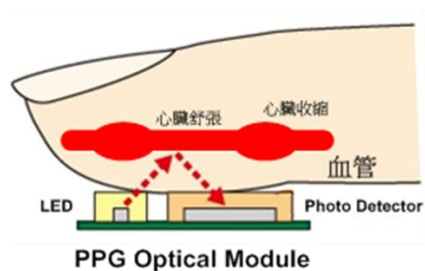


# AWPPG P-type 連續裝置復刻

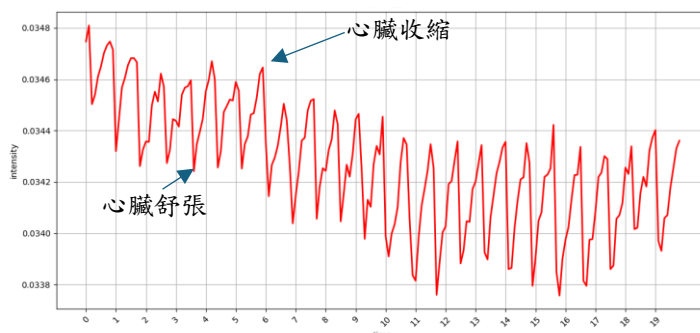
## 一、 引言

本專題旨在製作一套整合了 NSP32 微型光譜儀、ESP32 微控制器 與 TCA6507 LED 驅動器 的 AWPPG (全波段光體積變化描記圖法) 量測裝置。我們將透過一系列系統化的步驟，從底層硬體通訊、光譜資料的採集與儲存，到直觀的人機介面設計與視覺化呈現，逐步建構一個功能完善且使用者友善的便攜式光譜分析系統。最終目標是將所蒐集到的連續型光譜數據，結合機器學習模型，實現即時的生理訊號分析與應用。本文件將詳細闡述復刻過程中的各個環節，包括硬體焊接、韌體撰寫、資料處理與介面開發，為未來的研究與產品化奠定堅實基礎。

PPG (光體積描記圖法) 是一種利用光學原理測量血液體積變化的非侵入性技術。當光照射到皮膚時，隨心臟的搏動，血液的流量會發生改變，導致穿透或反射的光量產生相應的變化，如圖。



▲ PPG 原理示意圖



▲ PPG 訊號 (單波光量變化)

---

## 二、練習

---

### 1. 在 ESP32 上復刻 NSP32 on Arduino 範例程式

NSP32 是微型光譜儀，ESP32 是 MCU 晶片，我們想把光譜的資料拿來做運算、演算，就必須把兩個"小電腦"連結起來做資料傳輸

I. 請先觀看這部介紹 SPI 通訊方式的影片(也可以自行尋找)。

<https://www.youtube.com/watch?v=4zx0s9ZKYZY>

II. 請先參考“[20221120\\_NSP32m\\_datasheet 研讀.pptx](#)”，提供 NSP32 的硬體介面、暫存器配置和通訊協議細節。完整版 NSP32m\_datasheet：

[http://165.227.7.198/Resources/Release/Data%20Sheet\\_NSP32m\\_v1.0.0.pdf](http://165.227.7.198/Resources/Release/Data%20Sheet_NSP32m_v1.0.0.pdf)

III. 前置作業

到 nanoLambda 官網>Resources：<https://nanolambda.myshopify.com/pages/resources>

點選 "Arduino Example" 下載範例教學

點選 "Click here for all free APIs, API documents..." 下載 API

下載 "nsp32\_api\_cpp\_mcu.zip"

路徑 "nsp32\_api\_cpp\_mcu\examples\Arduino\NanoLambdaNSP32\examples" 下有 Arduino 範例程式

路徑 "nsp32\_api\_cpp\_mcu\examples\Arduino\NanoLambdaNSP32\src" 下有 nsp32 相關函式庫



---







#### For NSP32m & NSP32m DBK

A wide range of platforms are supported:

- [NSP32m datasheet](#)
- [DBK Quick Guide](#)
- [nRF52 BLE and Android Example](#)
- [Arduino Example](#)
- [Raspberry Pi Example](#)
- [Desktop C# Example](#), [Java Example](#), [Python Example](#), [Win10 BLE entry example](#).
- [MATLAB and LabVIEW example](#)

Click [here](#) for all free API(s), API documents and example source codes

## Index of /Resources/Release/API/API\_1.0.1/SourceCode

<a href="#">Name</a>	<a href="#">Last modified</a>	<a href="#">Size</a>	<a href="#">Description</a>
 <a href="#">Parent Directory</a>		-	
 <a href="#">MATLAB_Labview_SciWrapper.zip</a>	2020-01-20 07:19	3.0M	
 <a href="#">nsp32_api_cpp_mcu.zip</a>	2020-01-20 07:19	703K	
 <a href="#">nsp32_api_csharp_desktop.zip</a>	2020-01-20 07:19	1.6M	
 <a href="#">nsp32_api_java_android_desktop.zip</a>	2020-01-20 07:19	4.5M	
 <a href="#">nsp32_api_nvthon_desktop.zip</a>	2020-01-20 07:19	6.4M	

#### IV. 採購硬體

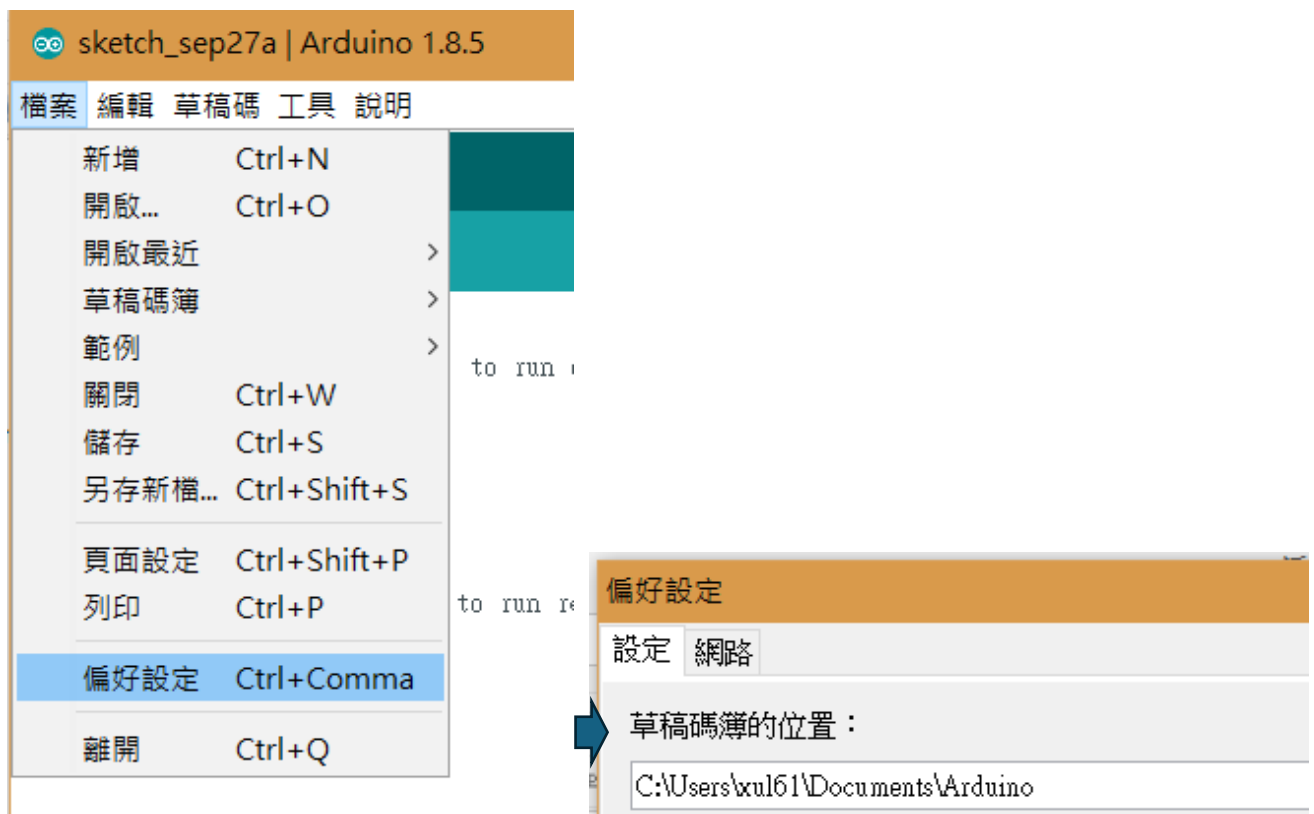
"Bi-directional Level Shifter" \* 2 為 5V 轉 3.3V 的邏輯準位轉換器

"Arduino Uno" \* 1、麵包板 \* 1、杜邦線 若干

以上可到光華的今華電子、祥昌電子購買(或問學長實驗室有無)

記得打統編

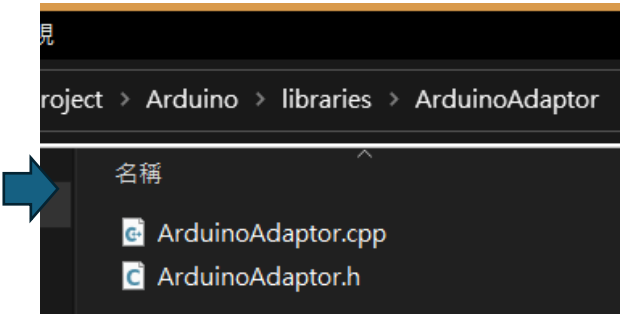
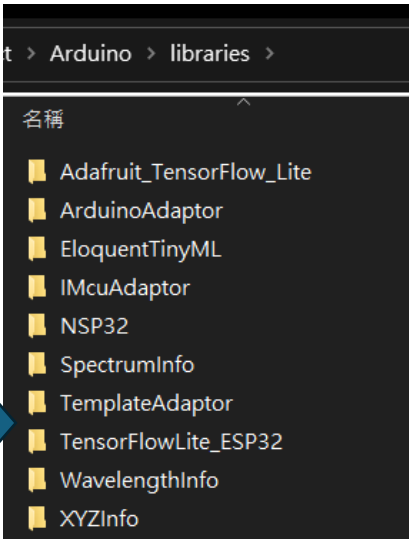
#### V. 在 Arduino IDE 中匯入函示庫



匯入外部函示庫需在草稿簿路徑下新增 "libraries" 資料夾

Software Setup

- 1) Runs on
  - Arduino IDE 1.8.5
- 2) Setup
  - 1. Install Arduino IDE.
  - 2. Put the [/examples/Arduino/NanoLambdaNSP32] folder under [{sketchbook}/libraries/].  
\*This will add NanoLambdaNSP32 to Arduino's libraries. The {sketchbook} path could be found under Arduino IDE > File > Preferences > Sketchbook location.
  - 3. Re-open Arduino IDE, the "Beginner" and "ConsoleDemo" examples should

