見得相同 (Arduino 是 Atmel AVR 核心)，不能用同一套 compiler tool-chain 工具，有興趣者可以參考[2]一文。因此，在 IDE 裡面提供一個欄位，讓使用者輸入網址，並自動下載第三方開發板的開發工具 (tool-chain)，該工具會自動整合到 IDE 裡面，使用者也能夠非常便利地在 Arduino IDE 開發第三方的程式[3]。下圖一所示為支援聯發科 Filogic 130 開發板的網址，以及瑞昱阿米巴 Ameba 開發板的網址。

Filogic 130 在 Arduino IDE 安裝的步驟，需要多幾項操作：先自行到大聯大公司的大大通網站下載 SDK，另外在本機上安裝 tomcat 網路服務器，有關詳細的操作過程，請參考[4]影片。

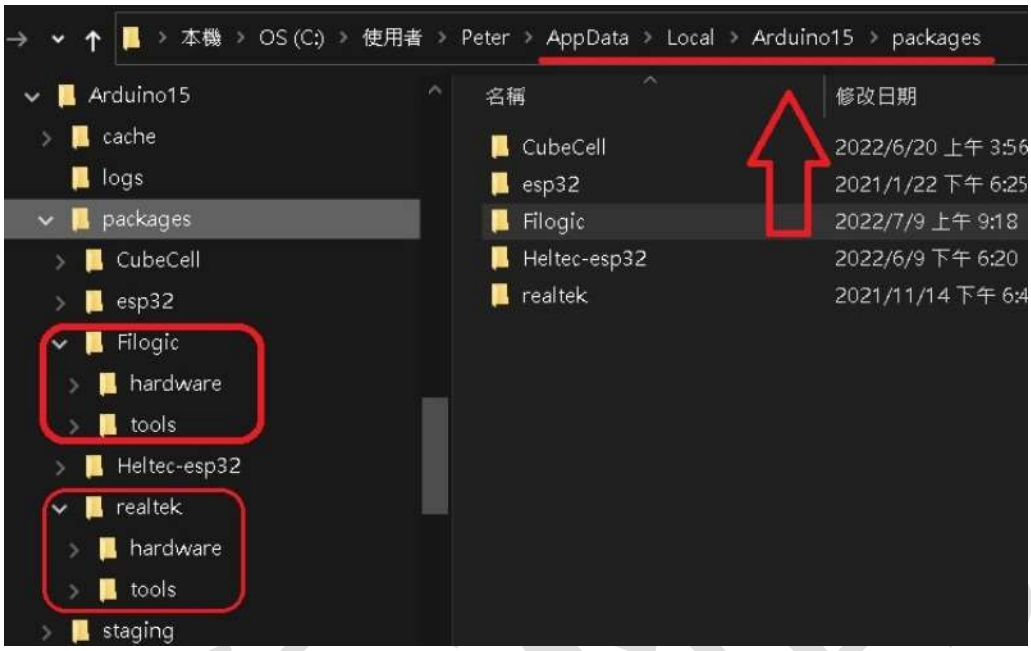

圖一：設定第三方開發板的下載網址

IDE 自動下載並安裝後，我們在下圖二的目錄中能看到第三方廠商的工具包，明顯觀察到工具包都包含了 hardware 和 tools 資料夾。其中，tools 裡面就是編譯的工具程式 (包括 compile 和 link...)，還有燒錄韌體上傳到板子的程式。以 EPS32 為例，這晶片的核心是 Xtensa 處理器，原廠用開源的 gcc 製作了一套屬於這顆處理器的編譯工具，在目錄底下可以找到 "xtensa-esp32-elf-c++.exe" .... 等編譯或 link 相關的執行檔案，這些執行檔能在 Windows 環境下編譯我們的 source code 成為 ESP32 處理器能運行的韌體。當中燒錄工具為 esptool.py，工具包裡也有提供。

同樣在 Filogic 和 realtek 目錄下，也能找到編譯相關的工具程式，聯發科和瑞昱的阿米巴系列開發板上面的核心晶片採用 **ARM Cortex**，所以聯發科公司採用開源的 gcc 製作了能編譯 ARM 晶片的工具，我們也能在目錄下找到如 "arm-none-eabi-c++.exe"....等在 Windows 環境下的執行檔，工具包裡面也提供了燒錄上傳韌體的工具。上述的這些執行檔案都已經被整合到 Arduino IDE 裡了，因此 IDE 根據使用者設定的板子來載入並執行適當的編譯工具，經過 compile、link 後，並燒錄上傳 firmware image 到開發板子上。

在工具包的 hardware 目錄，主要是開發板的驅動程式和廠商提供的範例程

式，我們在 IDE 畫面上看到的範例都是來自這個目錄裡面。有些廠商也會把 SDK source code 放到這個目錄底下，所以我們想要找尋 *.ino 函數或宣告的源頭，倒是可以從圖二的這目錄開始找起。




圖二：第三方開發板的編譯工具目錄

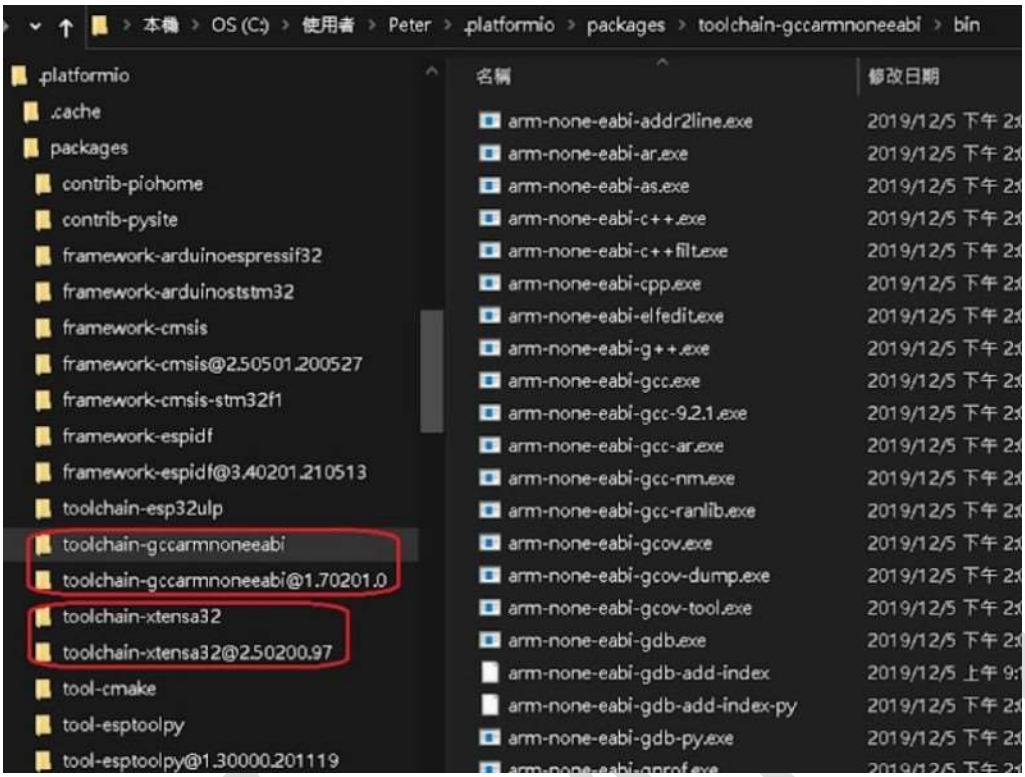
如果對於其他 IDE 還有興趣的話，可以安裝 Visual Studio Code (VS Code)，在這套 IDE 環境中，安裝 PlatformIO 插件，這個插件支援很多板子，如：Arduino、STM 系列板子、ESP32...等各家公司的開發板，因此會下載 3rd party 廠商的編譯和燒錄工具，像圖三所示為 ARM 的工具和 ESP32 的工具包。編譯工具能在 Windows 環境下執行，如果是 Linux 作業系統，插件就會下載 Linux 環境的編譯工具，這能滿足各類使用者。總歸一句話，IDE 的選擇還是以自己熟悉與操作方便為主。

每次在 Arduino IDE 的程式中，總會記得最重要的兩個起始函數：setup( ) 和 loop( )，為何每次開啟程式要先呼叫這兩個呢？首先，我們打開第三方廠商的 hardware 目錄下，應該能搜尋到 main.cpp 檔案，檢查這個路徑看看吧。

\Filogic\hardware\filogic_rtos\1.0.0\cores\arduino
\esp32\hardware\esp32\1.0.6\cores\esp32
\realtek\hardware\AmebaD\3.1.0\cores\arduino