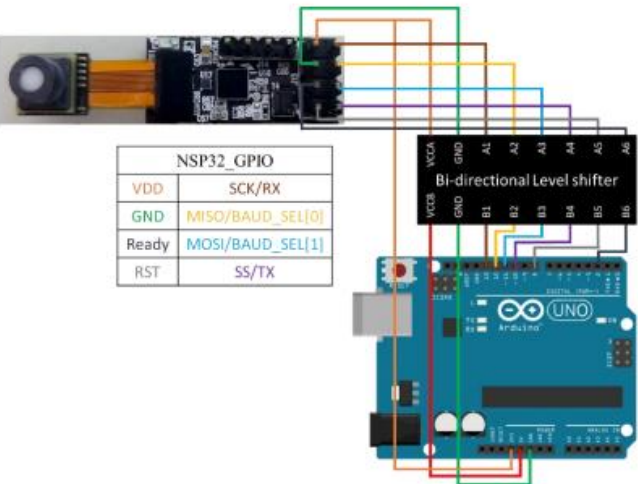


VI. 接線

如 "readme\_Arduino\_example.pdf" 檔案中所接線

2) Setup

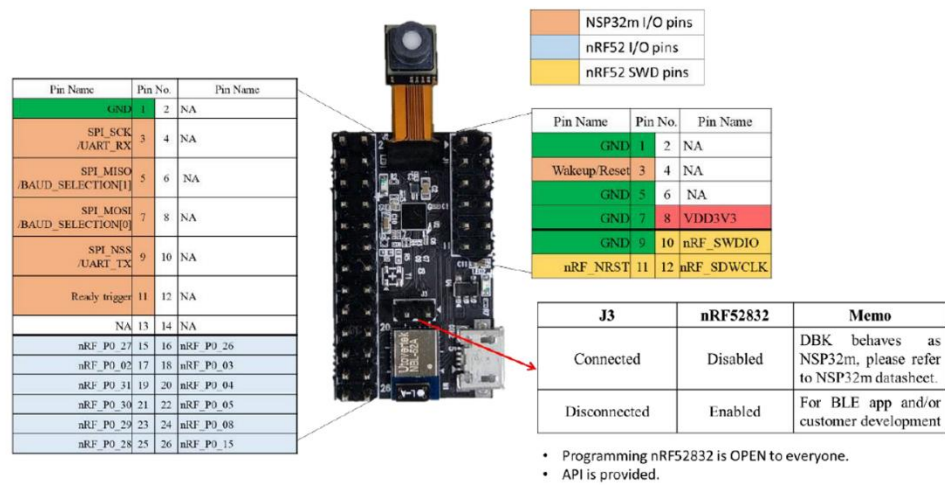
Arduino Uno Example: Hardware connection with SPI/UART through a level-shift:



The following table shows the general pin connections between NSP32 module and the Arduino board.

使用 DBK 可參照下圖接線

## NSP32 DBK



## VII. 執行程式

執行 "Beginner.ino 及 ConsoleDemo.ino"

### 1) Beginner (SPI version)

A clean and simple example for beginners to start with NSP32, to demonstrate the basic usage of the API.

### 2) ConsoleDemo

A console program to demonstrate full functionalities of NSP32. Users can operate NSP32 by interactive console commands.

## 2. 使用 ESP32 連接 NSP32 Print 一維資料

### I. PlatformIO 環境建置

<https://youtu.be/tc3Qnf79Ny8?si=YHI-0SoWeARC6mx1>

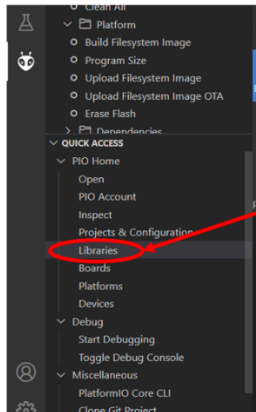
"Ctrl+左鍵"這個功能很重要一定要會用

# PlatformIO



- Visual Code + PlatformIO 教學影片：

<https://youtu.be/tc3Qnf79Ny8?si=YHI-0SoWeARC6mx1>



如果需要找更多library可以從這

compile

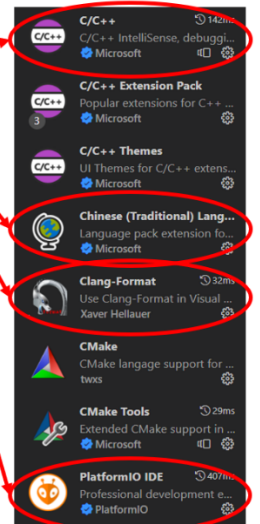
compile&upload

monitor

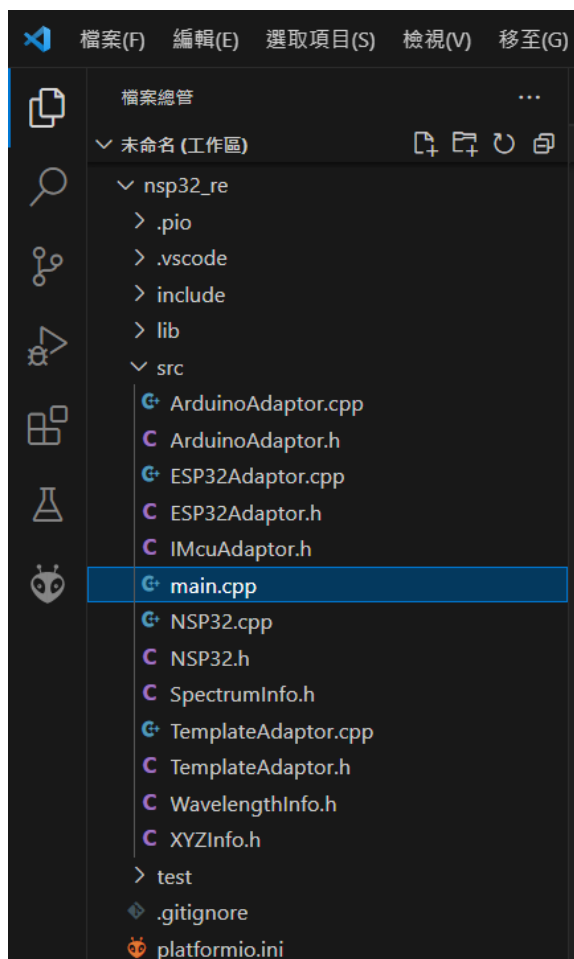
選擇要upload到板子的專案



請把這4個都下載吧!



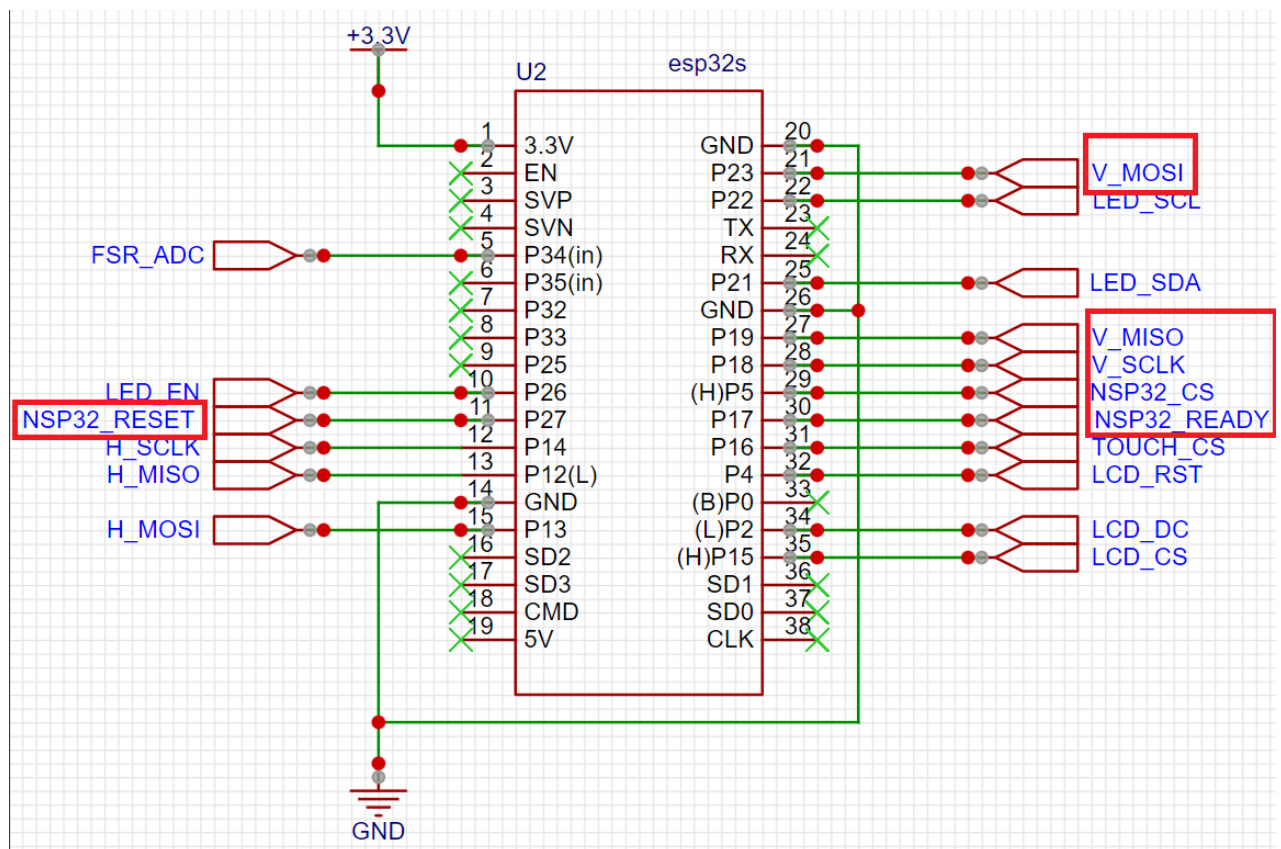
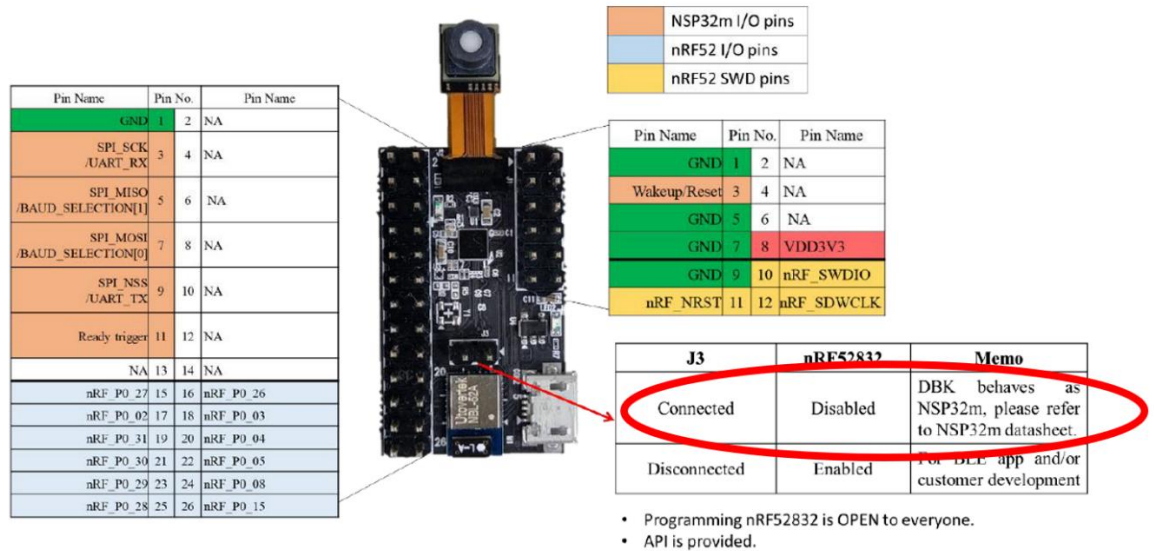
匯入函數庫(直接丟 src 資料夾)