看完後，清楚明白它是一個 FreeRTOS 作業系統，然後創建一個 main task，先呼叫 setup 函數當作初始化，然後進入一個迴圈後，再呼叫 loop 函數。挖掘這

些底層的程式碼後，我們就能更清楚整個系統運行的架構了。



圖三：PlatformIO 插件的編譯工具

底下整理三個具有無線網路和藍芽的晶片規格，不過硬體規格會隨著系列產品而升級，最終的詳細規格還是以公司官網為主。

| | **Filogic 130A** | **RTL8722** | **ESP32** |
|---|---|---|---|
| CPU | M33 + HiFi4 DSP | M33 + M23 | Xtensa 雙核心 |
| 最高內頻 | 300 MHz | 300 MHz | 240 MHz |
| Flash | 16 MB | 4 MB | 2-4 MB |
| SRAM | 1 MB | 512 KB | 512 KB |
| WiFi | 2.4G / 5G | 2.4G / 5G | 2.4G |
| BT | BLE 5.0 | BLE 5.0 | BLE 4.2 (新系列可支援 5.0) |

## Filogic 130A 方塊圖

為了方便往後實驗線路的解說，我們對於 Filogic 130A 開發板的周邊介面必

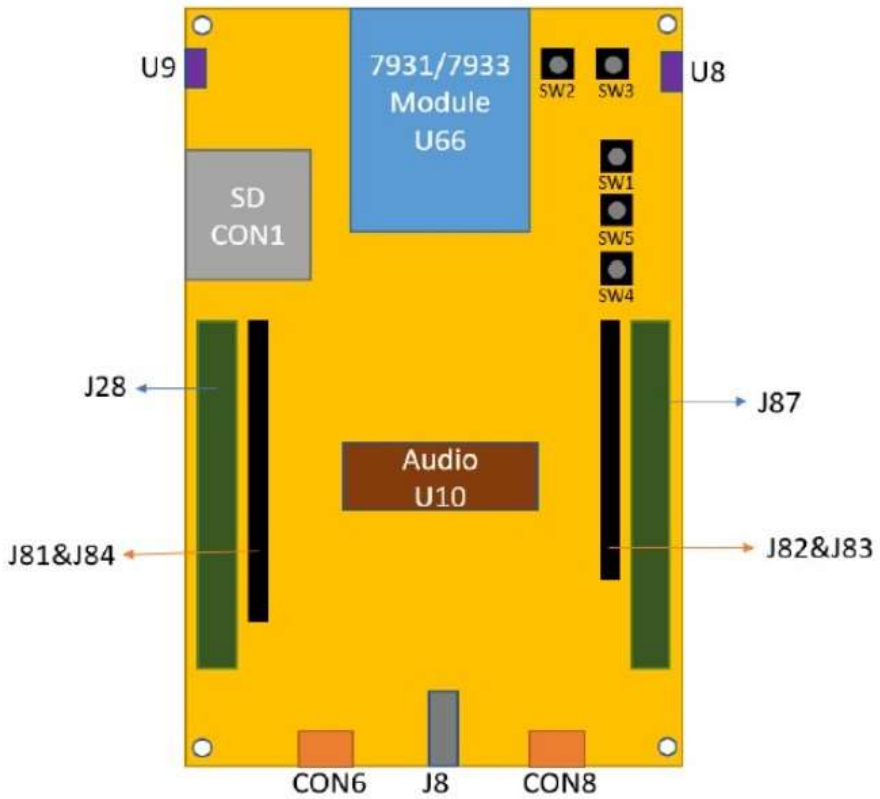
須先有一個圖像，底下圖四為整塊開發板的方塊圖，由聯發科的 MT7931/7933 晶片為系統核心(編號 U66)，周邊帶著幾顆測試的按鈕以及 SD 卡的插槽介面，中間有一顆音訊放大晶片(編號 U10)。

開發板的電源從 **CON8** 輸入，來自 USB 所提供的+5V 電源。另外，**CON8** 接到電腦後，也做為 Arduino IDE 的序列埠使用。CON6 是 MT7931/7933 晶片模組的 USB 介面，目前暫時不開放。

## ➢ 注意燒錄程序

試著上傳韌體到開發板上，首先我們開啟一個範例程式，在 Arduino IDE 上面編譯並上傳。當編譯時，先按住板子上的 SW1 (Reset)鈕，然後觀察 IDE 的下方訊息欄，直到我們看到“INFO: Goto open COMxx”這段訊息時，再放開板子上的 SW1 鈕，IDE 工具便會進入燒錄韌體的階段，開始上傳。最後，上傳成功後，我們會看到“Finished!”的訊息。

每次上傳後，我們必須再按一次板子上的 SW1 (Reset)鈕，這樣板子才會重新執行我們的程式碼。它並不像 Arduino 開發板會自動重新啟動。




圖四：Filogic 130A 系統方塊圖

Filogic 開發板是由群登科技所設計製造，聯發科的 MT7933 晶片被封裝在一個金屬外殼裡面(圖四的正上方)，再將 IO 腳位接到外部的板子上。我們可以到群登的網站下載硬體規格與電路圖，網上提供有關硬體方面的資源。

接下來實驗的 SDK 版本，主要基於 Filogic IoT Arduino SDK 版本 1.0.0 (2022/07/04)所開發的範例。如果範例程式遇到問題的話，請到大大通網站下載 SDK 更新。

# IO mux 實驗

## 實驗目的：

市面常見 MCU 上的 IO 接腳功能是可以當作多重用途的，一根 IO 腳可能是 GPIO，也可能是 analog input，也可以是某通訊介面的腳位。在 MT7933 晶片上的每根腳位也都具有多重功能，因此使用 IO 前必須對每一根腳位先設定其功能，我們習慣稱之為 IO mux (IO multiplexing)。在 Filogic Arduino SDK 的初始化過程中，系統已經給每根腳位配置了預設功能，本實驗希望能了解 Filogic 130A 開發板上每根腳位所設定的功能為何，這對之後的實驗也有幫助。

寫過 Arduino 程式的人都知道，用 **pinMode** 標準函數設定 IO 的功能，不過這兒不打算介紹上層的標準函數，而是談談 SDK 裡面的驅動程式。在 SDK 安裝包裡面，除了有 Arduino 的標準函數之外，仔細搜尋先前提到的目錄還可以發現 MT7933 晶片的驅動函數，像是 GPIO 相關的函數，被封裝為 HAL 層。底下的程式碼會列印出所有 GPIO 腳位的功能，以及 IO 的方向，這樣我們便能知道 IO 腳位的對應，再搭配電路圖一一對應，整個系統的架構就會清楚了。

```
26  // show all GPIO mapping
27  for(int i = 0; i < HAL_GPIO_MAX; i++)
28  {
29    hal_gpio_get_function((hal_gpio_pin_t)i, &gpioMode);
30    hal_gpio_get_direction((hal_gpio_pin_t)i, &gpioDir);
31    // show all GPIO mode and direction
32    sprintf(buf, "GPIO%d Mode=%d DIR=%d\n", i, gpioMode, gpioDir);
33    Serial.print(buf);
34  }
```

圖一：列印所有 GPIO 腳位功能

Arduino SDK 系統啟動後，系統程式會將每根 GPIO 接腳依據功能進行初始化。因此，我們將系統啟動後的 GPIO 腳位功能列印出來後，再根據 hal_pinmux_define.h 檔案裡面的模式定義，把腳位的模式整理成表單，然後把電路圖上的標示和檔案的定義做對應，如下圖二所示。舉例來說，當我們在 Arduino 的 setup 函數裡面呼叫 **pinMode**(6, OUTPUT)，則對應到電路圖的 C_SDIO_CLK 腳被設定為 GPO 的輸出。換句話說，當 GPIO6 被設定 GPO 之後，那 SDIO 的功能就不能使用了。如果我們沒有呼叫 **pinMode** 函數做腳位設定的話，所有的腳位模式就是系統所給定的預設值。

依據 pinmux 的預設值，我們就能了解到 Arduino IDE 上面的 Serial 埠對應

到 UART1，也就是 GPIO_T_1 和 GPIO_T_3 腳位。開發板上的 SW2 和 SW3 按鈕對應到電路圖的 GPIO_T_6 和 GPIO_T_8 腳位，分別當作 GPI_47 和 GPI_49 輸入使用。


圖二：GPIO 腳位模式與電路圖之對應

將這個實驗用到的程式碼都放在下面的連結，請自行下載和修改 https://drive.google.com/file/d/1Sb5Q3PYCyfJh8EBPSuYjT2bgEQSpY2v1/view?usp=sharing (基於 SDK 1.0.0 開發)