## II. 接線

如圖(使用 DBK 請將上方針腳用短路夾 short)

# NSP32 DBK





	NSP32	Arduino Uno/101	Arduino Mega	Arduino Nano v3	ESP32
SPI					
Signal	Pin	Pin	Pin	Pin	
Wakeup/Reset	RST	8	49	D8	27
SPI SSEL	SS	10	53	D10	5
SPI MOSI	MOSI	11 / ICSP-4	51	D11	23
SPI MISO	MISO	12 / ICSP-1	50	D12	19
SPI SCK	SCK	13 / ICSP-3	52	D13	18
Ready	Ready	2	21	D2	17

### III. 程式(從 Beginner.ino 改，練習用，需自行 debug)

```
#include <ArduinoAdaptor.h>
#include <NSP32.h>

using namespace NanoLambdaNSP32;

/*****
 * modify this section to fit your need
 *****/

const unsigned int PinRst = 27;    // pin Reset
const unsigned int PinReady = 17; // pin Ready

/*****

ArduinoAdaptor adaptor(PinRst);          // master MCU adaptor
NSP32 nsp32(&adaptor, NSP32::ChannelSpi); // NSP32 (using SPI channel)

void PinReadyTriggerISR() {
    // make sure to call this function when receiving a ready trigger from NSP32
    nsp32.OnPinReadyTriggered();
}

void setup() {

    // initialize "ready trigger" pin for accepting external interrupt (falling edge trigger)
    pinMode(PinReady, INPUT_PULLUP); // use pull-up
    attachInterrupt(digitalPinToInterrupt(PinReady), PinReadyTriggerISR, FALLING); // enable interrupt for falling
    edge

    // initialize serial port for "Serial Monitor"
    Serial.begin(115200);
    // initialize NSP32
    nsp32.Init();
    Serial.println("Init");
    // ===== standby =====
    nsp32.Standby(0);
    Serial.println("Standby");
    // ===== wakeup =====
    nsp32.Wakeup();
    Serial.println("Wakeup");
    // ===== hello =====
    nsp32.Hello(0);
```



```

Serial.println("Hello");
// ===== get sensor id =====
nsp32.GetSensorId(0);
Serial.println("GetSensorId");

char szSensorId[15]; // sensor id string buffer
nsp32.ExtractSensorIdStr(szSensorId); // now we have sensor id string in szSensorId[]

Serial.print(F("sensor id = "));
Serial.println(szSensorId);

// ===== get wavelength =====
nsp32.GetWavelength(0);

WavelengthInfo infoW;
nsp32.ExtractWavelengthInfo(&infoW); // now we have all wavelength info in infoW, we can use e.g.
"infoW.Wavelength" to access the wavelength data array

Serial.print(F("Elements of wavelength = "));

//Serial.println(infoW.Wavelength[0]);

for(int i = 0; i < 75; i++){
    Serial.printf("%d ",infoW.Wavelength[i]);
}

// ===== spectrum acquisition =====
nsp32.AcqSpectrum(0, 32, 3, false); // integration time = 32; frame avg num = 3; disable AE

// "AcqSpectrum" command takes longer time to execute, the return packet is not immediately available
// when the acquisition is done, a "ready trigger" will fire, and nsp32.GetReturnPacketSize() will be > 0
while (nsp32.GetReturnPacketSize() <= 0) {
    // TODO: can go to sleep, and wakeup when "ready trigger" interrupt occurs

    nsp32.UpdateStatus(); // call UpdateStatus() to check async result
}

SpectrumInfo infoS;
nsp32.ExtractSpectrumInfo(&infoS); // now we have all spectrum info in infoW, we can use e.g. "infoS.Spectrum"
to access the spectrum data array

Serial.print(F("Elements of spectrum = "));

//Serial.println(infoS.Spectrum[0], 6);
for(int i = 0; i < 75; i++){
    Serial.printf("%.6f ",infoS.Spectrum[i], 6);
}
}

void loop() {
}

```

#### IV. 注意事項

## 心得

- 熟悉SPI各個腳位 & SPI的四種mode

- 更加熟悉NSP32這塊sensor

`SPI.begin(SCLK, MISO, MOSI, CS)`

`SPI.begin(18, 19, 23, 22);`

- 用硬體定義的CS(5)不行，但自己定義的CS(22)卻可行？

ANS：在SPI.begin下加這行

`SPI.setHwCs(true);`

### ESP32 Default SPI Pins

Many ESP32 boards come with default SPI pins pre-assigned. The pin mapping for most boards is as follows:

SPI	MOSI	MISO	SCLK	CS
VSPi	GPIO 23	GPIO 19	GPIO 18	GPIO 5
HSPI	GPIO 13	GPIO 12	GPIO 14	GPIO 15

## V. 結果

```
sensor id = CB-74-0B-2C-33
Number of points = 75
Elements of wavelength =
390 395 400 405 410 415 420 425 430 435 440 445 450 455 460 465 470 475 480 485 490 495 500 505 510 515 520 525 530 535 540 545 550 555 560 565 570 575
5 580 585 590 595 600 605 610 615 620 625 630 635 640 645 650 655 660 665 670 675 680 685 690 695 700 705 710 715 720 725 730 735 740 745 750 755 760
Elements of spectrum =
0.000112 0.000169 0.000163 0.000150 0.000198 0.000260 0.000271 0.000214 0.000152 0.000110 0.000119 0.000163 0.000219 0.000241 0.000209 0.000127 0.0000
54 0.000058 0.000138 0.000258 0.000361 0.000429 0.000436 0.000383 0.000296 0.000249 0.000265 0.000287 0.000266 0.000222 0.000200 0.000230 0.000274 0.0
00354 0.000460 0.000558 0.000582 0.000513 0.000420 0.000369 0.000381 0.000436 0.000505 0.000560 0.000599 0.000623 0.000652 0.000663 0.000656 0.000609
0.000490 0.000315 0.000147 0.000047 0.000060 0.000159 0.000281 0.000361 0.000344 0.000239 0.000100 0.000000 0.000000 0.000003 0.000124 0.000260 0.0002
94 0.000137 0.000000 0.000000 0.000000 0.000068 0.000506 0.000774
Elements of spectrum =
0.000000 0.000000 0.000028 0.000154 0.000288 0.000362 0.000334 0.000215 0.000102 0.000051 0.000121 0.000246 0.000367 0.000418 0.000389 0.000293 0.0001
76 0.000098 0.000094 0.000169 0.000291 0.000389 0.000395 0.000319 0.000231 0.000220 0.000285 0.000361 0.000383 0.000344 0.000281 0.000255 0.000280 0.0
00388 0.000516 0.000589 0.000577 0.000524 0.000509 0.000532 0.000566 0.000584 0.000584 0.000582 0.000590 0.000618 0.000652 0.000662 0.000634 0.000564
0.000448 0.000306 0.000183 0.000105 0.000101 0.000169 0.000270 0.000344 0.000318 0.000194 0.000036 0.000000 0.000000 0.000071 0.000241 0.000372 0.0003
54 0.000127 0.000000 0.000000 0.000000 0.000000 0.000154 0.000507 0.000699
Elements of spectrum =
0.000000 0.000032 0.000099 0.000132 0.000189 0.000263 0.000282 0.000209 0.000117 0.000084 0.000154 0.000255 0.000342 0.000374 0.000349 0.000292 0.0002
07 0.000153 0.000153 0.000202 0.000277 0.000335 0.000330 0.000268 0.000185 0.000172 0.000242 0.000346 0.000416 0.000439 0.000441 0.000460 0.000504 0.0
00589 0.000669 0.000685 0.000602 0.000450 0.000326 0.000293 0.000334 0.000405 0.000467 0.000504 0.000535 0.000569 0.000614 0.000663 0.000672 0.000587
0.000416 0.000226 0.000120 0.000130 0.000198 0.000244 0.000238 0.000213 0.000180 0.000145 0.000077 0.000000 0.000000 0.000000 0.000059 0.000315 0.0004
27 0.000275 0.000000 0.000000 0.000000 0.000000 0.000000 0.000263 0.000602
Elements of spectrum =
0.000000 0.000112 0.000153 0.000180 0.000220 0.000242 0.000195 0.000104 0.000056 0.000089 0.000190 0.000275 0.000304 0.000280 0.000221 0.000144 0.0000
53 0.000016 0.000047 0.000156 0.000310 0.000440 0.000474 0.000411 0.000303 0.000254 0.000285 0.000310 0.000280 0.000217 0.000178 0.000192 0.000235 0.0
00124 0.000173 0.000356 0.000584 0.000471 0.000350 0.000238 0.000182 0.000117 0.000076 0.000055 0.000007 0.000054 0.000077 0.000056 0.000000
```