**ADC 接腳**

Filogic 開發板也提供 ADC 的腳位，當呼叫 analogRead( )函數的時候，系統會初始化 ADC 功能，板子上的 **GPIO_B_15** (HAL_GPIO_27)和 **GPIO_B_16** (HAL_GPIO_27)就設定為 ADC 接腳。不過測試後發現，可偵測的電壓範圍似乎只有 0~1.8V，這好像有點問題？！(等新版本的更新修正)

# 感測器實驗

## 實驗目的：

Filogic 130A 開發板就是以 ARM Cortex 為核心的 MCU，具有各類常用的通訊介面，本實驗希望實現 I2C 通訊介面的連接與操作。在各類控制器晶片中，除了 UART 通訊介面之外，I2C 介面也是控制器必備的周邊之一，它的通訊速度高於 UART，而且在 I2C bus 上能掛載多個模組並進行通訊。

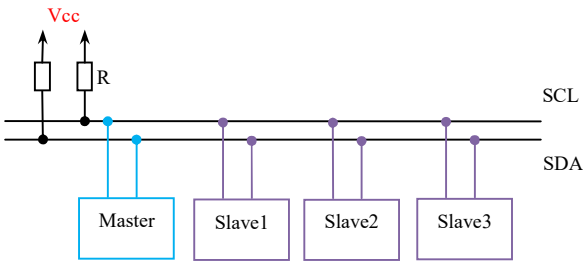
## 實驗目標：

在 Filogic 130A 開發板的 I2C 介面上，我們試著掛載三個模組到這 I2C bus，這兒三個模組分別是照度感測器 **BH1750**、溫溼度感測器 **SHT30/31** 以及一個 **LCM 1602** 顯示模組。並且將 BH1750 和 SHT30 讀取的數值，顯示到 LCM1602 上面，我們便能看到目前光線的照度以及環境溫溼度的變動情況。

最終的目標，希望學習以 PID (Proportional, Integral, Derivative)控制之概念，在 Filogic 裡面寫一套溫溼度和光照的計算與邏輯程式，以控制 GPIO 繼電器的開與關，參考[5]。

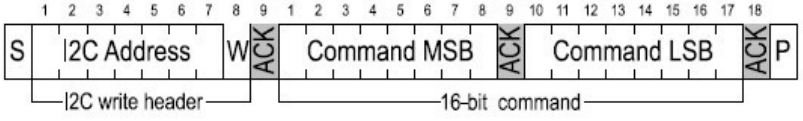
## I2C 簡介：

從電路的角度來看，I2C 通訊介面是由兩條訊號線所組成，分別是 SCL（clock）和 SDA（data），要注意的一點是這兩條線路必須接 pull-high 的電路，這是 I2C 在實體層能否通訊的關鍵。這介面的通訊速度大約介於 100 kbps～400 kbps，可分為低速模式與高速模式，比起 UART 介面的速度還要快。另外，I2C 介面上能串接多個模組，如下圖一所示，每個 slave 模組或晶片都會對應一個屬於自己通訊的位址，當 master 想與某個 slave 模組通訊時，便會先發送其位址，再後帶著資料串給 slave。



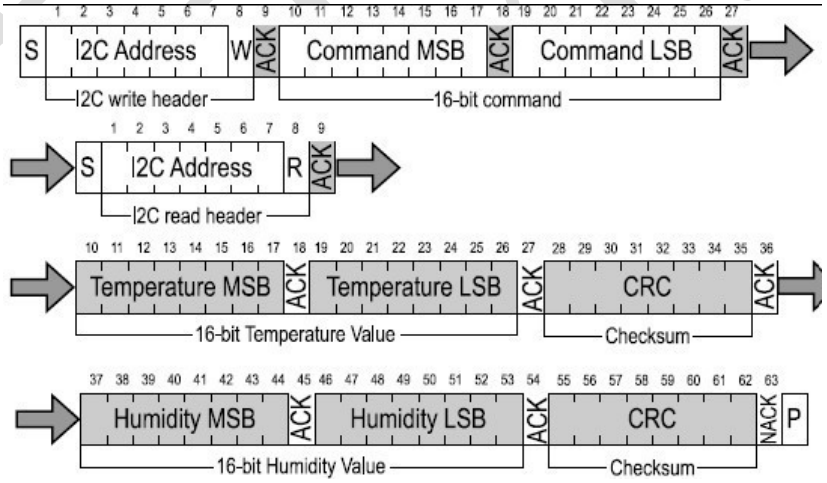

圖一：I2C 實體電路的配置圖

從 I2C 的時序圖來觀察通訊格式會比較容易了解，圖二是 master (也就是 Filogic 130A)發送一個寫入的命令給 bus 上的某一模組。首先，要發一個 Start condition (圖上的 S)，然後緊接著模組的位址，最後帶著一個 Write bit (圖上的 W)，此時等待模組回應。對應到位址的模組可回應 ACK 或 NACK，圖中灰色的 ACK 是由模組回應，所以收到 ACK 的 master 就是依序發送命令給模組。如果收到 NACK，代表該模組拒絕後續的命令，也可以說返回錯誤給 master。最終結束通訊時，master 發一個 Stop condition (圖中的 P)。




圖二：master 發送一個寫入的通訊格式 (取自 SHT30 datasheet)

下圖三則是 master 發送一個讀取數據的命令給 bus 上的某一模組，同樣地，先發一個 Start condition (圖上的 S)，然後緊接著模組的位址，再帶著命令列傳給模組。當該模組對於這命令回應 ACK 之後，master 發送一個讀取的封包，如圖上的 S + Address + R，等待模組的 ACK。接著，模組便會劈哩啪啦地送出一串數據 (圖中灰色的部分)，而這次的 ACK (圖中呈白色)就是由 master 回應給模組，最終再發一個 Stop condition 結束通訊。

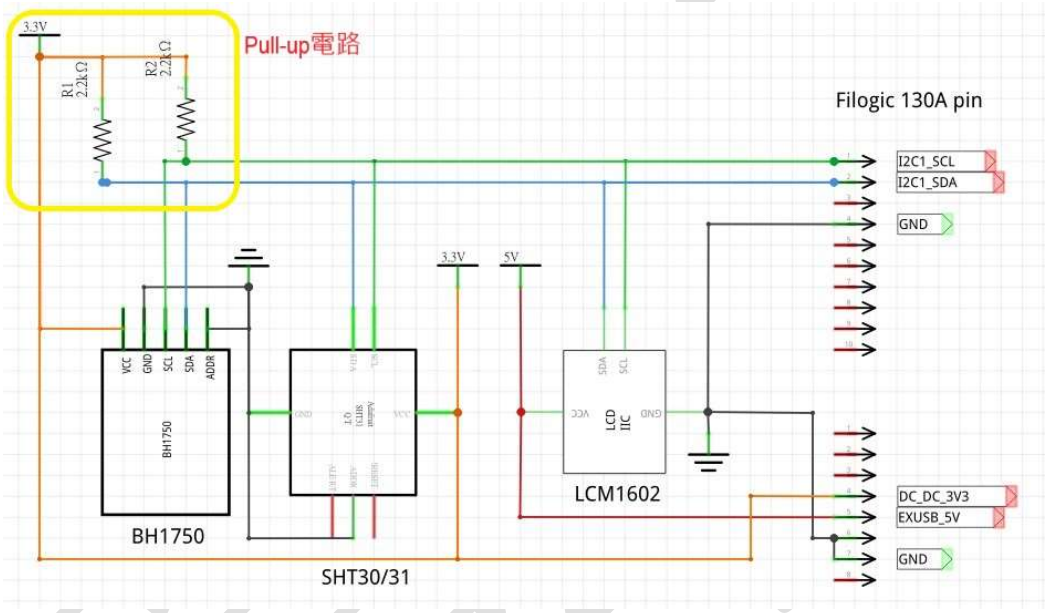



圖三：master 發送一個讀取的通訊格式 (取自 SHT30 datasheet)