## 3. 讓燈亮起來

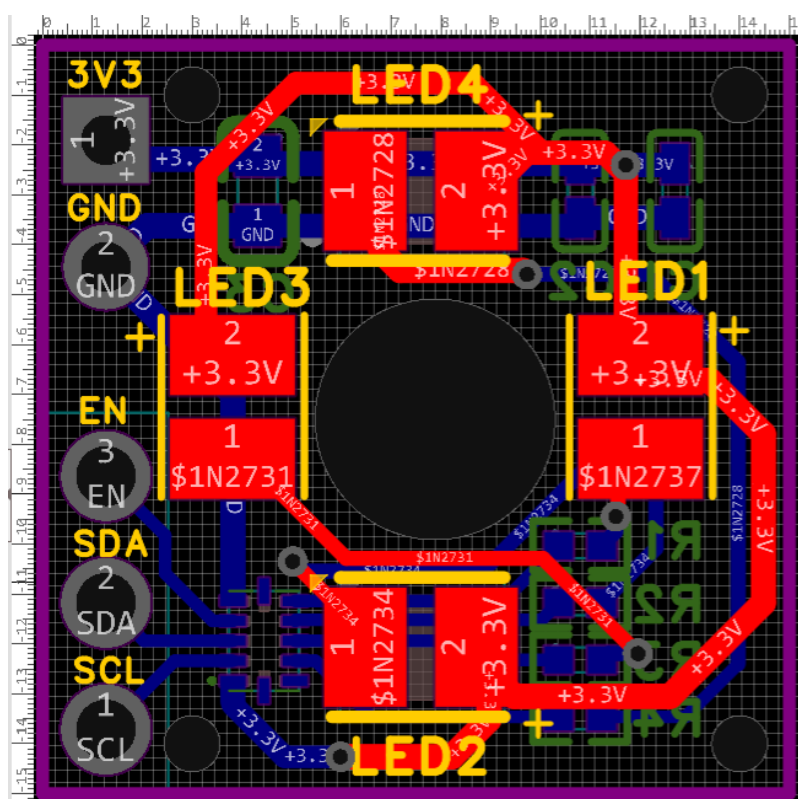
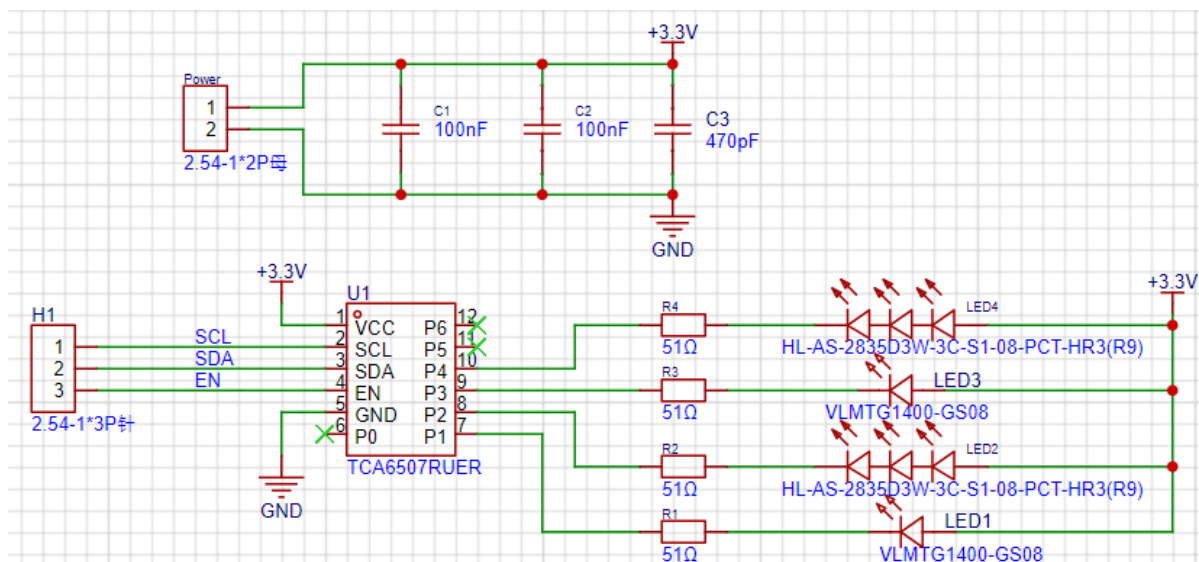
想要量測 PPG 訊號，必須要有“光源”與“光譜儀”，負責“打訊號”與“接收訊號”，光譜儀的部分已經可以順利接收，接下來我們只要把 LED 的訊號輸出，就可以完成 PPG 裝置最重要的部分。

- 請先觀看這部介紹 I2C 通訊方式的影片(也可以自行尋找)。

[https://www.youtube.com/watch?v=u62\\_Rjd5oMY](https://www.youtube.com/watch?v=u62_Rjd5oMY)

- 復刻光源模組(此步可先跳過)

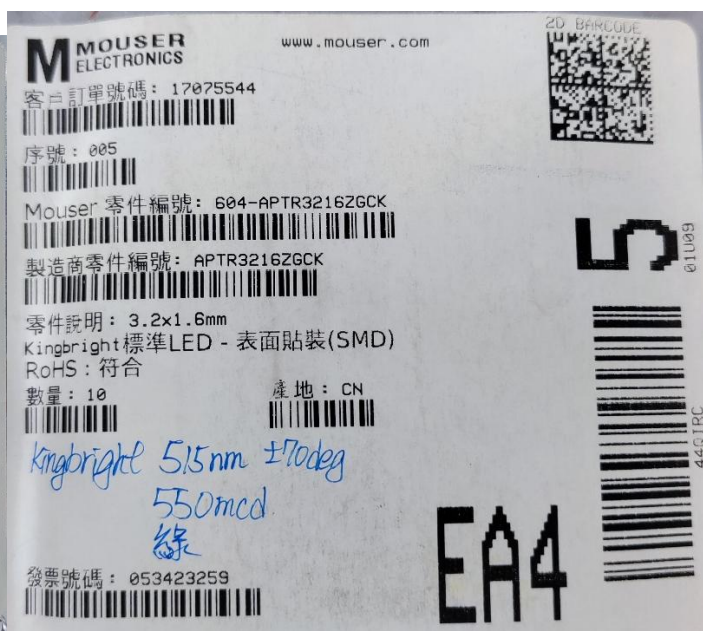
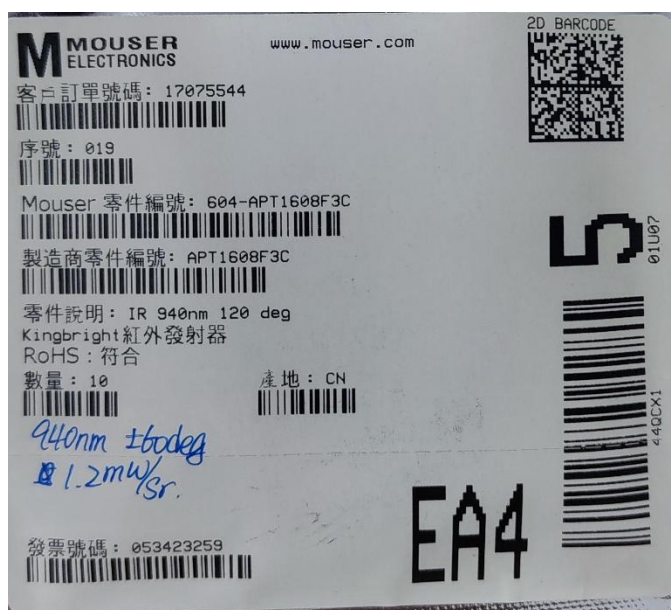
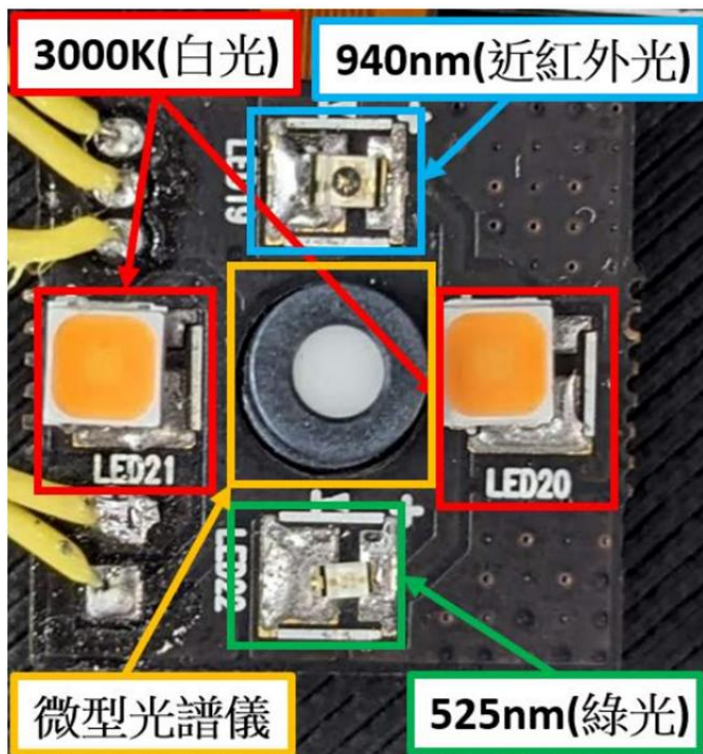
TCA6507 是一顆 LED 驅動器，光源模組上搭載了這個晶片。使用 easyEDA 軟體復刻光源模組，操作方法請自行研讀，參考已完成檔案”[PCB\\_light\\_1.1.epri](#)”



### III. 焊接

請用實驗室前人提供的光源模組，挑選 4 顆 SMD 封裝的 LED，焊接到光源模組上如圖





#### IV. 研讀 TCA6507 datasheet

強烈建議"還是自己讀過一遍 datasheet，這樣對自己比較有幫助

[https://www.ti.com/lit/ds/symlink/tca6507.pdf?ts=1695970436803&ref\\_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FTCA6507%253Futm\\_source%253Dgoogle%2526utm\\_medium%253Dcpc%2526utm\\_campaign%253Dasc-int-null-44700045336317593\\_prodfolderdynamic-cpc-pf-google-](https://www.ti.com/lit/ds/symlink/tca6507.pdf?ts=1695970436803&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FTCA6507%253Futm_source%253Dgoogle%2526utm_medium%253Dcpc%2526utm_campaign%253Dasc-int-null-44700045336317593_prodfolderdynamic-cpc-pf-google-)

[tw\\_int%2526utm\\_content%253Dprodfolddynamic%2526ds\\_k%253DDYNAMIC%2BSEARCH%2BADSD%2526DCM%253Dyes%2526gclid%253DCjwKCAjwyNSoBhA9EiwA5aYlb7JddEi5t0lPffNHRYaTY-DATnTsM2m6wT50RoShAkQ86KoaUz1-oRoCvgYQAvD\\_BwE%2526gclsrc%253Daw.ds](http://tw_int%2526utm_content%253Dprodfolddynamic%2526ds_k%253DDYNAMIC%2BSEARCH%2BADSD%2526DCM%253Dyes%2526gclid%253DCjwKCAjwyNSoBhA9EiwA5aYlb7JddEi5t0lPffNHRYaTY-DATnTsM2m6wT50RoShAkQ86KoaUz1-oRoCvgYQAvD_BwE%2526gclsrc%253Daw.ds)

讀過 datasheet 之後發現要讓燈亮只需更改 Select0、Select1、Select2 三個暫存器(register)

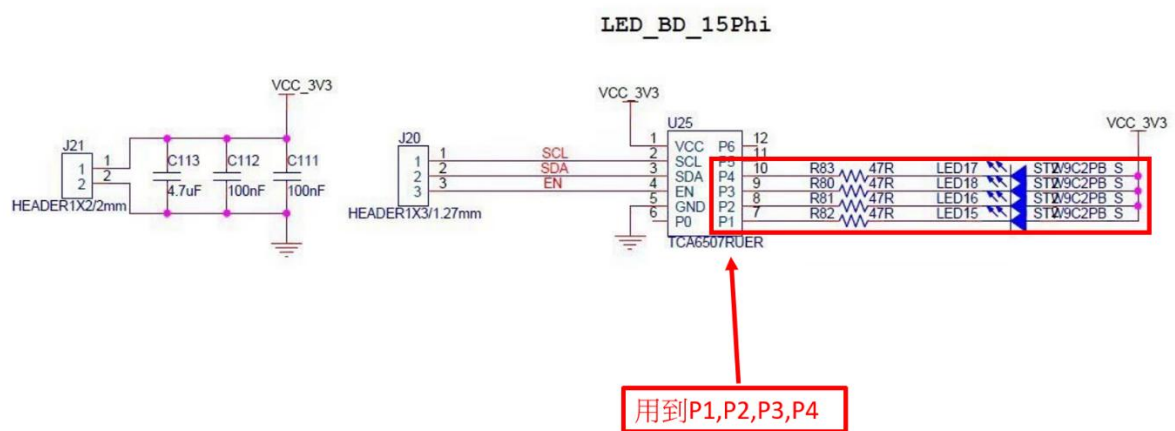
Fully on (開到最大亮度)

### Table 8. Select2, Select1, and Select0 Register States

SELECT2	SELECT1	SELECT0	STATE
0	0	0	LED off (high impedance).
0	0	1	LED off (high impedance).
0	1	0	LED on with maximum intensity value of PWM0 (ALD value or BRIGHT_F0 value, depending on One Shot / Master Intensity Register setting).
0	1	1	LED on with maximum intensity value of PWM1 (ALD value or BRIGHT_F1 value, depending on One Shot / Master Intensity Register setting).
1	0	0	LED fully on (output low). Can be used as general-purpose output.
1	0	1	LED on at brightness set by One Shot / Master Intensity register.
1	1	0	LED blinking with intensity characteristics of BANK0 (PWM0).
1	1	1	LED blinking with intensity characteristics of BANK1 (PWM1).