了解上述的 I2C 通訊之後，對於我們在 Filogic 上面設計一個 I2C 介面的驅動程式將會有所幫助。

## 實驗電路接法：

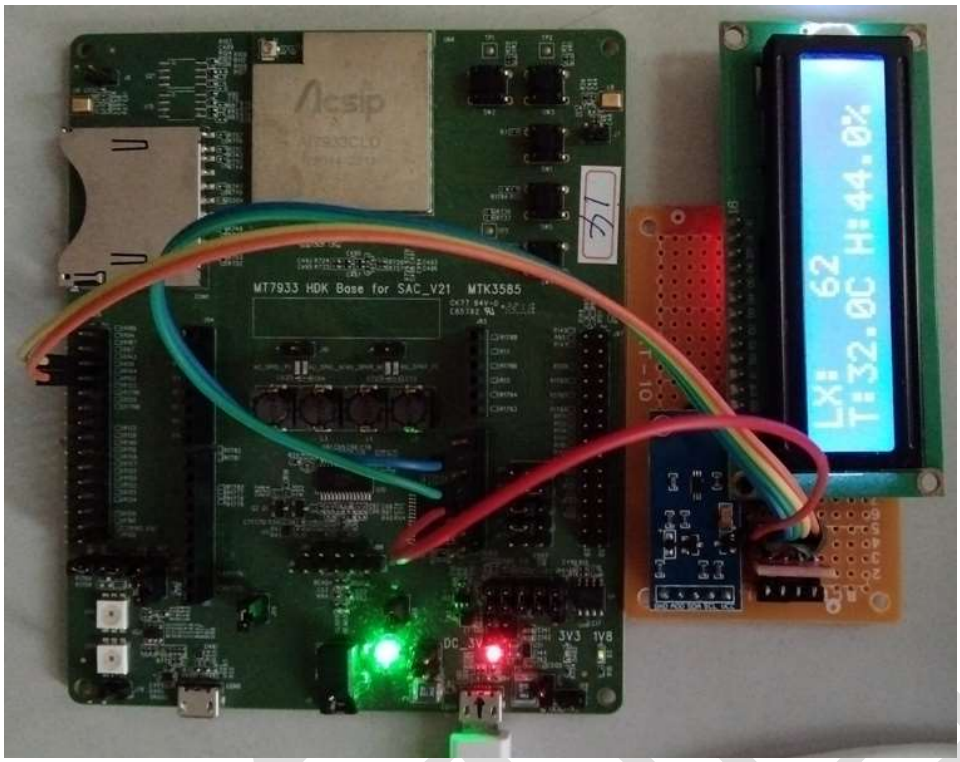
在實驗中，我們必須將 Filogic 開發板上的 I2C 介面與三個模組(SHT30/31、BH1750和LCM1602)串接起來，同時需要事先準備一塊麵包板和一些杜邦線材。這幾個模組之間電路的參考接法，如下圖四所示，左上方為 pull-up 電路，而在 Filogic 130A 開發板上有預留 I2C 線路的 pull-up，需要看一下板子上的 pull-up 電阻有沒有焊上去(或量測有沒有 pull-up 電壓)，如果有，就不用在外部再增加 pull-up 電路了。




圖四：電路參考圖

上圖的 **I2C1**_SCL 接腳在板子的 **J28** 排針第七 pin (GPIO_T_5)，**I2C1**_SDA 接腳在板子的 **J28** 第五 pin (GPIO_T_4)。電源的部分來自 **J82** 的第二與第四 pin 為+3.3V，第五 pin 為+5V，第六與第七 pin 為 GND。

整體配線接好後，如下圖五所示，右邊是 LCM 模組、溫溼度模組和光照模組，透過 I2C 介面接線到左手邊的 J28 排針上，電源接到 J82 上。在 LCM 螢幕上，顯示目前的光度數值，越大代表越亮，第二行顯示目前的溫溼度數值。




圖五：開發板與周邊模組的接線圖

首先，程式碼要初始化 I2C 通訊介面，呼叫 **hal_i2c_mater_init** 函數，通訊的時脈為 400 kbps，參考如下：

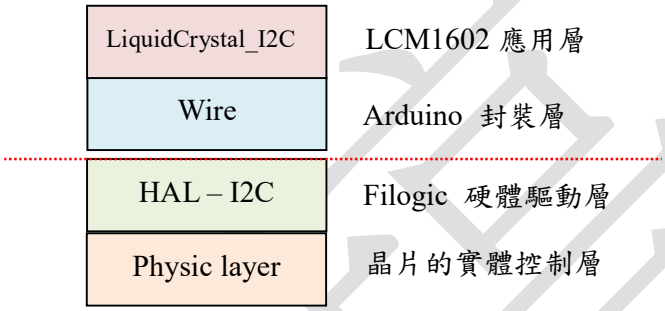
```
40  hal_i2c_master_deinit(HAL_I2C_MASTER_1);
41  i2c_cfg.frequency = HAL_I2C_FREQUENCY_400K;
42  if( HAL_I2C_STATUS_OK != hal_i2c_master_init(HAL_I2C_MASTER_1, &i2c_cfg) )
43  {
44    Serial.print("hal_i2c_master_init err!!!\n");
45  }
```

接著透過 I2C 介面初始化每個模組，而每個模組都有一個位址，LCM 位址為 0x27，溫溼度位址為 0x44，光照位址為 0x23，程式碼參考如下。然後，我們根據模組的 datasheet 寫它的驅動程式。

```
47  Lcm_Init(0x27);
48  Sht_Reset(0x44);
49  delay(500);
50
51  Sht_Init(0x44);
52  Bh1750_Init(0x23);
53  delay(500);
```

最後，將這個實驗用到的程式碼都放在下面的連結，請自行下載和修改。
https://drive.google.com/file/d/19-7W1KpNbkh9sWL9JLxO31IXE5CjrcO4/view?usp=sharing (基於 SDK 1.0.0 開發)

在 Arduino SDK **1.1.0** 版本中，提供了 I2C 頂層的 API 呼叫方式，如範例程式“I2C LCD Module 1602”，其中 **hal_i2c_mater_init** 函數被封裝在 Wire 類別之中，請參考目錄 \packages\Filogic\hardware\filogic_rtos\1.1.0\libraries\Wire 下的程式碼，軟體的封裝架構如下所示。




在 SDK 1.0.0 版本的開發中，我們直接呼叫 HAL 的驅動函數，然後自己寫 LCM1602 的控制程式。不過在 **1.1.0** 版本中，多了 Wire 的 Arduino 封裝層，所以我們可以用 Wire 裡面的函數控制 I2C 通訊，如：Wire.beginTransmission( )、Wire.endTransmission( )、Wire.write( )….等 Arduino 的標準接口來開發應用層的控制程式。

SDK **1.1.0** 版本所控制的 I2C 實體線路是 **I2C0**，而前面 1.0.0 版本提供的範例是實體線路 **I2C1**，所以程式碼有些不同。從開發板的接線方式來說，請參考電路圖，**I2C0**_SCL 接腳在板子的 **J28** 排針第三 pin (GPIO_T_2)，**I2C0**_SDA 接腳在板子的 **J28** 第一 pin (GPIO_T_0)。把 LCM1602 模組接好後，載入範例程式“I2C LCD Module 1602”就能看到螢幕的動態變化了。

# 數據收集實驗

## 實驗目的：

Filogic 130A 開發板上面的 MT7933 晶片模組整合了無線網路/藍芽的功能，要如何操作 WiFi 呢？本實驗希望利用其 WiFi 介面從無線 AP 取得本板的 IP 位址，建立起 TCP/UDP 連線後，並執行應用面的通訊協定以完成物聯網的概念。

## 實驗目標：

學習在 Filogic 130A 開發板上的無線網路連線，接著寫一個 UDP 連線程式從 NTP（Network Time Protocol）服務器上取得日期與時間，並將這日期時間填入 MT7933 晶片的 RTC 裡。另外，再寫一個連線程式將周邊 I/O 的數據上傳到雲端，如：ThingSpeak 網站，來達到數據收集的目的以及呈現數據的可視化，請參考[6]。

開啟範例程式中的 **ConnectWithWPA**，只要將程式碼的 ssid 和 pass 分別填入無線 AP 的 SSID 名稱以及登入的密碼後，這個範例程式便能從 AP 獲得 IP 位址，並在 console 上面列印出網路相關的訊息。

在下圖一顯示所提供的 SDK 目錄下，LWiFi 裡面有無線網路相關的範例與函數庫，其中函數的類別(Class)有 WiFi、WiFiClient、WiFiServer、WiFiUdp…等，對於 TCP 連線會調用 WiFiClient 或 WiFiServer，而 UDP 連線會調用 WiFiUdp 裡面的函數。在開發程式的過程，我們會到這目錄下參考 SDK 如何使用。


圖一：Filogic 所提供的 SDK 範例程式碼

從 WiFi AP 取得 IP 位址後，要寫一個連線到 NTP 服務器的程式，以便獲得網路的日期和時間。首先，我們要設定 DNS，才能獲得 NTP 服務器的 IP 位址，