我們只需控制 P1、P2、P3、P4 四個腳位

# 光板電路圖



所以我們只需做以下更改，即可將燈打開

## Fully on (開到最大亮度)

Table 9. Register 0x00 (Select0 Register)

BIT	S0-7	S0-6	S0-5	S0-4	S0-3	S0-2	S0-1	S0-0
DEFAULT	X <sup>(1)</sup>	0	0	0	0	0	0	0

(1) X = Don't care.

Table 10 show the Register 0x01 (Select1 Register).

Table 10. Register 0x01 (Select1 Register)

BIT	S1-7	S1-6	S1-5	S1-4	S1-3	S1-2	S1-1	S1-0
DEFAULT	X <sup>(1)</sup>	0	0	0	0	0	0	0

(1) X = Don't care.

Table 11 show the Register 0x02 (Select2 Register).

Table 11. Register 0x02 (Select2 Register)

BIT	S2-7	S2-6	S2-5	S2-4	S2-3	S2-2	S2-1	S2-0
DEFAULT	X <sup>(1)</sup>	0	0	1	1	1	1	0

(1) X = Don't care.

結論：其實我們只需要改動 Select2 這個暫存器的值就可以控制燈的亮滅

亮：Select2 = 00011110 = 0x1E

滅：Select2 = 00000000 = 0x00

### V. I2C 相關知識研讀(此步可跳過)

韌體開發前請先研讀 I2C 相關知識包含：需要幾條線、電路、接線、時序圖、START Condition、Stop Condition...等，參考資料 or 自行 Google：

<https://wiki.csie.ncku.edu.tw/embedded/I2C>

<https://rexpighj123.pixnet.net/blog/post/219960237>

### VI. 撰寫韌體關鍵資訊(TCA6507 datasheet)

#### 8.5.2.1 Writes

To write on the I<sup>2</sup>C bus, the master sends a START condition on the bus with the address of the slave, as well as the last bit (the R/W bit) set to 0, which signifies a write. After the slave sends the acknowledge bit, the master then sends the register address of the register to which it wishes to write. The slave acknowledges again, letting the master know it is ready. After this, the master starts sending the register data to the slave until the master has sent all the data necessary (which can be only a single byte), and the master terminates the transmission with a STOP condition.

Figure 14 shows an example of writing a single byte to a register.

- ☒ Master controls SDA line
- ☐ Slave controls SDA line

#### Write to one register in a device

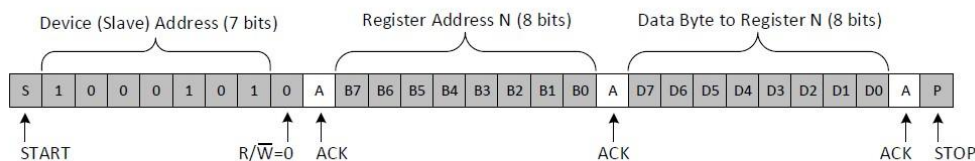


Figure 14. Write to Register

Figure 15 shows an example of writing to a Fully On register.

- ☒ Master controls SDA line
- ☐ Slave controls SDA line



### 8.5.2.2 Reads

Reading from a slave is very similar to writing, but requires some additional steps. To read from a slave, the master must first instruct the slave which register it wishes to read from. This is done by the master starting off the transmission in a similar fashion as the write, by sending the address with the R/W bit equal to 0 (signifying a write), followed by the register address it wishes to read from. Once the slave acknowledges this register address, the master sends a START condition again, followed by the slave address with the R/W bit set to 1 (signifying a read). This time, the slave acknowledges the read request, and the master releases the SDA bus, but continues supplying the clock to the slave. During this part of the transaction, the master becomes the master-receiver, and the slave becomes the slave-transmitter.

The master continues to send out the clock pulses, but releases the SDA line so that the slave can transmit data. At the end of every byte of data, the master sends an ACK to the slave, letting the slave know that it is ready for more data. Once the master has received the number of bytes it is expecting, it sends a NACK, signaling to the slave to halt communications and release the bus. The master follows this up with a STOP condition.

Figure 16 shows an example of reading a single byte from a slave register.

- ☒ Master controls SDA line
- ☐ Slave controls SDA line

#### Read from one register in a device

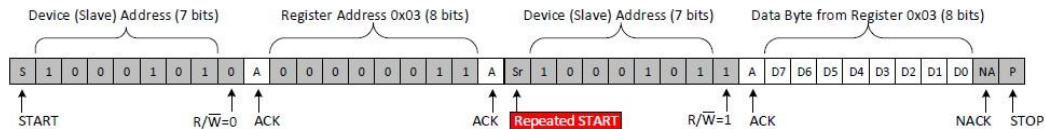


Figure 16. Read From Register Example

### 8.5.3 Device Address

The address of the TCA6507 is shown in Figure 17.

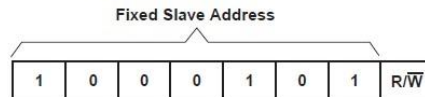


Figure 17. TCA6507 Address

## VII. 韌體程式(練習用，需自行 debug)

以下程式應用 Wire.h 完成 Datasheet 上要求的規則，達到傳輸資料的效果

```
#include <Wire.h>

//ESP32 I2C 腳位 & TCA6507 Enable
#define SDA 21
#define SCL 22
#define EN 26

enum TCA_registers{
    SELECT0 = 0x00,
    SELECT1 = 0x01,
    SELECT2 = 0x02,
    FADE_ON_TIME = 0x03,
    FULLY_ON_TIME = 0x04,
    FADE_OFF_TIME = 0x05,
    FIRST_FULLY_OFF_TIME = 0x06,
    SECOND_FULLY_OFF_TIME = 0x06,
    MAXIMUM_INTENSITY = 0x08,
    ONE_SHOT = 0x09,
    INITIALIZATION = 0x10,
};

int8_t TCA_writeBytes(uint8_t devAddr, uint8_t regAddr, uint8_t length, uint8_t *data){
    uint8_t status = 0;

    Wire.beginTransmission(devAddr);
    Wire.write(regAddr);

    for (uint8_t i = 0; i < length; i++) {
        Wire.write((uint8_t) data[i]);
    }
}
```



```

    status = Wire.endTransmission();
    return status == 0;
}

int8_t TCA_writeByte(uint8_t devAddr, uint8_t regAddr, uint8_t data){
    return TCA_writeBytes(devAddr, regAddr, 1, &data);
}

int8_t TCA_read(uint8_t devAddr, uint8_t regAddr, uint8_t length, uint8_t *data, uint16_t timeout){

    int8_t count = 0;
    uint32_t t1 = millis();

    for (uint8_t k = 0; k < length; k += std::min<int>(length, BUFFER_LENGTH)) {
        Wire.beginTransmission(devAddr);
        Wire.write(regAddr);
        Wire.endTransmission();

        Wire.requestFrom(devAddr, (uint8_t) std::min<int>(length - k, BUFFER_LENGTH));

        for (; Wire.available() && (timeout == 0 || millis() - t1 < timeout); count++) {
            data[count] = Wire.read();
        }
    }

    if (timeout > 0 && millis() - t1 >= timeout && count < length) count = -1; // timeout

    return count;
}

bool TCA_isConnected(){
    uint8_t result = 0;

    return ( TCA_read(device_address, SELECT0, 1, &result, 0) == 1);
}

void setup() {
    Serial.begin(115200);

    Wire.begin(SDA, SCL);

    pinMode(EN, OUTPUT);
    digitalWrite(EN, LOW);
    delay(20);
    digitalWrite(EN, HIGH);
    delay(20);

    if(TCA_isConnected()){
        Serial.print("TCA6507 connected!!!");
    }
    else{
        Serial.print("TCA6507 doesn't connected.");
    }
}

void loop() {
    //on_off 在後續 LCD 螢幕上可以用觸控來控制
    //先試試用 Serial port 控制 或 讓 on_off 每隔 1 秒+1 讓燈閃爍
    if(on_off % 2 == 1){
        uint8_t selects[3] = {0x00, 0x00, 0x1E};
        TCA_writeByte(device_address, SELECT2, selects[2]);
    }
}

```

```

if(on_off % 2 == 0){
    TCA_writeByte(device_address, SELECT2, 0x00);
}
}
}

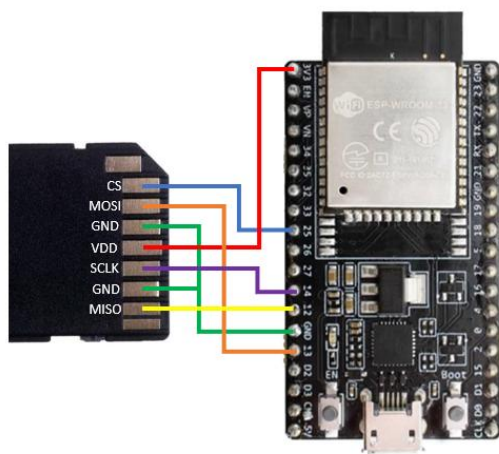
```

## 4. 把資料存起來

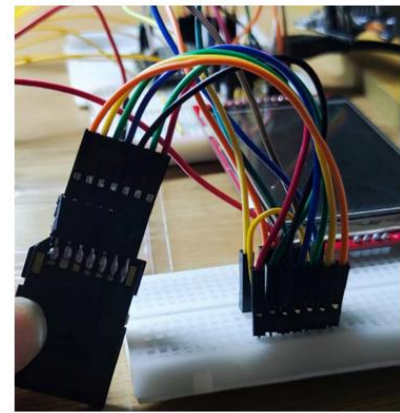
已經有了光譜儀和光源，但是資料卻沒有留存，使用 SD 卡的方式將資料存起來吧。

### I. 硬體接線

使用 TF 轉接卡焊接後連接 esp32，實際 layout 電路板直接使用 Micro-SD 卡模組焊接



SD card	ESP32S
MISO	12(HSPI)
GND	GND
SCLK	14(HSPI)
VDD	3V3
GND	GND
MOSI	13(HSPI)
CS	25



### II. SD 卡規格

採用 SDHC 類型，速度買越快越好，例如



### III. 程式學習

使用 SD.h 函式庫練習，參考 <https://randomnerdtutorials.com/esp32-microsd-card-arduino/#microsdcardsize>