參考下列紅色標示的程式碼：WiFi.**setDNS** 和 WiFi.**hostByName**

```
53  // start the RTC module
54  LRTC.begin();
55
56  WiFi.setDNS(dns);
57  if( WiFi.hostByName(hostname, ntpAddr) )
58  {
59    Serial.print("NTP IP: "); Serial.println(ntpAddr);
60    do {
61      sntp_init(ntpAddr);
62      delay(1000);
63      if( ++tryCnt >= 5 ) break;
64    } while ( sntp_recvfrom() == 0 );
65
66    ut = time(NULL) + sntp_timestamp + (8*60*60);
67    tmInfo = localtime(&ut);
68    LRTC.set(tmInfo->tm_year+1900, tmInfo->tm_mon+1, tmInfo->tm_mday,
69         tmInfo->tm_hour, tmInfo->tm_min, tmInfo->tm_sec);
70  }
```

接著，設計一個 UDP 連線程式到 NTP 服務器，我們需要用到 WiFiUdp 這個函數類別，以及從 GitHub 上面下載 sntp.c 的原始碼來參考看看怎麼寫，然後將 WiFiUdp 的函數套用進去。底下是 WiFiUdp 有關傳送封包的程式碼，先要創建一個 UDP 連線，呼叫 **beginPacket** (也就是創建 socket…等工作)，接著將要發送的資料填入緩存的 buffer 裡面，這兒呼叫 **write** 函數。最後，通知 WiFiUdp 將緩存的資料發送出去，必須呼叫 **endPacket** 函數。

```
200  if( !ntpClient.beginPacket(ntpIp, SNTP_PORT) ) Serial.println("beginPacket error!!!!");
201
202  sntp_initialize_request(sntpmsg);
203  txLen = ntpClient.write((const char*)sntpmsg, sizeof(struct sntp_msg));
204  Serial.print("Sending request:"); Serial.println(txLen);
205
206  if( !ntpClient.endPacket() ) Serial.println("endPacket error!!!!");
```

底下是 WiFiUdp 有關接收封包的程式碼，先要呼叫 **parsePacket** 函數，將收到的封包存放到緩存的 **buffer** 裡面，再呼叫 available 函數，確認有資料在 buffer 後，我們最後呼叫 **read** 函數將封包讀出來，放到我們的記憶體空間。以上就是 WiFiUdp 類別的使用方式，詳細的原始碼可以到圖一的目錄裡面去查找。

我們了解這個 UDP 連線介面的操作之後，就可以將 sntp.c 的部分程式碼用上述的函數替換掉，沒有意外的話，我們能收到 NTP 回傳的秒數(很長的一串數字)。然後，再呼叫系統函數 **localtime**，將這秒數轉換成日期與時間，並且寫到

Filogic 的 RTC 裡面。到此為止，Filogic 130A 開發板也就具有實際的時間了。

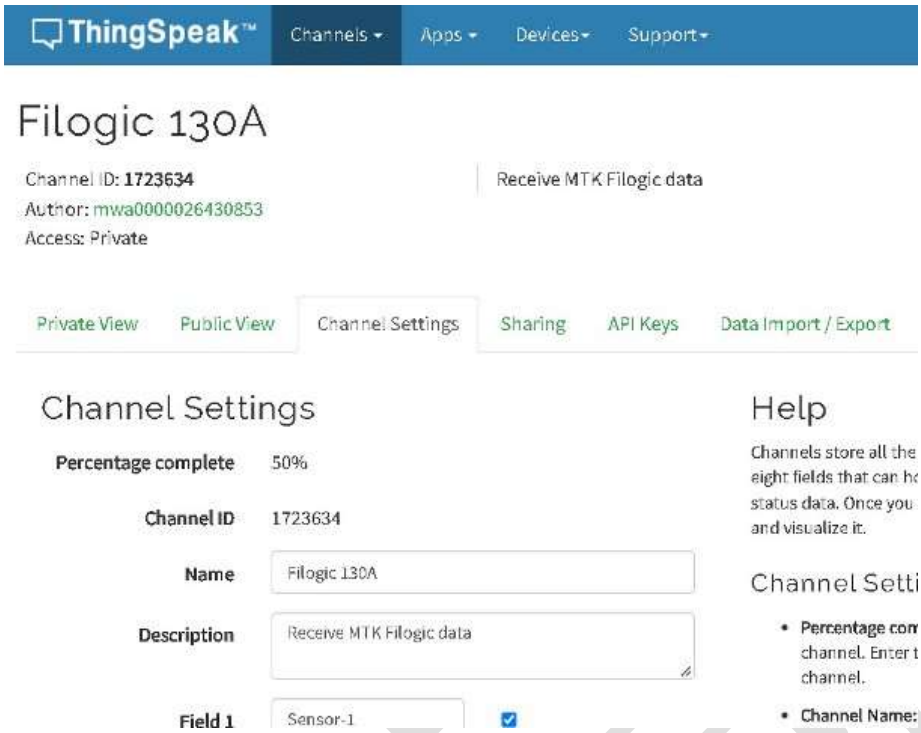
```
213   if( !ntpClient.parsePacket() ) Serial.println("parsePacket error!!!!");
214   else {
215     while( ntpClient.available() )
216     {
217       rxLen = ntpClient.read(sntpRxBuf, SNTP_MSG_LEN*2);
218       if( rxLen > 0 ) {
219         sntp_recv(sntpRxBuf, rxLen);
220       }
221     }
222   }
223   return rxLen;
```

實驗到這階段，先將網路連線程式碼都放在下面的連結，請自行下載和修改。
https://drive.google.com/file/d/13vf_eamUb53sGu02BP536MPm43rldfce/view?usp=sharing (基於 SDK 1.0.0 開發)

最後再提一點，在 Arduino SDK **1.1.0** 版本之後，提供了範例程式 **WiFiUdpNtpClient**，這也是一個取得網路時間的參考案例，大家可以執行並觀察其結果。

## ThingSpeak 簡介

為了在雲端伺服器上面呈現數據的可視化，我們將採用 ThingSpeak 網站所提供的服務，先註冊一個帳號，再建立一個自己的 channel，如下圖所示。接著，從帳號中能獲得 “Write API Key” 和 “Read API Key” ，記住這兩個 key 之後在開發程式碼的過程會使用到。




圖二：建立 ThingSpeak 帳號與通道

接著，研究看看數據用哪種方式上傳到該網站上，從網站裡的 Tutorial → "Write Data to Channel" 內文提到採用兩種方式，一種是 REST API，另一種是 MQTT API，而本實驗將會使用 REST API 的方式把 Filogic 130A 的數據上傳到這個雲端。請參考[9]。

在大聯大網站有另一個 Linux 版本的 SDK 1.1.3，裡面有提供 httpClient 的原始碼，裡面的函數可以實現 REST API 的功能，所以我們從這 SDK 拿到 httpClient 原始碼，並移植到 Arduino SDK 的專案裡面。首先，建立 http 連線，呼叫函數 httpclient_connect，如下：

```
sprintf(get_url, "https://api.thingspeak.com/update");
printf("httpclient_connect:%d", httpclient_connect(&iotClient, get_url));
```

接著，上傳感測器的數值到 ThingSpeak，需要調用到 httpclient_post 函數。我們把剛剛前面提到的 API Key 填入 http 的內容裡面，同時把要上傳的數值也放在內容裡，如下：

```
sprintf(iotPostBuf, "api_key=9Y9MTEF4LA44UTOJ&field1=%d&field2=%d", ++tempVal1 % 100, ++tempVal2 % 50);
iotData.post_buf = iotPostBuf;
iotData.post_buf_len = strlen(iotPostBuf);

sprintf((char*)buf, "application/x-www-form-urlencoded");
iotData.post_content_type = (char*)buf;

printf("connecting to ThingSpeak for update...\n");
if( httpclient_post(&iotClient, get_url, &iotData) == HTTPCLIENT_OK)
{
  printf("data_len=%d %s\n", iotData.response_content_len - iotData.retrieve_len, iotData.response_buf);
}
```

上傳的key和數值

http 內容

最後，從 ThingSpeak 網站上能看到 Filogic 130A 開發板上傳的資料，呈現出可視化的結果，如下圖三所示。這個實驗所使用到的範例原始碼，http 連線程式碼放在下面的連結，請自行下載和修改。

https://drive.google.com/file/d/1XeVhz5_2ipkKP9uTCCLbUfaeVFG7EhLD/view?usp=sharing


圖三：範例程式上傳的可視化結果

# 音訊介面實驗

## 實驗目的：

學習在 Filogic 130A 開發板上 SPI 通訊介面的連接與操作，這介面普遍使用在各類控制器晶片中，它具有各種變形的通訊方式，可以與高速晶片連接並互傳資料，可以做為 SD 卡的存取介面，也可以當成數位音訊資料的傳輸介面。