檔案格式可直接改，將範例中.txt 改成.csv，就能存成 excel，csv 的格式是用“，”隔開

## IV. 練習

將 nsp32 所採集的資料依次存進一個 csv 檔，總共存 20 筆資料

(使用生成式 AI 幫助你)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	0.141423	0.124415	0.087908	0.071854	0.069097	0.046761	0.010237	0	0	0.018228	0.067754	0.096066	0.095013	0.083144	0.077954	0.086932	0.113054	0.126783	0.129155	0.099534	0.054699	0.007133	
2	0	0	0	0.009317	0.093798	0.152503	0.148933	0.089187	0.013224	0	0	0	0	0.023549	0.035865	0.02282	0.014336	0.024739	0.055826	0.074221	0.066035	0.032937	0.000
3	0	0	0	0	0.064525	0.128475	0.136255	0.094401	0.039196	0.066434	0.00531	0.017953	0.033385	0.044016	0.051031	0.059086	0.068439	0.074929	0.080002	0.071851	0.05806	0.045984	0.038
4	0	0.00903	0.027235	0.069954	0.080887	0.067374	0.044753	0.022684	0.002714	0	0	0	0	0	0.016162	0.046442	0.067314	0.081162	0.083788	0.082268	0.082158	0.085455	0.091
5	0.069846	0.077434	0.090922	0.079291	0.03051	0	0	0	0	0	0.060438	0.107107	0.129181	0.127548	0.118058	0.102975	0.081057	0.046339	0.010349	0	0.011752	0.047056	0.075
6	0.109613	0.075724	0.072789	0.042146	0.040855	0.038135	0.025566	0.014048	0.018211	0.043863	0.059293	0.05551	0.025852	0	0	0	0	0.03333	0.065205	0.0592	0.035048	0.01589	0.016
7	0	0	0.016486	0.105553	0.084878	0	0	0	0	0	0	0	0	0	0	0	0	0.00974	0.025895	0.030141	0.025388	0.023446	0.032
8	0.083899	0.079176	0.051749	0.040215	0.031625	0.017715	0.008896	0.007676	0.020762	0.038704	0.051133	0.048329	0.037319	0.026648	0.021976	0.024617	0.033393	0.034727	0.028767	0.013538	0.004815	0.007401	0.0
9	0.147409	0.113005	0.065441	0	0	0	0	0.062171	0.105938	0.112457	0.090755	0.041707	0	0	0	0.008339	0.026457	0.028362	0.031751	0.04288	0.054674	0.047987	0.02
10	0	0	0	0	0	0.07972	0.141427	0.142272	0.101737	0.054027	0.027276	0	0	0	0	0.015623	0.025701	0.038528	0.06509	0.093084	0.102114	0.073691	0.032
11	0.020094	0.015778	0.043245	0.056037	0.046636	0.005412	0	0	0	0	0	0.022213	0.036945	0.028473	0.017888	0.024222	0.037713	0.046475	0.035655	0.012261	0	0	0
12	0.177446	0.12658	0.077017	0	0	0	0	0.042172	0.077248	0.092398	0.086202	0.064727	0.036847	0.009997	0	0	0	0.000061	0.008211	0.00669	0.009016	0.022666	0.043
13	0.001743	0	0	0	0.015128	0.016552	0	0	0	0	0.043221	0.066823	0.062372	0.035685	0.017165	0.017259	0.025384	0.029643	0.030531	0.028075	0.023414	0.013534	0.006
14	0.087017	0.06503	0.023	0	0	0	0.003268	0.034427	0.063119	0.082393	0.082092	0.063493	0.028552	0	0	0	0	0	0.056327	0.098771	0.117538	0.10	0
15	0.079126	0.086205	0.085751	0.081756	0.049812	0.001558	0	0	0	0	0	0.031422	0.085466	0.142511	0.165856	0.132321	0.071899	0.021002	0	0	0	0.002801	0.017
16	0	0	0	0	0.015555	0.04239	0.035535	0.005286	0	0	0.018824	0.041367	0.031024	0.005272	0	0.001255	0.037868	0.06884	0.081748	0.057928	0.018908	0	0.001
17	0.065014	0.076425	0.046894	0.049125	0.054681	0.080347	0.078085	0.048747	0.011485	0	0	0.01038	0.025637	0.038987	0.052556	0.06874	0.084449	0.087098	0.084858	0.064198	0.031472	0	0
18	0.322255	0.282859	0.206434	0.107012	0.023227	0	0	0	0	0	0	0.004164	0.036226	0.082992	0.119712	0.127094	0.101999	0.062876	0.030824	0.017005	0.011049	0	0
19	0.225028	0.168847	0.108718	0.013145	0	0	0	0	0.024859	0.06967	0.105697	0.118661	0.120258	0.106318	0.079317	0.042584	0	0	0	0	0	0.016299	0.033
20	0	0	0	0.013749	0.050465	0.078256	0.081504	0.052345	0.006021	0	0	0	0.00348	0.017435	0.03799	0.055898	0.056351	0.04505	0.031537	0.015976	0.001929	0	0.006

## 5. 打造自己的人機介面吧

先前我們學會裝置的三大重點，燈(送訊號)、光譜儀(收訊號)、SD 卡(存訊號)，但一直沒有一個簡單、方便的操作，這次我們拿一塊 ILI9341 TFT LCD 螢幕來打造這個 UI 介面。

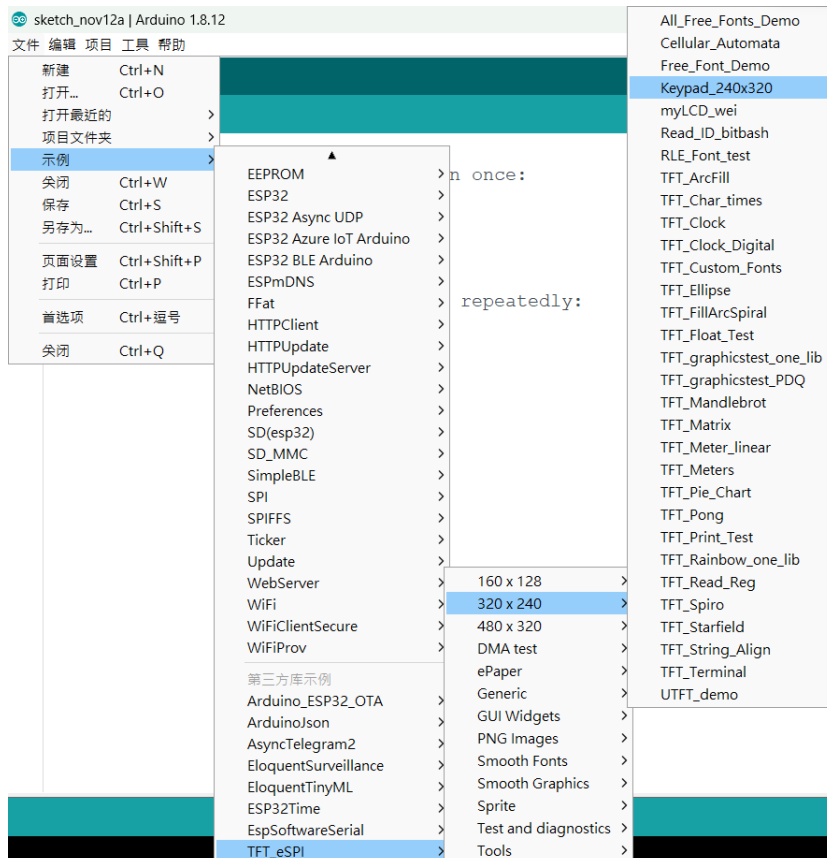
### I. TFT-LCD 練習

跟隨影片學習：[https://www.youtube.com/watch?v=rq5yPJbX\\_uk&ab\\_channel=XTronical](https://www.youtube.com/watch?v=rq5yPJbX_uk&ab_channel=XTronical)

注意:影片中是使用 ArduinoIDE 環境，但我們之後需要整合程式，使用 platformIO 環境

### II. 練習範例

TFT-LCD 螢幕連接 ESP32\_顯示鍵盤範例程式 with 觸控按鈕





### III. 打造 UI

利用 library 中的函式打造一個屬於自己，並且可以開關“LED 燈”&“NSP32”的 UI 介面  
(optional:嘗試把光譜訊號 print 到螢幕上)

---

## 三、 整合

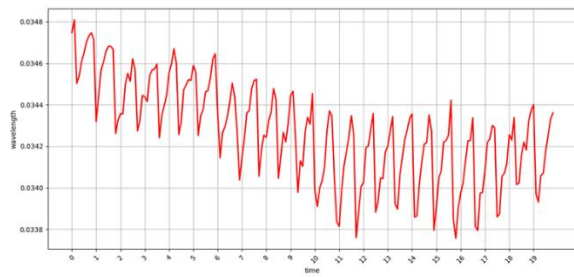
---

### 6. 在螢幕上顯示 3D 光譜及完成程式整合

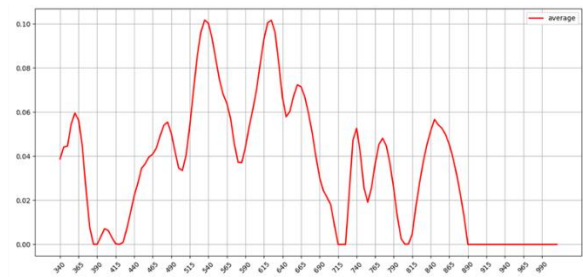
目前我們收到的資料是一串 1 維陣列，一般人很難理解它是甚麼東西，若能及時地看到圖像，使用者會馬上進入狀況，甚至對這個東西產生好奇心

#### I. 何謂 3D 光譜

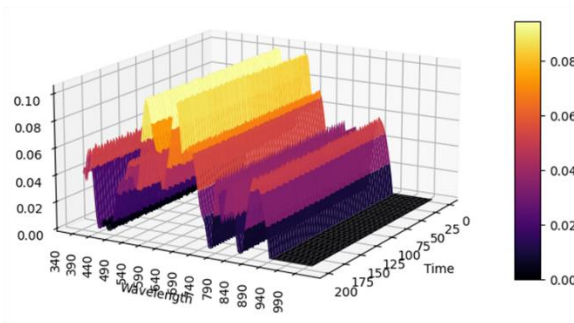
以前我們從 nsp32 所收到的資料都是一串一維 array，列出來看其實就是 340,345,350...1010nm 的個別波長的光量，把它製成圖片就如圖 2，上一個版本做的就是這個。所謂的 3D-PPG 就是指把時間軸加進去，如圖 3。時頻譜就是把高度(光量)壓扁改成用顏色顯示，縱軸是波長，橫軸是時間，這也是此裝置所要顯示的圖，如圖 4。PPG 就是指單一波長(例如：550nm)隨著時間光量的變化，如圖 1，也可以是所有波長相加後光量的變化。



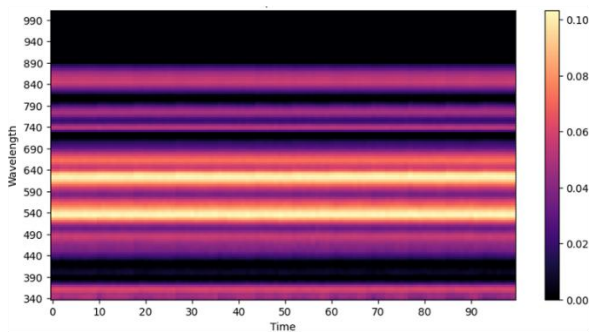
▲圖1 單一波段的光量變化



▲圖2 瞬時各波段的光量



▲圖3 3D圖(加入時間軸)



▲圖4 時頻譜 (3D圖高度轉顏色)