## 實驗目標：

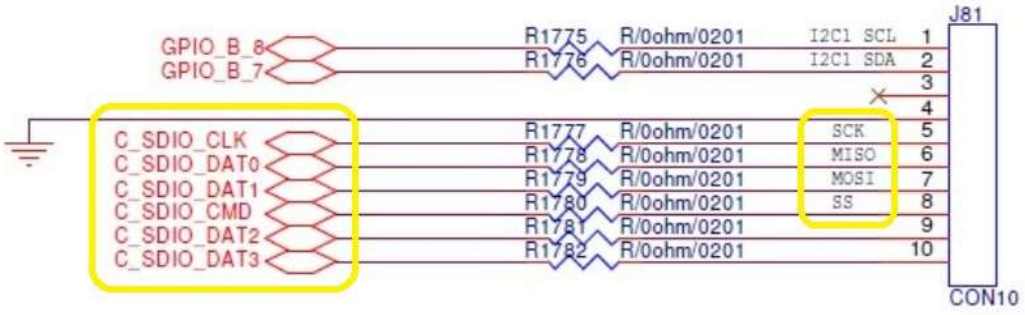
首先，SD 卡格式化後，將前一實驗的每筆數據寫到 Filogic 130A 開發板的 SD 卡裡面，完成資料讀寫的目標，因此這開發板就具有資料儲存的能力了。

再來，實現 Filogic 130A 開發板的音樂撥放功能，我們在 SD 卡裡面放入幾首（*.wav）的檔案，再呼叫 API 由 MT7933 晶片模組的處理器撥放（*.wav），將板子上的按鈕做為 play / stop 鈕，請參考[7]。

## SPI 簡介：

與前面的 I2C 介面相比，SPI 通訊介面的速路更快，傳輸是 mega 等級的速度，如：10M～25Mbps，速度高低與 MCU 本身的時脈有關，此外資料傳輸可以採用全雙工的方式，這些特性都比 I2C 介面更為優異。就電路的角度來看，SPI 介面由四條訊號線所組成，分別是 SCLK( clock )、MOSI、MISO 和 SS( chip select )，其中 MOSI 線和 MISO 線分別代表資料流向 slave 和資料流向 master。

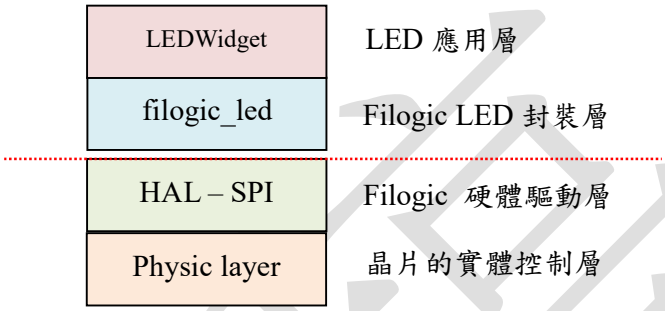
MT7933 晶片模組除了具備 SPI 介面之外，還能支援 SDIO 介面，這算是 SPI 介面的延伸。從電路來看，SDIO 額外多了兩組數據線，因此訊號線的組成為：SCLK、CMD、DATA0～3 共六條線，如下圖一所示。




圖一：SDIO 介面的訊號線

開發板上面有兩顆透過 SPI 控制的 LED 燈號，其中利用 SPI－**SCK** 和 SPI－**MOSI** 的接腳連接到 LED 上。另外，如果打算將 SPI 接腳連去控制其他晶片或模組的話，我們可以移除 **J79** 和 **J80** 的連接，便是斷開 LED 線路。

在 Arduino SDK **1.1.0** 版本中，提供了 SPI 頂層的 API 呼叫方式，如範例程式“RGB_LED”，其中 **hal_spi_mater_init** 函數被封裝在 filogic_led 函數集之中，請參考目錄 \packages\Filogic\hardware\filogic_rtos\1.1.0\libraries\RGB_LED 下的程式碼，軟體的封裝架構如下所示。

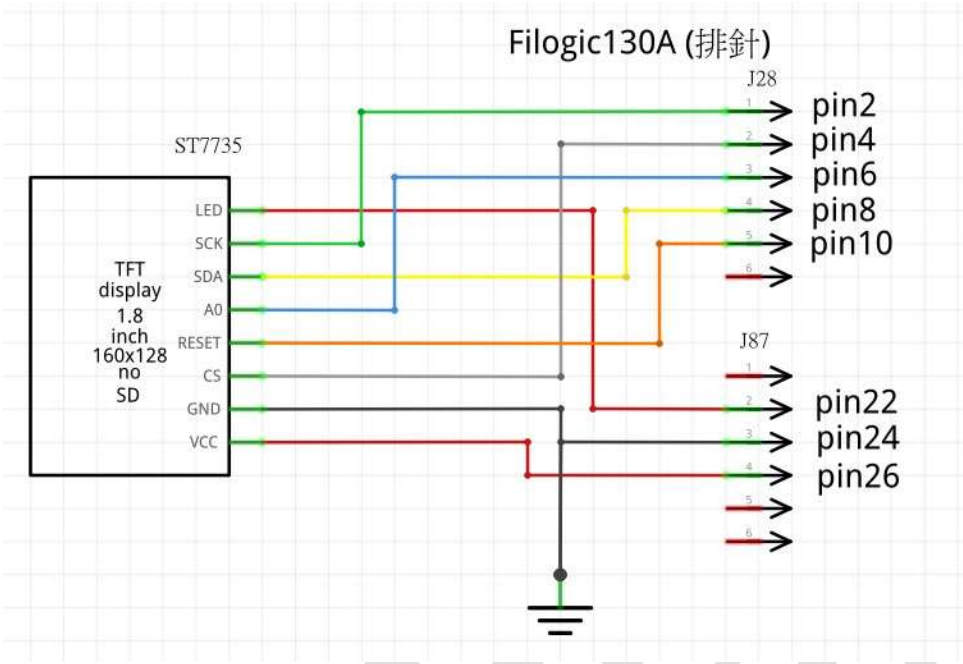



**TFT LCD 模組**

顯示模組 ST7735 也是透過 SPI 介面控制，模組除了 SPI 的基本線路（MISO 不需要）之外，還有兩條 Reset 和 DC 控制線，Reset 線顧名思義是重置模組的訊號，DC 線是切換 Data 或 Command 的訊號線，再加上電源與 GND 線，由上述這幾條控制線所組成。

ST7735 與 Filogic130A 開發板上的的排針對接，如下圖二所示。ST7735 模組需要+3.3V 電源，還有背光源，由開發板的 **J87** 排針提供電壓與 GND。而控制線路 SPI、DC、Reset 則需要接到開發板的 **J28** 排針上，這樣硬體線路便大致完成了。

接下來說明，有關如何驅動 ST7735 模組的方式，在 Arduino 軟體架構下，有些玩家已經將 ST7735 驅動程式以及上層的應用 API 封裝成一個通用的架構，這對 Filogic 開發板的控制而言非常便利。我們在 Arduino IDE 的程式庫裡面搜尋 Adafruit，然後安裝這幾個程式庫：Adafruit_BusIO、Adafruit_GFX_Library、Adafruit_ST7735_and_ST7789_Library。他提供了一些範例程式，我們需要修改調整一下 code，便能在 Filogic 上面顯示了。




圖二：ST7735 與開發板接線圖

**SD 卡/音源控制**

截至 SDK 1.1.0 版本前還未支援 SD 卡和語音方面的控制，等待後續版本更新。

# 藍芽 BLE 實驗

## 實驗目的：

Filogic 130A 開發板上提供了藍芽 BLE 功能，如何使用呢？本實驗希望製作一個藍芽遙控器，以無線方式能控制並操作 Filogic 開發板上的 I/O 輸出。試想這開發板可能是智慧音箱的核心、可以是物聯網裝置、也可以是移動裝置…等，透過一個藍芽 BLE 遙控器操控或讀取這個 Filogic 開發板的訊息。

## 實驗目標：

Filogic 130A 提供 BLE 的 GATT 範例程式碼，分為 Peripheral 端(或 server 端)和 Central 端(或 client 端)，因為 Filogic 開發板做為提供 I/O 點位或板子上各種服務的對象，它的角色是 Peripheral 端，所以我們要載入 BLE Peripheral 的範例程式碼。至於 BLE 遙控器(Central 端)，它可以是手機的 APP、可以是電腦的 BLE 程式、或者是另一塊開發板的 BLE，實驗最終目標是完成 BLE 控制 Filogic 板，如[8]。

BLE 裝置(Peripheral 端)可以透過 GATT 協定向 Central 端發送數據，或接收來自 Central 端的數據。而 GATT 協定使用服務、特徵和屬性來組成數據，並控制如何讀取和寫入數據。BLE 的藍牙 SIG 規範包括針對常見應用的幾種定義服務，不過使用者可以依據合適的數據結構與應用面的需求來實現自定義的服務和特性，如下圖一服務和特性的 UUID 碼。

| Allocation Type | Allocated UUID | Allocated for |
|---|---|---|
| Service Classes and Profiles | 0x1401 | HDP Source |
| Service Classes and Profiles | 0x1402 | HDP Sink |
| GATT Service | 0x1800 | Generic Access |
| GATT Service | 0x1801 | Generic Attribute |
| GATT Service | 0x1802 | Immediate Alert |
| GATT Service | 0x1803 | Link Loss |
| GATT Service | 0x1804 | Tx Power |
| GATT Service | 0x1805 | Current Time |
| GATT Service | 0x1806 | Reference Time Update |
| GATT Service | 0x1807 | Next DST Change |
| GATT Service | 0x1808 | Glucose |
| GATT Service | 0x1809 | Health Thermometer |
| GATT Service | 0x180A | Device Information |
| GATT Service | 0x180D | Heart Rate |
| GATT Service | 0x180E | Phone Alert Status |
| GATT Service | 0x180F | Battery |

| | | |
|---|---|---|
| GATT Declarations | 0x2800 | Primary Service |
| GATT Declarations | 0x2801 | Secondary Service |
| GATT Declarations | 0x2802 | Include |
| GATT Declarations | 0x2803 | Characteristic |
| GATT Descriptor | 0x2900 | Characteristic Extended Properties |
| GATT Descriptor | 0x2901 | Characteristic User Description |
| GATT Descriptor | 0x2902 | Client Characteristic Configuration |
| GATT Descriptor | 0x2903 | Server Characteristic Configuration |

| | | |
|---|---|---|
| GATT Characteristic and Object Type | 0x2A00 | Device Name |
| GATT Characteristic and Object Type | 0x2A01 | Appearance |
| GATT Characteristic and Object Type | 0x2A02 | Peripheral Privacy Flag |
| GATT Characteristic and Object Type | 0x2A03 | Reconnection Address |
| GATT Characteristic and Object Type | 0x2A04 | Peripheral Preferred Connection Parameters |
| GATT Characteristic and Object Type | 0x2A05 | Service Changed |
| GATT Characteristic and Object Type | 0x2A06 | Alert Level |
| GATT Characteristic and Object Type | 0x2A07 | Tx Power Level |
| GATT Characteristic and Object Type | 0x2A08 | Date Time |
| GATT Characteristic and Object Type | 0x2A09 | Day of Week |
| GATT Characteristic and Object Type | 0x2A0A | Day Date Time |

圖一：GATT 定義的 UUID (取自 bluetooth.com)

開啟範例程式中的**SimplePeripheral**，我們可以修改GATT服務的UUID碼，下圖的 110C 就是 UUID 碼，並宣告一個 GATT 的服務以及一個 GATT 的特性。接下來，需要修改程式碼裡面裝置的名稱，這是藍芽裝置用來廣播(advertisement)的設備名，為了實作方便，把名稱改為“MTK_BLE Lab 1”，避免藍芽裝置名稱重複，應該編號來測試。

SimplePeripheral §

```
11 */
12 #include <LBLE.h>
13 #include <LBLEPeripheral.h>
14
15 // Define a simple GATT service with only 1 characteristic
16 LBLEService ledService("110C0010-E8F2-537E-4F6C-D104768A1214");
17 LBLECharacteristicString switchCharacteristic("110C0011-E8F2-537E-4F6C-D104768A1214", LBLE_READ | LBLE_WRITE);
18
```

下面所示的這函數，用來呼叫從藍芽 BLE 接收到的字串，我們打算用這字串當作遙控的命令碼。

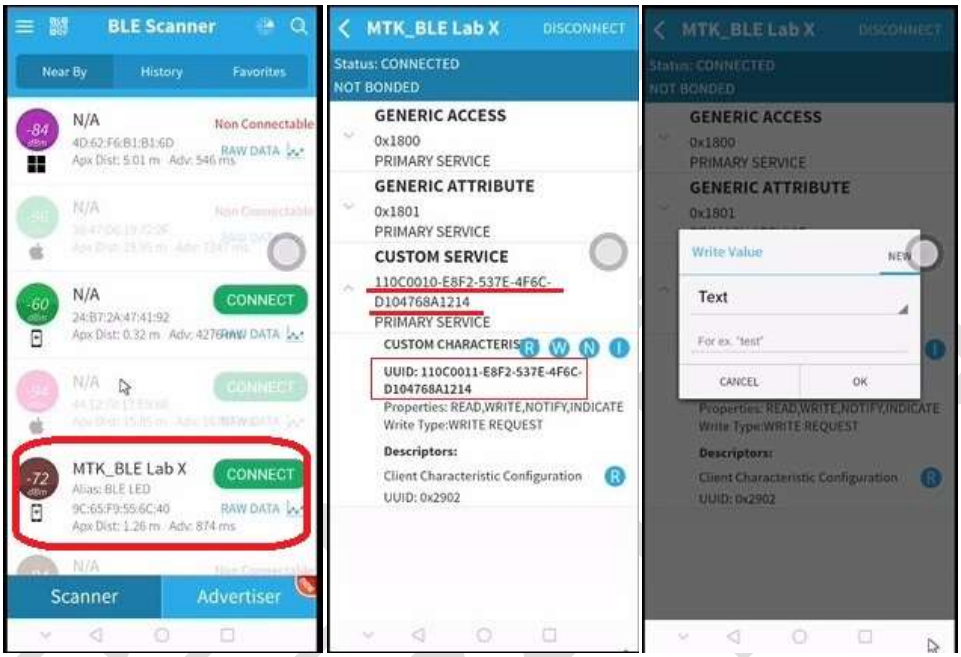
const String value = switchCharacteristic.**getValue**();

## Central 端：

底下提供幾種方式連接到 Filogic 130A 板子的藍芽 BLE，從手機 app 連線到

板子，或從電腦上的工具程式連線到板子，或從另一塊開發板連線。接下來，各項一步步操作。

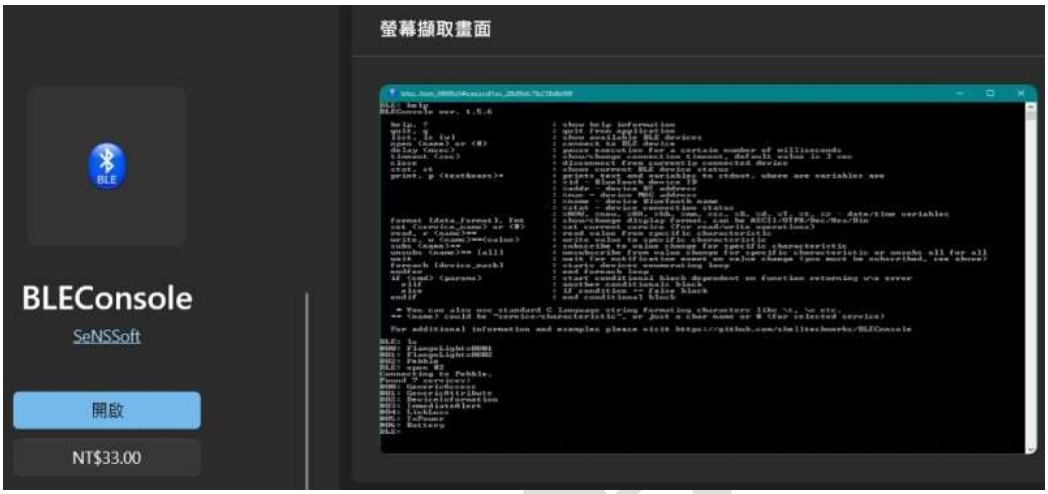
我們手機 app 下載安裝“BLE Scanner”或“B-BLE”，利用這工具連線到板子的 BLE，如下圖二所示，能看到 **MTK_BLE** 的裝置，點擊連線後，便能看到服務與特性的 UUID 碼。最後，我們按下 W，彈出一個小視窗，輸入字串後，再從 Filogic 130A 板子上觀察有沒有收到，而這字串就可以定義為遙控器的命令碼了。