## II. UI 介面所需功能

專題所用之版本：

按鍵 Start --用於量測光譜

按鍵 Detect --用於模型預測

按鍵 Reset --用於裝置重啟

顯示量測時間、時頻譜、儲存檔案名稱、模型預測結果

新版更動：

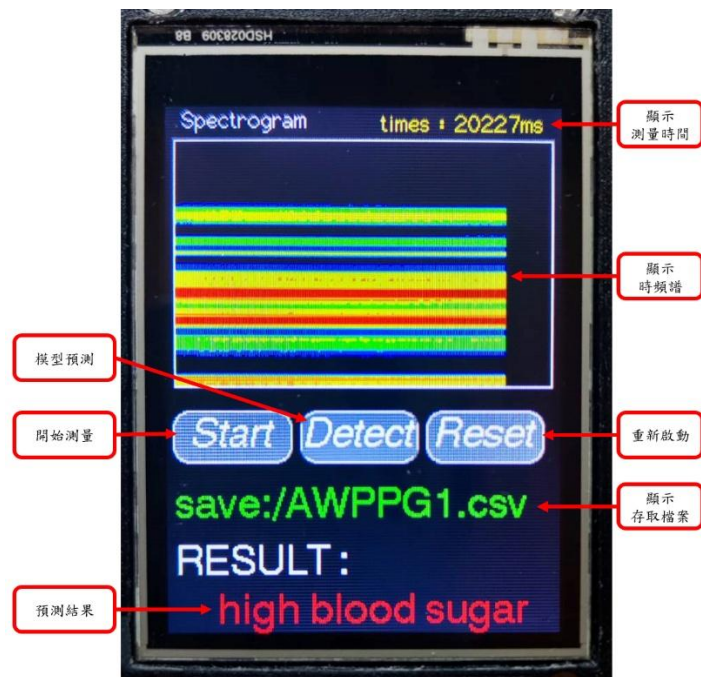
移除按鍵 Detect 改為 PPG 用於顯示 PPG 波形

移除顯示模型預測結果

新增可調 Integration time(按鍵+、按鍵-)與自動曝光(按鍵 AE)

顯示 Integration time 數值





專題版



新版

注：Integration time 為調整曝光時間參數，為 AWPPG 量測重要因素

### III. 整合程式重現 AWPPG 的 UI 介面及功能

請整合前述所有的程式，並實現上述所介紹的 UI 介面，可參考 [AWPPG1.0.zip](#)、[AWPPG1.2.zip](#)、[AWPPG2.1.zip](#)

若無法完成至少要讀懂以上檔案

## 7. 完成硬體裝置

先前我們做的東西都是插麵包版根本不像個裝置，所以我們需要 PCB 板讓 MCU、nsp32、光源模組、SD 卡連接。使用 3D 列印設計外殼

## I. PCB --軟體

使用軟體 >> Easy-EDA

基本操作：[https://www.youtube.com/watch?v=6\\_AXMmLOVB8](https://www.youtube.com/watch?v=6_AXMmLOVB8)

Create your own Component(7 分 19 秒開始)：

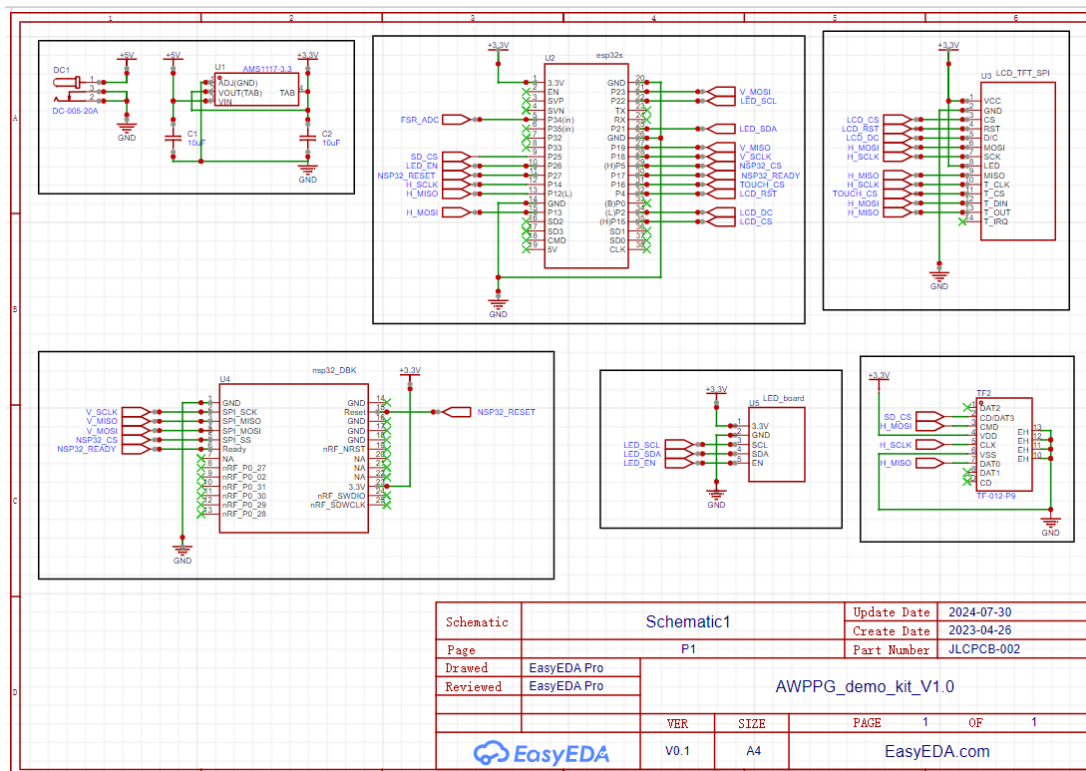
<https://www.youtube.com/watch?v=utBQqcuOt9U>

也可自行查找資料熟悉軟體

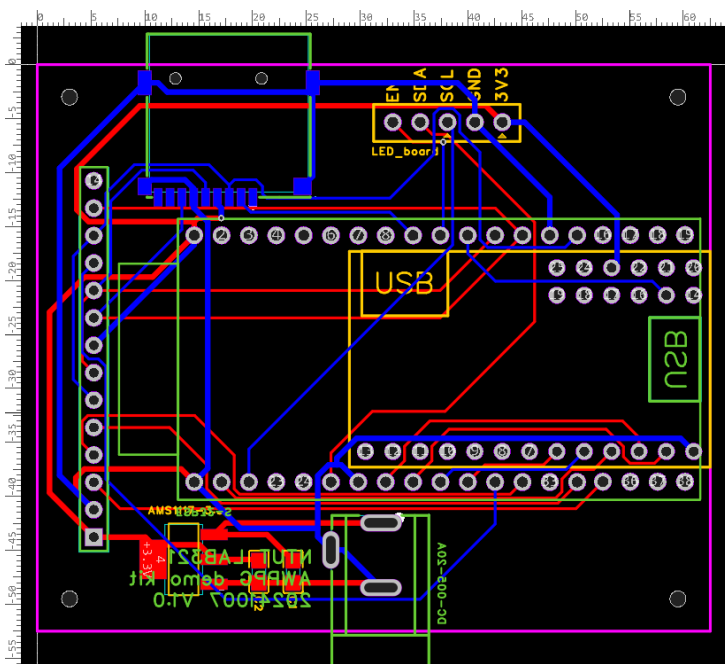
注意：自己畫 Footprint 時要小心測量，最好拿游標卡尺

參考已完成檔”[AWPPG\\_demo\\_kit\\_V1.0.eprj](#)”

## II. PCB --電路圖



## III. PCB --Layout 圖





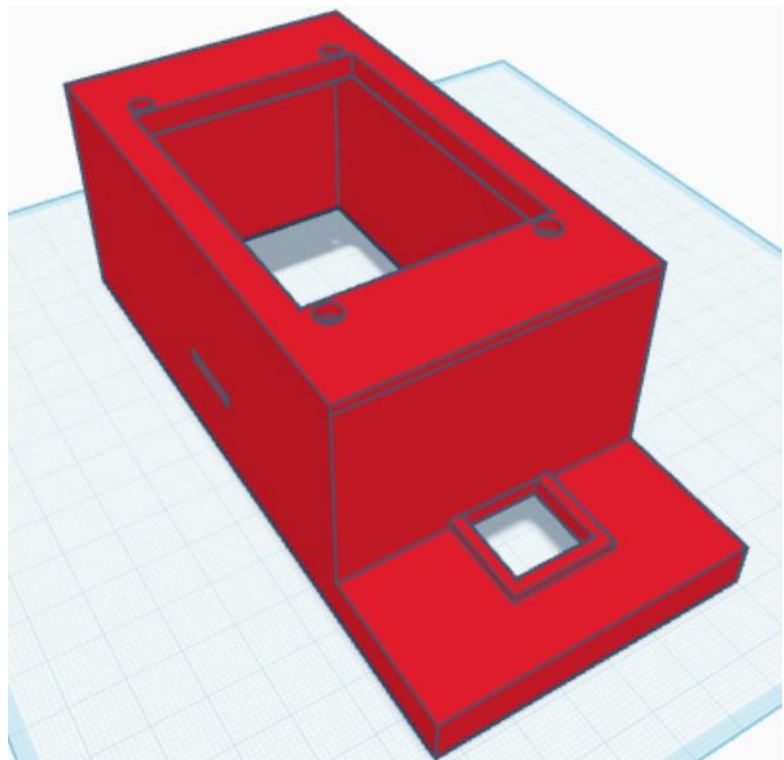
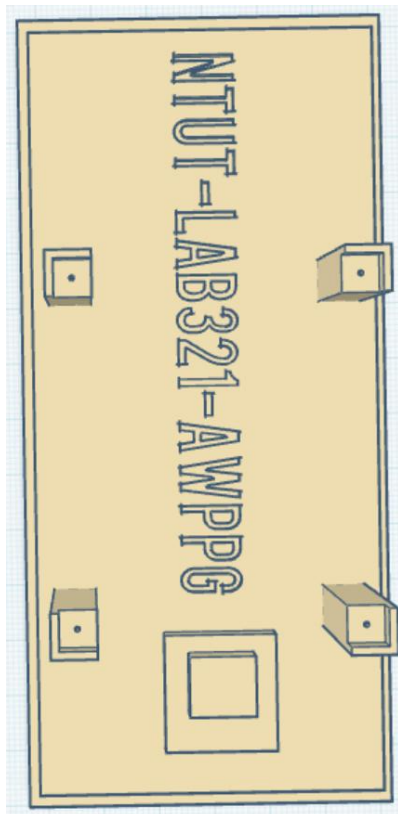
#### IV. PCB --焊接

材料：SMD 電容  $10\mu\text{F}$  \*2、AMS1117-3.3、2.1mm DC 插座、Hanbo Electronic TF-012-P9、2.54mm 排針、排針母座 19P \*2、排針母座 13P、排針母座 6X2P

#### V. 外殼設計

1. 使用 Tinkercad 繪製 3D 圖(用自己習慣的軟體也可以)，匯出使用 .STL 檔
2. 使用 Ultimaker 切片軟體轉檔 (PLA 材料溫度:230/70)，匯出為 .gcode 檔
3. 拿到 3D 印表機列印 (用黑色料)
4. 列印完成後進行打磨