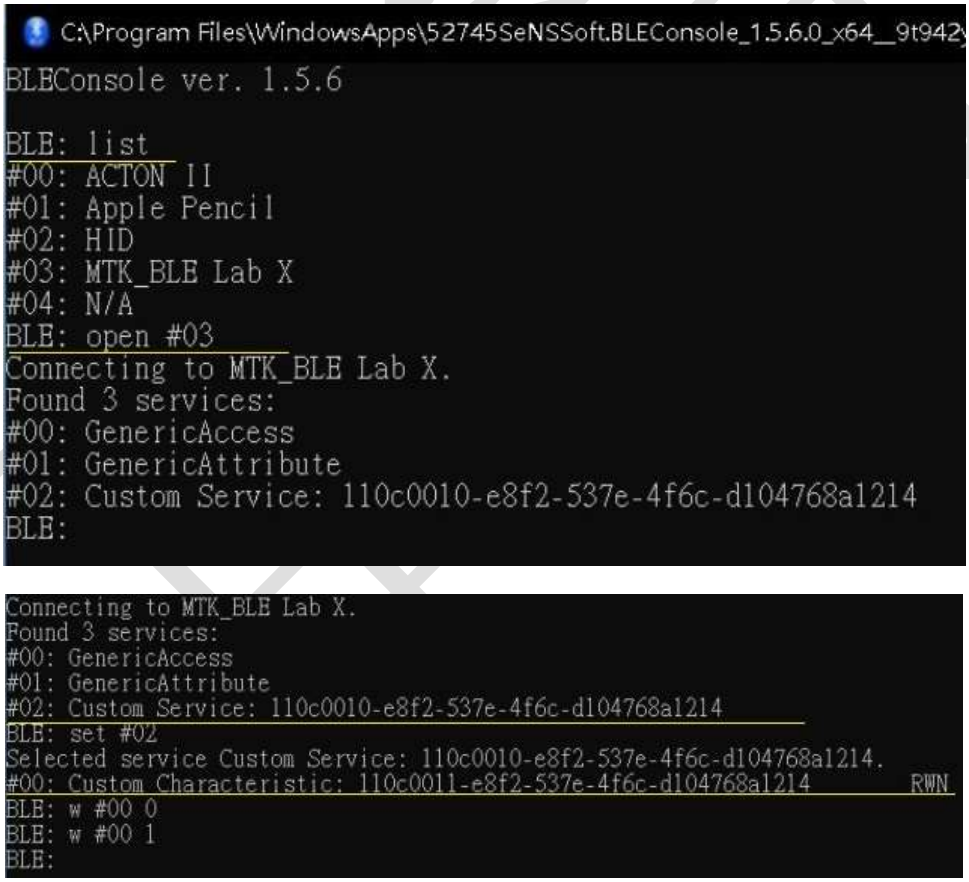

圖二：手機 BLE app 連線到板子

在電腦上，先從 Microsoft Store 裡找到 **BLEConsole** 工具程式，再安裝它，如下圖三。這工具是一個 Console 介面，只能透過命令列輸入，操作上比較不方便。開啟程式後，先輸入“list”指令，便能從畫面看到 BLE 裝置的列表，找到 **MTK_BLE** 裝置的編號後，再輸入“open #03”指令後，進行連線，視窗就會出現 Filogic 130A 板子的服務與特性的 UUID 碼。

此時，在 **BLEConsole** 要輸入“set #02” 指令，也就是選定板子上的**服務**。接下來，我們便可以開始輸入命令碼了，如下圖四所示。輸入“w #00 0” 指令代表從 BLE 的**特性**接口(#00)傳送了字串 ‘0’ 到 Filogic 開發板上，“w #00 1”則代表傳送了字串 ‘1’ ，最後用“close”指令斷開藍芽連線。



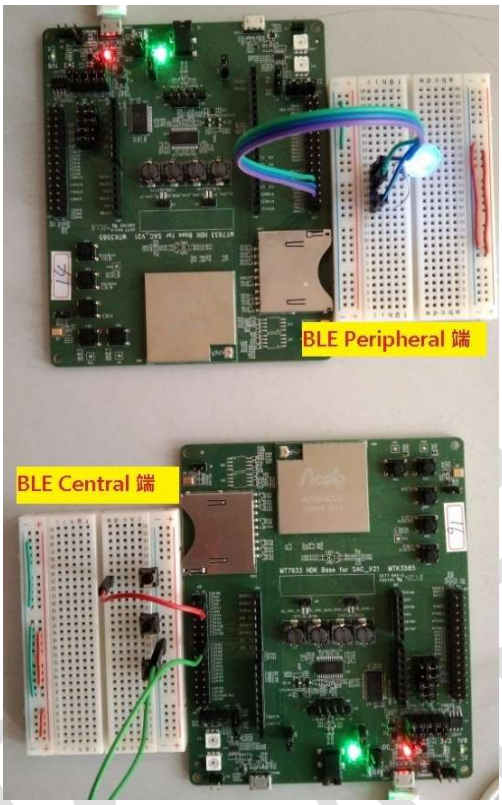

圖三：電腦程式連線到板子




圖四：BLEConsole 指令的操作流程

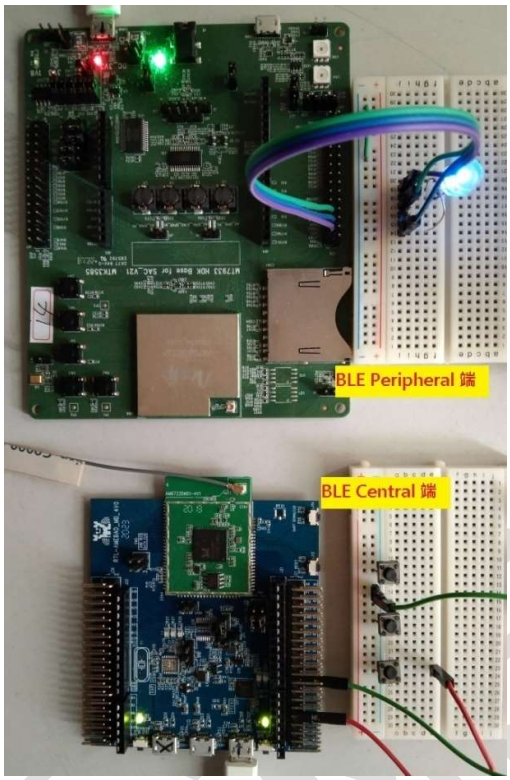
最後一個做法，拿另一塊開發板當作 Central 端，下圖五所示，在 Filogic 130 的 BLE 範例中有一個 **ConnectPeripheral**，我們需要修改一下程式碼，讓它能連上 **MTK_BLE Lab x** 的板子。在 Central 的程式碼中，讀取按鈕的 GPIO 狀態，

當按鈕按下後，透過 BLE 通訊控制另一塊板子的 LED 燈號。




圖五：以兩塊 Filogic 130A 板子，實現 BLE 通訊

除了用另一塊 Filogic 來實現之外，我們也可以拿另一個國產 IC 瑞昱的開發板(阿米巴 AM8722DM)來實現，如下圖六所示，它提供範例 **BLEBatteryClient** 和 BLEUartClient 都是做為 Central 端。我們修改 BLEBatteryClient 範例，讓板子搜尋 BLE 裝置名稱 **MTK_BLE Lab x** 的板子，取得服務與特性 UUID 碼，並連線。與上面同樣的功能設計，AM8722DM 板子偵測按鈕的 GPIO，按鈕的狀態透過藍芽 BLE 控制 Filogic 上面的 LED 燈號。




圖六：Filogic 130A 板與 AM8722DM 板，實現 BLE 通訊

最後，將這個實驗用到的程式碼都放在下面的連結，請自行下載和修改。
https://drive.google.com/file/d/13RBlRpTKldHnEqyK0EBVFL1pMBcE0y0N/view?usp=sharing (基於 SDK 1.0.0 開發)

## 參考資料


[1] Filogic 開發板的初次體驗，一個具有 WiFi 6 和藍芽 5.2 功能的模組：
https://han-ya.blogspot.com/2022/02/mediatek-filogic.html

[2] 漫談 ARM cross compiler 環境：
https://han-ya.blogspot.com/2021/11/arm-cross-compiler.html

[3] 在 Arduino IDE 環境下，也能編譯其他廠商的開發板：
https://han-ya.blogspot.com/2022/05/arduino-ide.html

[4] Filogic130 在 Arduino IDE 的環境配置：https://youtu.be/oZ9UFsgVlxw

[5] 利用光照感測，Filogic 130A 執行簡易的 I/O 控制：
https://youtu.be/C2uD4s_hxiI

[6] WiFi Modbus 在 Filogic 130A 開發板，實現無線的資料收集：
https://youtu.be/TPryHb3H2oM

[7] 智慧音箱的解決方案，以 Filogic 130A 開發板為例：
https://youtu.be/IgmZiBXS1HQ

[8] 藍芽遙控器+聲控的系統 (以 ESP32 與 Filogic 130A 為架構)：
https://youtu.be/esCSL31hE9I

[9] IoT Dashboard 數據可視化，如何在 Filogic 130A 實現資料上傳雲端？
https://youtu.be/jbtSzYi-x7M

[10] 本文的參考範例連結：https://github.com/yijenlu1971/Filogic_onArduino