目前採用：[cover2.0.stl](#)、[Base2.0.stl](#)

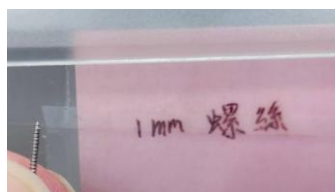
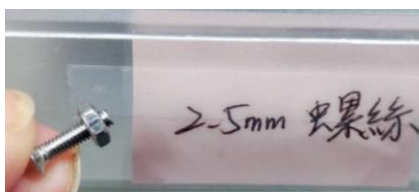


#### VI. 組裝全部零件並載入程式

鏡頭與光源模組的部分用膠帶黏起來，跟模組接觸的地方膠帶多一層反貼。



額外零件：2.5mm 螺絲\*4、1mm 螺絲\*4、19pin 排插、黑色 OK 線



成品：



## 8. 模型部署

目前裝置沒有開發、部署 AWPPG 連續型模型，但有 PPG 預測血糖的簡易模型(實務上無法使用，準確率很低)可以練習

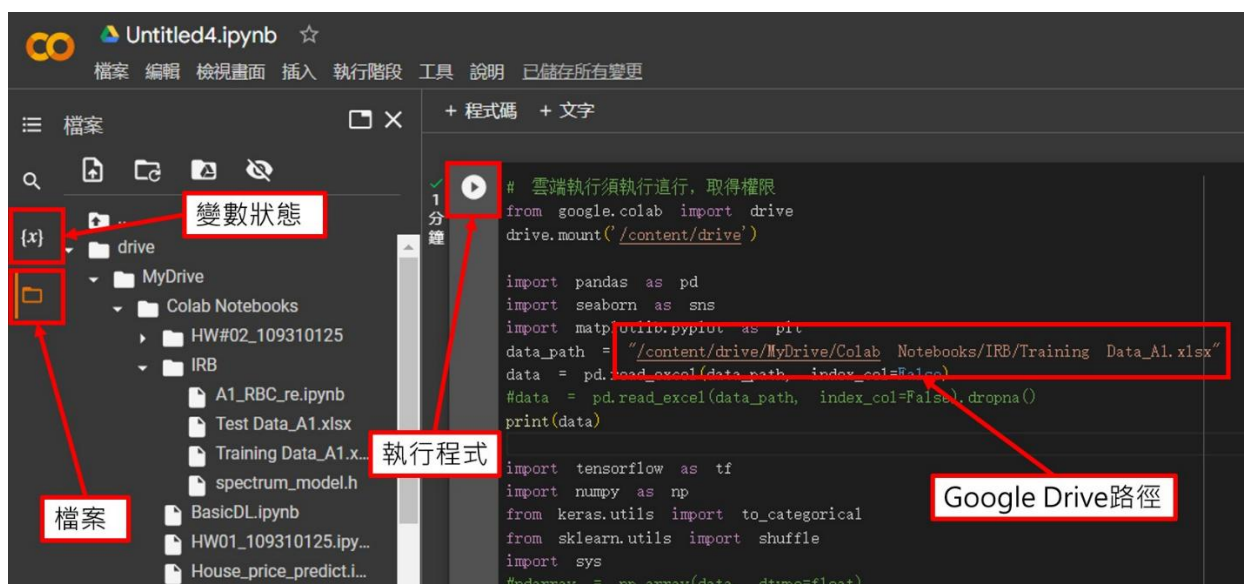
### I. 模型訓練

建立環境：Colab

其優點為可連結 Google Drive 使用的 IDE、不需自己建置環境、適合新手使用。

若需更加專業的環境，請 google 查詢 "Anaconda Jupyter Notebook" 建置虛擬環境。

從上方工具列：執行階段>變更執行階段類型 可更改成不同硬體(CPU、GPU、TPU)



資料集：資料夾(train\_data)

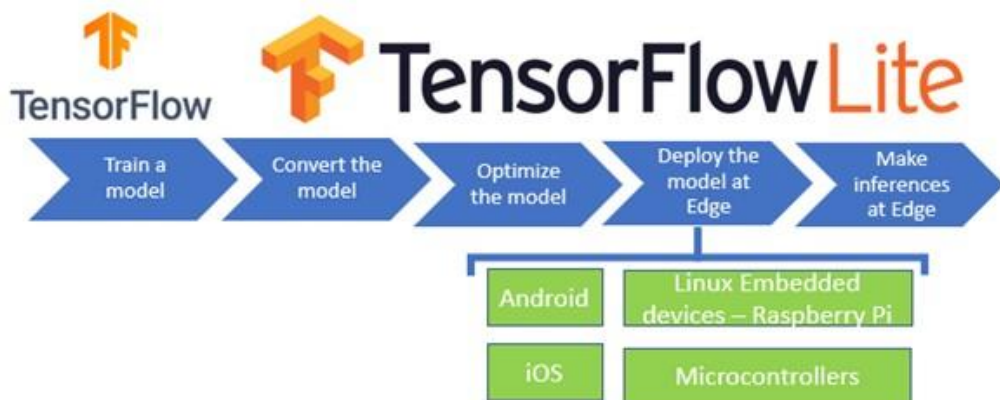
程式碼：PPG\_DNN.ipynb

獲取 "model.h" 進行部署

## II. 模型部署

### A. TensorFlow Lite

TensorFlow Lite(TFLite)是由 Google 開發的開源深度學習框架 TensorFlow 的輕量級版本，可以在行動裝置、嵌入式裝置和邊緣裝置上執行機器學習模型。主要目標是使機器學習模型能夠在資源有限的環境中有效運行，包括智慧型手機、物聯網設備、嵌入式系統和其他邊緣設備。

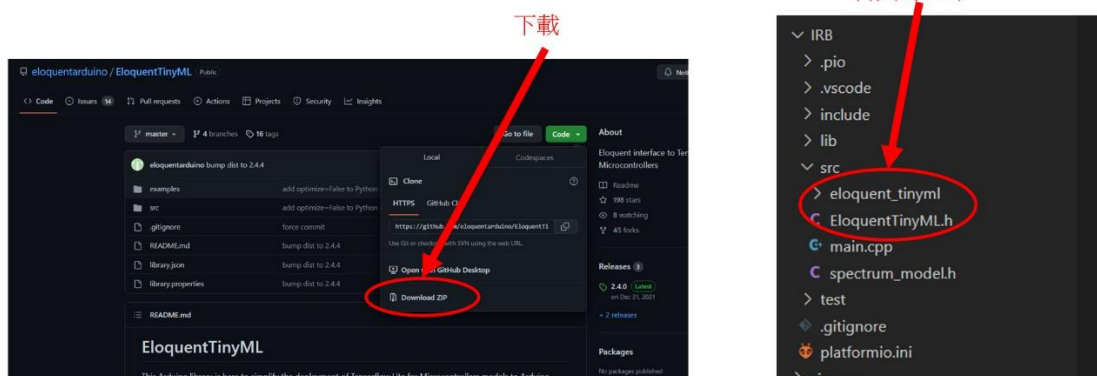


### B. 使用函示庫 >> EloquentTinyML.h

因為 PlatformIO 上的版本太舊，所以請自行到 Github 下載，並按照下圖說明安裝。

Github : <https://github.com/eloquentarduino/EloquentTinyML>

Download the library we need



### C. Sine 程式範例教學

<https://alankrantas.medium.com/%E6%9A%A2%E6%89%80%E6%AC%B2%E8%A8%80%E7%9A%84-tinyml-%E4%BD%BF%E7%94%A8-eloquenttinyml-%E8%B6%85%E8%BC%95%E9%AC%86%E4%BD%88%E7%BD%B2-tensorflow-lite->

[%E7%A5%9E%E7%B6%93%E7%B6%B2%E8%B7%AF%E6%A8%A1%E5%9E%8B%E5%88%B0%E5%BE%AE%E6%8E%A7%E5%88%B6%E5%99%A8%E4%B8%8A-%E5%A6%82-esp32-%E5%8F%8A-arduino-nano-33-ble-75900f5f9fb9](#)

程式碼：

```
#include <Arduino.h>
#include "EloquentTinyML.h"
#include "eloquent_tinymml/tensorflow.h"
#include "sine_model.h" // TinyML 模型
#define NUMBER_OF_INPUTS 1
#define NUMBER_OF_OUTPUTS 1
#define TENSOR_ARENA_SIZE 2 * 1024 // 模型使用記憶體大小
Eloquent::TinyML::TensorFlow::MutableTensorFlow<NUMBER_OF_INPUTS, NUMBER_OF_OUTPUTS,
TENSOR_ARENA_SIZE> tf;

void setup() {
    Serial.begin(115200);

    tf.addBuiltinOp(BuiltinOperator_FULLY_CONNECTED, Register_FULLY_CONNECTED(), 1, 9);

    tf.begin((unsigned char*) model_data); // 匯入模型
}

void loop() {

    // 隨機產生 x 和 y 當預測資料
    float x = 3.14 * random(101) / 100;
    float y = sin(x) * cos(x);
    float input[1] = {x};
    float predicted = tf.predict(input);
    Serial.print("Data: f(");
    Serial.print(x);
    Serial.print(") = ");
    Serial.print(y);
    Serial.print("\t predicted: ");
    Serial.println(predicted);

    delay(100);
}
```

#### D. 部署範例程式(練習用，自行 debug)

```
#include "EloquentTinyML.h"
#include "eloquent_tinymml/tensorflow.h"
#include "spectrum_model.h"

#define NUMBER_OF_INPUTS 25
#define NUMBER_OF_OUTPUTS 2
#define TENSOR_ARENA_SIZE 4 * 1024 // 模型使用記憶體大小
Eloquent::TinyML::TensorFlow::MutableTensorFlow <NUMBER_OF_INPUTS, NUMBER_OF_OUTPUTS,
TENSOR_ARENA_SIZE> tf;

void setup() {
```

```

Serial.begin(115200);

tf.addBuiltinOp(BuiltinOperator_FULLY_CONNECTED, Register_FULLY_CONNECTED(), 1, 9);
tf.addBuiltinOp(BuiltinOperator_SOFTMAX, Register_SOFTMAX(), 1, 1);
tf.addBuiltinOp(BuiltinOperator_LOGISTIC, Register_LOGISTIC(), 1, 1);

tf.begin((unsigned char*) model_data);

float test_list[25] = {0.0, 0.003030717, 0.018056406, 0.007386915, 0.0, 0.001869699, 0.000449, 0.000718,
0.006982542, 0.016715601, 0.024452891, 0.028154038, 0.047208875, 0.047339678, 0.033032645,
0.029521797, 0.03147052, 0.026025888, 0.033922244, 0.060511492, 0.07208954, 0.056529667, 0.035075348,
0.06048418, 0.022952799};

Serial.printf("Std features = ");
/*for(int i = 0; i < NOP; i++){
    if(i % 3 == 0){
        std_features[i / 3] = (float) std_spec[i];
        Serial.printf("%.6f ", std_features[i / 3]);
    }
}*/
float prediction[2] = {};

tf.predict(test_list, prediction);
//tf.predict(std_features, prediction);
Serial.printf("\nReslut of predict : %.2f, %.2f", prediction[0], prediction[1]);
}

void loop() {
}

```